

Efficient privacy preserving top-k recommendation using homomorphic sorting

Pranav Verma, Anish Mathuria, and Sourish Dasgupta

Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar.
{pranav_verma, anish_mathuria, sourish_dasgupta}@daiict.ac.in

Abstract. The existing works on privacy-preserving recommender systems based on homomorphic encryption do not filter top-k most relevant items on the server side. As a result, sending the encrypted rating vector for all items to the user retrieving the top-k items is necessary. This incurs significant computation and communication costs on the user side.

In this work, we employ private sorting at the server to reduce the user-side overheads. In private sorting, the values and corresponding positions of elements must remain private. We use an existing private sorting protocol by Foteini and Olga and tailor it to the privacy-preserving top-k recommendation applications. We enhance it to use secure bit decomposition in the private comparison routine of the protocol. This leads to a notable reduction in cost overheads of users as well as the servers, especially at the keyserver where the computation cost is reduced to half. The dataserver does not have to perform costly encryption and decryption operations. It performs computationally less expensive modular exponentiation operations. Since the private comparison operation contributes significantly to the overall cost overhead, making it efficient enhances the sorting protocol's performance. Our security analysis concludes that the proposed scheme is as secure as the original protocol.

1 Introduction

A recommender system (RS) provides personalized suggestions to users based on the previous behavior of users on similar items. For example, a news website will analyze the kind of news a user has been reading for some time and it shows more relevant articles for that individual. It helps users of that platform find items of interest among the plethora of other available items. Recommender systems are at the core of almost all major web applications in domains like social media, shopping websites, entertainment, news, etc. They track individuals' activities, which items a user is frequently viewing, how much time they spend on a particular web page, purchases after looking at items and how they react to suggested items. Such systems monitor every user activity and maintain a user profile based on this data. These user profiles are highly valued by online businesses. If a company that sells medicines or health insurance learns that a user has been searching for symptoms related to a particular disease, it may want to target that user for selling its products.

A recommender system helps the users find the relevant products, but often at the cost of revealing their personal information. Many users may not agree to this

trade-off and would like to use the recommendations facilities without exposing their personal information to unknown parties (data servers). A solution to this problem is to use privacy-preserving recommendation systems (PPRS) which keep user data hidden from the service providers and give relevant recommendations to the user. As noted by Rosinosky *et al.* [24] there are three broad categories of privacy-preserving recommender systems: decentralized or federated solutions that let users compute similarity in decentralized manner, obfuscation-based approaches where user ratings are masked with noise and the encryption-based approach, where homomorphic cryptosystems are used to perform computation on encrypt user ratings.

In [1] Badsha *et al.* proposed collaborative and content-based filtering approaches to generate the recommendations using ElGamal partially homomorphic encryption scheme. In [16], Arjan *et al.* proposed a familiarity-based recommender system that uses social network connections to generate recommendations efficiently. Cryptorec, proposed by Wang *et al.* [28], uses matrix factorization to train a model over ratings encrypted under the fully homomorphic scheme (FHE) scheme. In the cross-domain recommendation system, Oguneseyi *et al.* [21] proposed a protocol with BGV [5] as the underlying FHE encryption scheme. Their protocol works with data from multiple vendors having some common items to improve individual vendors' recommendation accuracy without leaking user information to other vendors. A collaborative filtering-based protocol that uses the BGV scheme with packing was proposed by Jumonji *et al.* [17].

Recently, Zhengqiang *et al.* [14] proposed to convert ratings into word vectors and find the top-k most relevant items for individual users. They use a model-based prediction scheme that is trained on the user data first. As noted by Erez and Tamir [26] and Michael *et al.* [12] the model-based recommender systems are less accurate as compared to item-based collaborative filtering schemes for top-k item recommendations.

Problem statement A real world recommender system usually has several thousand items, but users expect to receive very few items as recommendations. In existing PPRS schemes [13], [6], [15] etc., the RS sends to the user the entire rating vector that consists of encrypted rating predictions for all the items i.e., thousands of ciphertexts. These ciphertexts are decrypted and top-k predictions are simply filtered at the users' end. The communication cost considering thousands of ciphertexts every time the user asks for recommendations is inefficient, followed by decryption that requires significant time and computing power. We refer to this requirement as the private top-k recommendation problem; the server should send only top-k most relevant items to the user to reduce the user-side cost overhead.

Our approach To solve the top-k problem, we propose selecting the top-k items at the recommendation server itself so that the rating vector sent to the users only contains k ciphertexts corresponding to the items with the highest predicted ratings. The first challenge here is that selecting top-k items at the server must be done over the encrypted predicted ratings. The second challenge is that the server should not learn which items are chosen as the top-k recommendations as this violates the

fundamental goal of user privacy. We use homomorphic encryption based private sorting as the basis of our solution. It allows us to compare two encrypted values and eventually sort the entire vector without revealing the order of elements to the server.

Two-server private sorting model Foteini and Olga [2] proposed a partial homomorphic encryption based private sorting scheme. Their scheme uses two servers: a DataServer (S_1) and a KeyServer (S_2). The KeyServer generates the required keys and shares them with the appropriate parties. The user encrypts the data and stores them on the DataServer, the two servers then execute the private sorting protocol and sort the data elements without either party learning the data values or the relative positions of the elements. Both servers are assumed to be semi-honest, meaning that they follow the protocol and do not collude with anyone while they can try to learn additional information during the protocol execution.

We adapt the two-server private sorting model to our setting. We note that the Foteini-Olga protocol uses the DGK private comparison protocol [10] as a building block. The DGK protocol compares two inputs by encrypting them bit-by-bit, this step increases the number of ciphertexts exchanged between the two servers as well as the computation overhead. The core of the DGK protocol is that given two l -bit inputs a and b , if we compute $z = 2^l + a - b$ then, the most significant bit of z would be 1 if $a \geq b$ and 0 otherwise. In case the inputs are encrypted with homomorphic encryption scheme, computing encrypted z is not difficult. As an alternative to the DGK protocol, we use a secure bit decomposition scheme.

Contributions Private sorting using homomorphic cryptosystem has been used in many applications, for example, in [9] authors used private sorting for resource allocation, in [18] authors proposed to design graph database for privacy-preserving social searches. We are not aware of any previous work on application of private sorting to recommender systems. We list our contributions as bellow.

- Our proposed enhancement of Foteini-Olga protocol does not require the dataserver to perform encryption whereas in Foteini-Olga protocol $\mathcal{O}(\ell)$ encryptions were needed.
- The cost overhead at the keyserver is reduced similarly, in the original protocol the keyserver is required to perform $\mathcal{O}(\ell)$ encryptions and decryptions, in our modified version only $\mathcal{O}(\ell)$ decryptions are needed.
- In terms of communication costs, our proposal reduces the communication cost between two servers by a factor of one-third.
- We implement both the Foteini-Olga protocol (the original paper [2] does not provide experimental results) and our modified version, and show that our enhancement outperforms the original protocol.
- We show the formal security analysis of SBD protocol proposed in [25] that was missing in original work.

Our proposed modification based on SBD is more efficient for the following reasons. First, the DGK protocol uses secret sharing where the information is divided into parts and shared by multiple parties and they jointly execute a protocol. Whereas

Symbol	Description
n	RSA modulus
g	Chosen randomly from \mathbb{Z}_{n^2}
\mathbb{Z}_n	Group under addition modulo n
\mathbb{Z}_p^*	Group under multiplication modulo a prime p
\in_r	Choosing a value randomly
pk, sk	Public and private key for generalised Paillier
$[m]$	Encryption of m with basic Paillier scheme
$\llbracket m \rrbracket$	Encryption of m with generalized Paillier scheme
$\ m\ $	Encryption of m with quadratic residuosity cryptosystem
d_l	l^{th} bit of integer d

Table 1: Notations

in SBD protocol, information is hidden by masking original values with a random noise and sharing it with the other party. Second, in SBD a substantial amount of work can be done in pre-computation phase. For example, the SBD protocol requires encryption of 0 bit and 1 bit ; these can be pre-computed.

Organisation of the paper The next section reviews the cryptographic building blocks that are used in this paper. In section 3 we revisit the working of the Foteini-Olga protocol. Section 4 proposes a private comparison protocol using SBD. We also give the efficiency and privacy analysis of our protocol. The experimental results are discussed in section 5. We conclude the paper in section 6.

2 Preliminaries

Here we will discuss about generalized Paillier cryptosystem that is used in Foteini-Olga protocol. The working of DGK private comparison scheme and secure bit decomposition protocol. The symbols used throughout the rest of the paper are listed in Table 1.

2.1 Generalised Paillier cryptosystem

It is a probabilistic public key encryption scheme, introduced by Damgard and Jurik [11], where $\text{mod } n^2$ in basic Paillier [22] is replaced with $\text{mod } n^{s+1}$, and the plaintext space \mathbb{Z}_n is replaced with $\mathbb{Z}_{n^{s+1}}^*$, where $s \geq 1$ is a layer of Paillier.

Key generation: Choose security parameter k and compute k -bit RSA modulus $n = pq$. Choose an element $g \in_r \mathbb{Z}_{n^{s+1}}$ such that $g = (1 + n)^j x \text{ mod } n^{s+1}$, for j relative prime to n and $x \in_r \mathbb{Z}_n^*$. Compute $\lambda = \text{lcm}(p-1, q-1)$ and choose d such that $d \text{ mod } n \in_r \mathbb{Z}_n^*$ and $d = \lambda \text{ mod } 0$. Here, n, g are public key and d is the private key.

Encryption: Choose a plaintext $i \in \mathbb{Z}_{n^s}$, take a random number $r \in_r \mathbb{Z}_{n^{s+1}}$, then the ciphertext is:

$$[[i]] = g^i r^{n^s} \bmod n^{s+1}$$

Decryption: Given a ciphertext $[[i]]$

$$\begin{aligned} & [[i]]^{d \bmod n^{s+1}}, \text{ here} \\ [[i]]^d &= \left(g^i r^{n^s} \right)^d = (1+n)^{jid \bmod n^s} \left(x^i r^{n^s} \right)^{d \bmod \lambda} \\ &= (1+n)^{jid \bmod n^s} \end{aligned}$$

In [11], an algorithm is shown to retrieve $jid \bmod n^s$ from $(1+n)^{jid \bmod n^s}$ efficiently. Similarly replacing i with g results into $jd \bmod n^s$, using these two values the message i can be retrieved as

$$i = (jid)(jd)^{-1} \bmod n^s$$

Homomorphic property: The following holds for both Paillier and generalised Paillier, say we have two messages m_1, m_2 and a constant c , then

$$[m_1 + m_2] = [m_1][m_2], \quad [cm_1] = [m_1]^c$$

$$[[m_1 + m_2]] = [[m_1]][[m_2]], \quad [[cm_1]] = [[m_1]]^c$$

The following additional homomorphic property holds for generalised Paillier.

$$[[[m_1][m_2]]] = [[m_1 + m_2]] = [[m_1]]^{[m_2]}$$

2.2 Goldwasser-Micali cryptosystem

Proposed by Goldwasser and Micali in 1982, it is based on quadratic residuosity (QR) problem. In QR problem, given x, N such that N is RSA modulus, it is difficult to determine if $x = y^2 \bmod N$ for any y when the Jacobi symbol of $x = 1$. On the other hand it is easy if the factors of N are known.

Key generation: Similar to RSA modulus, randomly choose two large primes p, q and compute $N = pq$. Next find x such that the Jacobi symbol $\left(\frac{x}{N}\right) = 1$, by choosing random numbers and testing. The public key is consist of (x, N) and the secret key (p, q) .

Encryption: The message m has to be in represented in bits for encryption. A bit m_i can be encrypted using the public key and a random y_i such that $gcd(y_i, N) = 1$. The ciphertext is:

$$c_i = y_i^2 x^{m_i} \bmod N$$

Decryption: To decrypt a c_i , we need factors of N as p, q . Determine if the c_i is a quadratic residue by calculating Jacobi symbol over c_i . If it is a quadratic residue, then $m_i = 0$ otherwise $m_i = 1$.

Homomorphic property: Given two ciphertexts $\|m_1\|, \|m_2\|$ corresponding to two messages m_1, m_2 :

$$\|m_1 \oplus m_2\| = \|m_1\| \|m_2\|$$

2.3 DGK private comparison

It is a secure two party computation protocol proposed by Damgård, Geisler and Krøigård [10]. In the basic version of the protocol, each party has a secret number and they want to compare their secret inputs without telling the actual value to the other party. There are other variations of the protocol depending on which party learns the comparison result [4]. DGK protocol uses a modified Paillier scheme, and it can be realized using Goldwasser-Micali scheme as well. For this discussion we consider two parties A and B with private ℓ -bit integer inputs x and y respectively. The key pair is (pk, sk) , party B has the private key sk . At the end, A learns the result of $x < y$ in under encryption while B does not learn about the relation. The notation $\|x_i\|$ denotes the encryption of i th bit of x and t is the security parameter used in key generation. Table 2 follows the protocol description in [27]. In step-3, A chooses the variable δ_A to restrict B from learning about the relation, to compute λ , knowledge of δ_A and δ_B is necessary and B only knows δ_B in plain and never gets δ_A . Similarly in step-6 A receives $\|\delta_B\|$ so that it learns λ under encryption only. In step-4, the value c_i will only be 0 when $x_j = y_j$ for each $j, i < j < \ell$ and at the same time $x_i \neq y_i$.

In the protocol we can see B has to perform ℓ encryptions in the first step. A has to perform ℓ modular multiplications in step-2, and blind ℓ elements before sending them to B in step-5. Finally, in the last step B performs ℓ decryptions to learn the comparison result.

2.4 Batcher's sort

Batcher proposed an odd-even merge sort [3]. The sorting scheme can be divided into two parts *odd-even_merge()* and *odd-even_mergesort()*. Table 3 and 4 shows the pseudocode for both routines.

2.5 Secure bit decomposition (SBD)

In the SBD protocol, there are two parties: Alice, who has a public key pk , and Bob who has a m bit integer x encrypted with pk . They both jointly run the SBD protocol and at the end Bob receives encrypted bits of $x = \langle E(b_0), E(b_1), \dots, E(b_{m-1}) \rangle$ whereas Alice learns nothing about x or its bits. We use the protocol proposed by Samanthula et al. [25], which is based on Paillier's additive homomorphic encryption scheme. The protocol is divided into two phases: *Encrypted_LSB()* computes the least significant bit of an encrypted integer input and *SVR()* verifies if the bits representation

Input: Party $A: x, B: y$ and $sk, 0 < x, y < 2^\ell$
Output: A learns $\ \lambda\ , \lambda \in \{0, 1\}$ such that $\lambda = 1$ if $x \leq y$ and $\lambda = 0$ otherwise
<ol style="list-style-type: none"> 1. B sends the encrypted bits $\ y_i\ , 0 < i < 2^\ell$ to A 2. For each $i, 0 < i < 2^\ell$, A computes $\ x_i \oplus y_i\$ as follows: $\ x_i \oplus y_i\ = \begin{cases} \ y_i\ , & \text{if } x_i = 0 \\ \ 1\ \ y_i\ ^{-1} \bmod n & \text{otherwise} \end{cases}$ 3. A chooses a uniformly random bit δ_A and computes $s = 1 - 2\delta_A$ 4. For each $i, 0 < i < 2^\ell$, A computes $\ c_i\ = \ s\ \ x_i\ \ y_i\ ^{-1} \left(\prod_{j=i+1}^{\ell-1} \ x_j \oplus y_j\ \right)^3 \bmod n$ 5. A blinds $\ c_i\$ by raising them to a random exponent r_i of $2t$ bits: $\ c_i\ = \ c_i\ ^{r_i} \bmod n$, and sends them in random order to B 6. B checks whether one of the numbers c_i is decrypts to zero. If there is one, $\delta_B = 1$ else $\delta_B = 0$ 7. B sends $\ \delta_B\$ to A where it computes $\ \lambda\ = \ \delta_A \oplus \delta_B\$

Table 2: DGK private comparison protocol [27]

Input: Sequence a_0, a_1, \dots, a_{n-1} of length $n > 1$ whose two halves $a_0, a_1, \dots, a_{n/2-1}$ and $a_{n/2}, \dots, a_{n-1}$ are sorted (n is power of two)
Output: The sorted sequence
<p>if $n > 2$ then:</p> <ul style="list-style-type: none"> - apply <i>odd-even_merge</i>($n/2$) recursively to the even subsequence a_0, a_2, \dots, a_{n-2} and to the odd subsequence a_1, a_3, \dots, a_{n-1} - comparison $[i : i + 1]$ for all $i \in \{1, 3, \dots, n - 3\}$ <p>else:</p> <ul style="list-style-type: none"> - comparison $[0 : 1]$

Table 3: Batcher sort: odd-even_merge()

Input: Sequence a_0, a_1, \dots, a_{n-1} (n is power of two)
Output: The sorted sequence
<p>if $n > 1$ then:</p> <ul style="list-style-type: none"> - apply the <i>odd-even_mergesort</i>($n/2$) recursively to the two halves $a_0, a_1, \dots, a_{n/2-1}$ and $a_{n/2}, \dots, a_{n-1}$ of the sequence - <i>odd-even_merge</i>(n)

Table 4: Batcher sort: odd-even_mergesort()

is correct for the corresponding integer (under encryption). Both these sub-routines cost Alice and Bob $\mathcal{O}(\log n)$ communication. Algorithm 1 shows the working of the protocol between the dataserver and the keyserver, the details of SVR() is given in the appendix A.

Input : Common input: Keyserver's public key pk , and modulus N
Keyserver's input: private key: sk
Dataserver's input: Encrypted $[x]$, $0 \leq x < 2^m$
Output: Dataserver learns: $\bar{x} = \langle [b_0], [b_1], \dots, [b_{m-1}] \rangle$, and the keyserver learns nothing

1. $\ell = 2^{-1} \bmod N$
2. $T = [x]$
3. for $i = 0$ to $m - 1$ do:
 - $[x_i] \leftarrow \text{Encrypted_LSB}(T, i)$
 - $Z \leftarrow T[x_i]^{N-1} \bmod N^2$
 - $T = Z^\ell \bmod N^2$
4. end for
5. $\gamma \leftarrow \text{SVR}(E(x), \langle [b_0], [b_1], \dots, [b_{m-1}] \rangle)$
6. if $\gamma = 1$ then
 - return
7. else
 - go to step 2
8. end if

Algorithm 1: Secure bit decomposition protocol

First, the Encrypted_LSB() routine takes two inputs: a ciphertext of an encrypted integer and an integer in plaintext, and returns the encrypted least significant bit of the encrypted integer passed to it. The two observations that this sub-protocol follows are:

Observation-I For any given x , let $y = x + r \bmod N$, where r is a random number in \mathbb{Z}_n . Here the relation between y and r depends on whether $x + r \bmod N$ leads to an overflow or not. y is always greater than r if there is no overflow. Similarly, in the case of overflow y is always less than r .

Observation-II For any given $y = x + r \bmod N$, where N is odd, the following property regarding the least significant bit of x always hold:

$$x_0 = \begin{cases} \lambda_1 \oplus \lambda_2, & \text{if } r \text{ is even} \\ 1 - (\lambda_1 \oplus \lambda_2), & \text{otherwise} \end{cases}$$

Here λ_1 denotes whether an overflow occurs or not, and λ_2 denotes whether y is odd or not. That is $\lambda_1 = 1$ if $r > y$, and 0 otherwise. Similarly, $\lambda_2 = 1$ if y is odd and 0 otherwise, \oplus denotes the XOR operation.

Input : Dataserver's input: Encrypted integer: T , and plain iteration number: i

Keyserver's input: private key: sk

Output: Dataserver learns: Encrypted least significant bit of T as $[x_i]$

1. Dataserver:
 - $r \in_r \mathbb{Z}_N$
 - $Y = [r_i]T \bmod N^2$
 - Send Y to keyserver
2. Keyserver:
 - $y = \text{decrypt}(Y)$
 - If y is even: $\alpha = [0]$
 - Else: $\alpha = [1]$
 - Send α to dataserver
3. Dataserver:
 - If r is even: $[x_i] = \alpha$
 - Else: $[x_i] = [1]\alpha^{N-1} + 1 \bmod N^2$
4. Return $[x_i]$

Algorithm 2: Secure bit decomposition sub-protocol: Encrypted_LSB()

3 Private sorting

The private sorting is a well known problem where a user uploads encrypted data on a cloud server and asks the server to perform sorting without decryption. The privacy requirement is that the server must not learn the relative order between any element as well as the values of those elements. Homomorphic encryptions allow computation over encrypted data, and thus can be used in designing private sorting protocols. The existing private sorting schemes such as [8], [7], [20] proposed FHE based sorting schemes that are better than some previous approaches. A major drawback of FHE-based schemes is the computation cost and noise growth in the ciphertext, especially when the multiplicative depth is more. In [8], the average time to sort 40 elements is around 1400 sec using the Lazy sort method. In [20], authors compared the performance of sorting based on comparison done using integer-wise and bit-wise operations. Here authors proposed a polynomial interpolation-based approach to compare two integers encrypted with FHE. They concluded that though the bit-by-bit comparison performs more operations than integer-wise comparison, the practical time required is lesser in bit-wise element comparison. A better scalable solution was proposed in [7], in this work authors proposed a polynomial rank sort scheme using BGV protocol. Their results show better scalability and it took 865 seconds to sort 60 items (in their test environment).

A private sorting scheme that uses a partial homomorphic scheme is shown in [2], it meets the functional requirements and can perform over a large number of items. It has a semi-trusted third party as a keyserver that helps cloud server during the private sort operation. To the best of our knowledge this is the only private sorting scheme that is based on PHE.

3.1 Foteini-Olga private sorting protocol

The protocol consists of several sub-routines that are executed jointly by both the servers. The protocol uses the Batcher's sorting scheme [3]. It is a comparison-based algorithm that compares two elements at a time and gradually sorts the entire array. The basic features of the scheme are that it is highly parallelizable and its performance does not depend on the element values. The important sub-routines used in the private sorting process are EncSelect(), EncPairSort(), EncCompare() and EncSort(). In this section we describe EncCompare() in detail and given the functionalities of each sub-routine. For a complete description we refer the reader to the original paper [2].

EncSelect() This function takes three inputs, two encrypted integers $[a], [b]$ and an encrypted boolean $\llbracket v \rrbracket$. Based on the value of v it returns either re-encrypted $\llbracket [a] \rrbracket$ or $\llbracket [b] \rrbracket$. In other words, the ciphertext is different than what was given as input so that the output does not leak the relationship between the input values.

$$\text{EncSelect}([a], [b], \llbracket v \rrbracket) = (\llbracket 1 \rrbracket \llbracket v \rrbracket^{-1})^{\llbracket a \rrbracket} \llbracket v \rrbracket^{\llbracket b \rrbracket} = \llbracket (1 - v)[a] + v[b] \rrbracket = \llbracket [c] \rrbracket$$

We can see here that the output is $\llbracket [c] \rrbracket$, to get $[c]$ from this, there is another sub-routine StripEnc(). S_1 masks it with a random noise e and sends masked $\llbracket [c + e] \rrbracket$ to S_2 , who replies with $[c + e]$. The server S_1 un.masks and gets $[c]$. For simplicity we represent the output of EncSelect() as $[c]$ henceforth.

EncPairSort() In this routine as shown in Table 5, S_1 has PK_P , two encrypted integers $([a], [b])$. After execution S_1 will receive a pair of ciphertexts $([c], [d]) = ([a], [b])$ if $a \geq b$ otherwise $([c], [d]) = ([b], [a])$. Basically the two numbers are arranged in sorted order without revealing the values a, b or their relative order. First, S_1 executes EncCompare() to obtain $\llbracket v \rrbracket$, then it executes EncSelect() twice to obtain the ciphertext pair $([c], [d])$. To sort the entire input array, both servers will jointly execute the pair sort function iteratively.

$S_1 (PK_P, [a], [b])$	$S_2 (SK_P)$	operation
$\llbracket v \rrbracket \leftarrow$	EncCompare($pk, [a], [b]$)	$v = 1$ if $a \geq b$ and 0 otherwise
$[c] \leftarrow$	EncSelect($[a], [b], \llbracket v \rrbracket$)	$c = (1 - v)a + vb$
$[d] \leftarrow$	EncSelect($[a], [b], \llbracket 1 \rrbracket \llbracket v \rrbracket^{-1}$)	$d = va + (1 - v)b$

Table 5: EncPairSort() method [2]

EncSort() In this routine as shown in Table 6, S_1 uses Batcher's sorting scheme [3] (see appendix 2.4) to sort the encrypted elements with the help of S_2 . Here the encrypted values to be sorted are stored in an array A and at each level of the sorting

$S_1 (PK_P, A)$	$S_2 (PK_P, SK_P)$	Operation
$A = [[a_1], [a_2], \dots, [a_N]]$		
1: $A_i \leftarrow A$		
2: for $i \in \{1, \dots, k-1\}$		$k = (\log N)^2, N = A$
3: set of element pairs $S \leftarrow \text{pairs}_i.A$		i^{th} level of Batchers sort
4: while ($S \neq \perp$)		
5: $(x, y) \in S$		next pair of A_i to sort
6: $(A_{i+1}[x], A_{i+1}[y]) \leftarrow \text{EncPairSort}(pk, A_i[x], A_i[y])$		
7: $S \leftarrow S \setminus \{(x, y)\}$		remove (x, y) from S
8: //end while		
9: //end for		
10: $B \leftarrow A_k$		

Table 6: EncSort() method [2]

process the pairs are chosen in a sub-array where they are compared and sorted using EncPairSort(). In Table 6, A is the input array that is copied into A_1 . Then for each level $\text{pairs}_i.A$ stores the element pairs in a set S . Next we take one pair (x, y) at a time and run EncPairSort() till all the array elements are sorted in pairs. This results into an array A_2 , where the pairing is now done by combining two elements at consecutive odd indices, similarly the even indices elements are paired together. These pairs are passed into EncPairSort() one at a time and the cycle repeats till all the pairs are processed. Finally the sorted array is stored in B as output. EncPairSort() calls EncCompare() and EncSelect() to sort the pair as discussed earlier. An example of the sorting process is given below.

Example: We will take the example shown in [2] and to simplify the notation we write the inputs as plaintexts. Let the given input array be $A = [5, 1, 2, 9]$. First it will be copied into A_1 then in the first iteration pairs_1 will be $\{(5, 1), (2, 9)\}$ (index 1,2 and 3,4). It will be passed to EncPairSort() function and the resultant array will be $A_2 = [1, 5, 2, 9]$, Next, $\text{pairs}_2 = \{(1, 2), (5, 9)\}$, (index 1,3 and 2,4) that will produce $A_3 = [1, 5, 2, 9]$. Finally, in the last iteration, the $\text{pair}_3 = \{(5, 2)\}$ (index 2,3) will be passed to EncPairSort() and the result will be a sorted array $A_4 = [1, 2, 5, 9]$. All numbers will be encrypted and S_1 performs swapping using EncSelect(). After every iteration, the ciphertext will change so that S_1 cannot infer the order.

The following definition by Foteini and Olga captures the functionality of EncSort().

Definition 1. *An encrypted sorting functionality EncSort() takes as input a public/secret key pair (PK_P, SK_P) of a semantically secure cryptosystem and an array $A = [[v_i]]$, $i \in \{1, N\}$ of N elements encrypted with PK_P . Let π be a permutation of indices 1 to N that corresponds to the indices of A 's elements sorted using its unencrypted values v_i . Then the output if EncSort() is an array $B = [[(v'_j)]]$, $j \in \{1, N\}$, where $v'_j = v_{\pi(i)}$ and $i \in \{1, N\}$.*

Key-value sort() The steps of EncPairSort() can be applied to a key-value pair based array of elements where sorting is done based on *value*. In PPRS the *value* will be

user rating and *key* will be corresponding item index. The elements in a rating vector are in the form of a tuple with each element encrypted individually. For example, the tuple $\langle [a], [i] \rangle$ indicates that rating a was assigned to an item at index i . To implement this, only change is required that `EncSelect()` function is executed not only on *value* but on *key* as well. The rest of the steps remain the same as shown in Table 7. Here, based on $\llbracket v \rrbracket$ indices will be updated to $\llbracket i' \rrbracket, \llbracket j' \rrbracket = \llbracket i \rrbracket, \llbracket j \rrbracket$ or $\llbracket i' \rrbracket, \llbracket j' \rrbracket = \llbracket j \rrbracket, \llbracket i \rrbracket$.

$S_1 (PK_P)$ $(\langle [a], [i] \rangle, \langle [b], [j] \rangle)$	$S_2 (SK_P)$	operation
$\llbracket v \rrbracket \leftarrow$	<code>EncCompare</code> ($PK_P, [a], [b]$)	$v = 1$ if $a \geq b$ and 0 otherwise
$[c] \leftarrow$	<code>EncSelect</code> ($[a], [b], \llbracket v \rrbracket$)	$c = (1 - v)a + vb$
$[d] \leftarrow$	<code>EncSelect</code> ($[a], [b], \llbracket 1 \rrbracket \llbracket v \rrbracket^{-1}$)	$d = va + (1 - v)b$
$[i'] \leftarrow$	<code>EncSelect</code> ($[i], [j], \llbracket v \rrbracket$)	$i' = (1 - v)i + vj$
$[j'] \leftarrow$	<code>EncSelect</code> ($[i], [j], \llbracket 1 \rrbracket \llbracket v \rrbracket^{-1}$)	$j' = vi + (1 - v)j$

Table 7: Key-value sort method [2]

EncCompare() The `EncCompare()` is based on the observation that for any two ℓ -bit numbers a, b the most significant bit of $z = 2^\ell + a - b$ represents the relationship between a and b . The bit $z_\ell = 1$ if $a \geq b$ and $z_\ell = 0$ otherwise. As shown in Table 8, first S_1 computes and masks $[z]$ before sending it to S_2 . Then both servers jointly execute `DGKCompare()` and for this phase they use QR key pairs (PK_Q, SK_Q) . This step lets S_1 learn the QR-encrypted relation λ between d' and r' . The remaining steps are used to convert the QR-encrypted $\|\lambda\|$ to Paillier encrypted $\llbracket v \rrbracket$. The final output $\llbracket v \rrbracket$ is used in `EncSelect()` function to pick either a or b based on the value of v .

The call to `DGKCompare()` sub-routine requires multiple rounds of communication between the two servers. The communication cost between the servers is $\mathcal{O}(\ell k)$ for ℓ bit number and where k is a security parameter. The computation cost for both servers is $\mathcal{O}(\ell \log \ell)$ modular multiplications. A sorting algorithm needs to compare elements from the list very frequently and therefore use of a multi-round sub-routine greatly affects the efficiency of the sorting protocol. Although we may assume that the servers have enough computational power and are connected with high speed communication networks, the cost overheads limit the scalability of any protocol.

4 Enhancement using SBD

The most important routine in Foteini-Olga is `EncCompare()` that compares two encrypted inputs. It is jointly executed by both servers and the dataserver learns the encrypted result as the output. We propose to use secure bit decomposition (SBD) instead of `EncCompare()` in Foteini-Olga protocol. The dataserver can compute encrypted z and then execute SBD with the keyserver, the dataserver will learn encrypted bits of the z (including MSB) efficiently. We discuss the detailed steps below.

Information known to S_1: $PK_P, PK_P, [a], [b]$ Information known to S_2: PK_P, SK_P, PK_Q, SK_Q Output: S_1 learns $\llbracket v \rrbracket$, where $v = 1$ if $a \geq b$ else $v = 0$	
Dataserver (S_1)	Keyserver (S_2)
$[z] = [2^\ell][a][b]^{-1} \bmod n^2$ pick $r \in_r \{0, 1\}^{\ell+k}$ $[d] = [z][r] \bmod n^2$ $r' = r \bmod 2^\ell$ encrypt r_ℓ $\llbracket v \rrbracket = \llbracket d_\ell \rrbracket \llbracket r_\ell \rrbracket \llbracket \lambda \rrbracket$ choose $t \in_r \{0, 1\}$ $\llbracket s_t \rrbracket = \llbracket v \rrbracket \llbracket 0 \rrbracket, s_{t-1} = \llbracket v \rrbracket \llbracket 1 \rrbracket$ $\llbracket v \rrbracket = \llbracket s_t \rrbracket$	 decrypt $[d]$ $d' = d \bmod 2^\ell$ encrypt d_ℓ decrypt $\llbracket s_0 \rrbracket, \llbracket s_1 \rrbracket$ encrypt s_0, s_1
	$\xrightarrow{[d]}$ $\xleftrightarrow{\llbracket \lambda \rrbracket = \text{DGKCompare}(r', d')}$ $\xleftarrow{\llbracket d_\ell \rrbracket}$ $\xrightarrow{\llbracket s_0 \rrbracket, \llbracket s_1 \rrbracket}$ $\xleftarrow{\llbracket s_0 \rrbracket, \llbracket s_1 \rrbracket}$

Table 8: EncCompare() method [2]

In SBD protocol, various computations can be done offline. For example, the keyserver can generate several encryptions of bit 0 and bit 1 in advance and use them during the protocol run, it removes the encryption cost at runtime. As shown in appendix 1, in Encrypted_LSB() routine, the keyserver assigns value of α as encryption of 0 or 1, this operation is performed ℓ times for a ℓ -bit integer decomposition. This cost can be replaced by a simple read() operation if the keyserver has precomputed list of encrypted 0 and 1.

The SBD setting in [25] is similar to that in [2] with the keyserver and the dataserver. The dataserver that has two encrypted inputs, first computes $[z] = [2^\ell][a][b]^{-1} \bmod n^2$. Next it runs SBD protocol with the keyserver, that knows public and private keys. During the SBD execution, the keyserver does not learn value of z or the bit representation of it. The keyserver will however know the length of z , which is public information and does not affect the secrecy of user ratings. Once the dataserver learns the encrypted bits of z , it can recognize the most significant bit as $\llbracket v \rrbracket$ and send it to the next module as shown in Foteini-Olga protocol. The advantage of using SBD is that it is efficient than EncCompare(), it is compatible with Paillier cryptosystem and thus does not require additional key management (like in [2]). Comparing Table 8 and Table 9, we can notice that, apart from running DGK and SBD module, in EncCompare() there are additional rounds of communication required to compute $\llbracket v \rrbracket$ that is not needed in the modified scheme.

Information known to S_1: $PK_P, [a], [b]$	
Information known to S_2: PK_P, SK_P	
Output: S_1 learns $\llbracket v \rrbracket$, where $v = 1$ if $a \geq b$ else $v = 0$	
Dataserver (S_1)	Keyserver (S_2)
$[z] = [2^\ell][a][b]^{-1} \bmod n^2$ $\xrightarrow{[z], \text{SBD}([z], \text{pk}, \text{sk})} \xleftarrow{pk, sk} ([x_{\ell-1}, \dots, x_0])$ $\xleftarrow{([x_{\ell-1}, \dots, x_0])} \llbracket v \rrbracket = \llbracket x_{\ell-1} \rrbracket$	

Table 9: SBD based private comparison (SBDEncCompare())

4.1 Correctness

The correctness of the fact that if a and b are ℓ -bits integers, $x = 2^\ell + b - a$ is a $\ell - 1$ bit integer and its most significant bit (the $\ell + 1$ th) bit is 1 iff $a < b$ can be directly extended from the details shown in [4].

The SBD used here is a probabilistic protocol that with high probability produces correct output as encrypted bits of the input. The correctness depends on the security parameter (key length) of the cryptosystem and the length of the input integer in plaintext. The probability of protocol producing correct bit representation is given by following equation:

$$Pr = \left(1 - \frac{1}{2^{K-m}}\right)^m$$

Here, K is the keylength and m is bitlength of the input. In a PPRS application, the input length is 4 as most common applications use 10 or 5-star rating systems. The minimum Paillier keysize that is used in practice is 1024 bits. So with these parameters, the probability of correct output:

$$Pr = \left(1 - \frac{1}{2^{1024-4}}\right)^4 \approx 1$$

Thus there is negligible probability that the SBD protocol will fail in the proposed scheme.

4.2 Privacy analysis

The private comparison scheme must satisfy the following properties to protect user's privacy:

- It should not reveal the rating values (vector elements) to any server.
- It should not disclose the relationship between two ratings to any server.
- There must be no relation between ciphertexts at a particular index before and after the sorting is done. Basically none of the servers should know the position of any (encrypted) element before and after sorting the vector

An IND-CPA secure cryptosystem satisfies the first two requirements. The two ciphertexts are indistinguishable by definition of IND-CPA hence, the dataserver who only has access to the encrypted ratings cannot map a relationship between two ratings. At the same time it hides the actual values of each rating from the dataserver. The keyserver on the other hand, holds the private key but under the assumption that the two servers are non-colluding and they follow the protocol honestly prevents the keyserver to learn rating values of any item. In our modification, we are employing a secure bit decomposition protocol by Samanthula *et al.* [25]. They have shown that the keyserver does not learn anything about the encrypted input being decomposed. It does not learn any bit value after the decomposition as well. The dataserver only learns the encrypted bits and cannot infer any useful information from it.

The third requirement mentioned above is to prevent a semi-honest server from learning the relationship between two items' ratings. A semi-honest server will try to learn additional information by observing the available data, and if the ratings are sorted under encryption and a server can map those ratings with corresponding indices it will reveal that which items are rated higher than others. To decouple the item rating and corresponding index mapping, we must ensure that the rating and index ciphertexts are changed every time they are operated on. The homomorphic operations are performed over ratings and indices during the sorting process iteratively and corresponding ciphertexts are updated. The dataserver does not know the mapping between item with its index and hence cannot infer the relationship between the items.

4.3 Formal discussion

First we recall the sequential modular composition theorem given by Lindell and Yehida [19].

Theorem 1. *Let f_1, \dots, f_m be two-party probabilistic polynomial time functionalities and p_1, \dots, p_m protocols that compute respectively f_1, \dots, f_m in the presence of semi-honest adversaries.*

Let g be a two-party probabilistic polynomial time functionality and Π a protocol that securely computes g in the (f_1, \dots, f_m) -hybrid model in the presence of semi-honest adversaries.

Then Π^{p_1, \dots, p_m} securely computes g in the presence of semi-honest adversaries.

The authors of [2] and [4] have used the above theorem to show security of their protocols. They show it by building a simulator for each party and proving that their views are not distinguishable since the randomness added are from the same distributions. The security of the protocol depends on the secure computation of private comparison. The DGK protocol is a provably secure protocol for doing private comparison.

In our scheme we are suggesting to use secure bit decomposition protocol proposed by Samanthula *et al.* instead of the DGK protocol as the building block. In [25],

authors have shown in an informal way that the security of SBD depends on the security of Paillier cryptosystem. Here, we extend the same by showing views for both parties running the SBD protocol.

We say that the SBD protocol is secure if the views of both the participants (dataserver and keyserver) and the ones generated by a probabilistic polynomial time simulator that are computationally indistinguishable. We represent view of the participants as V_{DS}, V_{KS} for dataserver and keyserver respectively. Let PK_P, SK_P represent Paillier public and secret key-pair, a message m encrypted with PK_P is represented as $\llbracket x \rrbracket$, m is length of x in bits, r_i is a random number chosen from \mathbb{Z}_N , $\alpha \in \{0, 1\}$, U, V, W, Y are computed by the dataserver using homomorphic property of the cryptosystem, $\gamma \in \{0, 1\}$.

$$V_{DS} = (PK_P, \llbracket x \rrbracket, m, r_i, \llbracket \alpha \rrbracket, U, V, W, \gamma)$$

$$V_{KS} = (PK_P, SK_P, \llbracket Y \rrbracket, \llbracket W \rrbracket)$$

Now, we can build simulators S_{DS}, S_{KS} for the dataserver and keyserver respectively, and as discussed above if the view of the simulator is computationally indistinguishable from the respective views of the participants, we can claim that the protocol is secure. Given the input $PK_P, \llbracket x \rrbracket, m$, the dataserver's view can be build as follows:

- \tilde{r}_i is chosen randomly from \mathbb{Z}_N
- Generate $\llbracket \tilde{\alpha} \rrbracket \in_r \{0, 1\}$ randomly
- Generate $\tilde{U} \in_r \{0, 2^{m-1}\}$ since the input is m bit long
- Compute \tilde{V}, \tilde{W} using \tilde{U} and $\tilde{r}' \in_r \mathbb{Z}_N$
- Choose $\tilde{\gamma} \in_r \{0, 1\}$

Now, the view of the simulator S_{DS} will be

$$\widetilde{V}_{DS} = (PK_P, \llbracket x \rrbracket, m, \tilde{r}_i, \llbracket \tilde{\alpha} \rrbracket, \tilde{U}, \tilde{V}, \tilde{W}, \tilde{\gamma})$$

Next we can claim that $V_{DS} \equiv_c \widetilde{V}_{DS}$, meaning the two views are computationally indistinguishable. It is so because in the above transcript, \widetilde{V}_{DS} can be generated by taking the values with tilde from the same distribution as their respective counterparts in the V_{DS} . For example, in V_{DS} the value of γ is generated by the keyserver that is either 0 or 1, and in $\tilde{\gamma}$ in \widetilde{V}_{DS} is chosen from $\{0, 1\}$ and so are the other values. The Paillier cryptosystem is proven to be IND-CPA secure so the encryption of a random number from the given range, and the encryption of actual input in V_{DS} would be indistinguishable.

A similar argument holds for the keyserver, the simulator S_{KS} can be used to generate view \widetilde{V}_{KS} . These arguments show that the SBD protocol is secure. We therefore conclude that the proposed scheme is secure under the semi-honest model.

5 Performance analysis

The computational performance of the Foteini-Olga scheme heavily depends on the cost of `EncCompare()` routine that uses DGK private comparison protocol. In the

proposed scheme, we use SBD to reduce the cost for the private comparison step. The idea behind this modification is that the sorting operation requires frequent calls to the comparison function. Therefore making comparison efficient should lead to a significant performance gain in the overall sorting process.

Server	EncCompare()		SBDEncCompare()	
	Encryption	Decryption	Encryption	Decryption
S_1	$(\ell + 2)$ QR	-	-	-
S_2	$2 \text{ Pai} + (\ell + 1)$ QR	$2 \text{ Pai} + (\ell + 2)$ QR	-	$(\ell + 1)$ Pai

Table 10: Cost comparison with the existing scheme. ℓ : length of a rating value. QR, Pai: Enc/Dec operations under QR and Paillier cryptosystems respectively

Protocol	Server	Encryption	Decryption	ModExp	Comm b/w servers
EncCompare()	S_1	$\mathcal{O}(\ell)$	-	Const.	$(3\ell + 6)$ CT
	S_2	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell)$	Const.	
Modified scheme	S_1	-	-	$\mathcal{O}(\ell)$	$(2\ell + 1)$ CT
	S_2	-	$\mathcal{O}(\ell)$	-	

Table 11: Asymptotic cost comparison with the existing scheme, CT: ciphertext. ℓ : length of a rating value.

Table 11 shows the comparison between EncCompare() and our modified scheme. The computation cost at the dataserver (S_1) is reduced remarkably as the cost of encryption and decryption is much more than that of modular exponentiation. Unlike EncCompare(), in the proposed scheme the keyserver (S_2) requires only decryption operations since all the encryption operations can be pre-computed.

In terms of communication costs, our scheme requires 2/3 of the ciphertext exchanges between two servers required by the EncCompare() routine. It is important to note that these communications are required for each comparison operation, so to sort a vector with few hundreds of items, our scheme would save a lot of data exchange between servers.

It is evident from the comparison that the modified scheme reduces the communication cost significantly, as the comparison function is called multiple times in a sorting algorithm.

5.1 Experimental results

We implemented the both original private sorting protocol [2] and our variant using Andrew's implementation of various homomorphic encryption based protocols as a

Operation	Paillier	QR
Encryption	62	490
Decryption	530	512

Table 12: Runtime for Enc Dec operations in Paillier and QR schemes with Keysize 1024-bit, measured in millisecond, results for 100 iterations

Number of items	Foteini-Olga protocol	Modified protocol
10	8.5	2.2
20	32.4	8.3
40	143.5	34.3
60	321.8	74.8
80	544.3	133.1
100	850.3	208.6

Table 13: Run time (time in sec.)

codebase [23]. We use the implementations of DGK protocol, Paillier cryptosystem and other modules in our implementation. These protocols are implemented in Java, we used a windows-10 operating system with 16GB RAM and AMD Ryzen-7 2.3 GHz processor. For the QR and Paillier encryption protocols, the security parameter is taken as 1024 bits.

The aim of this work is to design an efficient private sorting scheme that can be used in encryption-based PPRS. Thus, our aim is not to evaluate the accuracy of any specific PPRS algorithm (content-based, collaborative or model-based protocols), so we have not used any well-known datasets for our experiments. We generated varying amount of random numbers from a range $[0, 10]$ that is very common in rating-based web applications and encrypted them. Then the encrypted vector is assumed to be the predicted rating vector and we run the protocol to find top-k values in the vector. We measured the time taken to perform private sorting, this does not include key generation or pre-computation time as they are one time computation. We performed the experiment at least five times for each vector size and noted the average time taken by the system to perform computations. The number of items we started with 10 and gradually increased to 100. Due to our system limitations we could not test for more items. We tested the protocol on a single machine so communication delays are not measured. It is worth noting that since we are not using DGK protocol, we only use Paillier cryptosystem. Table 12 shows that the Paillier cryptosystem is faster as compared to QR scheme and reduces the overhead on the servers. The comparison results between original using Foteini-Olga protocol and the modified scheme are shown in Table 13.

It is important to note that the recommendation service is not always online. For example, in YouTube the recommended video list is updated after every video is clicked. As another example, in news recommender systems, a user profile is created and news articles are pushed few times a day, or when an event occurs. If the

application is not very time sensitive, the server latency to generate the top-k most relevant items maybe tolerated.

6 Conclusion

In this work, we studied top-k item recommendation problem in privacy-preserving recommender systems and narrowed it down to efficient private homomorphic sorting scheme. Private sorting is a classical problem and the available solutions include homomorphic encryption based approaches as well. We studied a PHE-based private sorting scheme by Foteini and Olga, and modified it to enhance the performance. We propose to replace DGK private comparison protocol with a secure bit decomposition protocol, it reduces the communication and computation cost overheads. The communication cost after our modification is reduced by one-third. The computation cost at the keyserver is also reduced to almost half as no dynamic encryption are needed during the protocol execution now. The dataserver has to spend less on computing as the expensive operations like encryption is not needed in the modified scheme. These are replaced by comparatively cheaper operations like modular exponentiations. We show in our privacy analysis that the proposed modification does not compromise the privacy goals of a recommender system. In future we would like to observe the performance of our modification when the applications use multi-threading and to create more efficient codebase for the proposed protocol.

References

1. Badsha, S., Yi, X., Khalil, I., Bertino, E.: Privacy preserving user-based recommender system. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 1074–1083. IEEE (2017)
2. Baldimtsi, F., Ohrimenko, O.: Sorting and searching behind the curtain. In: International Conference on Financial Cryptography and Data Security. pp. 127–146. Springer (2015)
3. Batchier, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, spring joint computer conference. pp. 307–314 (1968)
4. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. *Cryptology ePrint Archive* (2014)
5. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Annual cryptology conference. pp. 505–524. Springer (2011)
6. Canny, J.: Collaborative filtering with privacy. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 45–57. IEEE (2002)
7. Cetin, G.S., Savas, E., Sunar, B.: Homomorphic sorting with better scalability. *IEEE Transactions on Parallel & Distributed Systems* **32**(04), 760–771 (2021)
8. Chatterjee, A., Kaushal, M., Sengupta, I.: Accelerating sorting of fully homomorphic encrypted data. In: International Conference on Cryptology in India. pp. 262–273. Springer (2013)
9. Cheng, K., Tong, W., Zheng, L., Fu, J., Mu, X., Shen, Y.: A secure and fair double auction framework for cloud virtual machines. *IEEE Access* **9**, 87982–87994 (2021)

10. Damgård, I., Geisler, M., Krøigaard, M.: Efficient and secure comparison for on-line auctions. In: Australasian conference on information security and privacy. pp. 416–430. Springer (2007)
11. 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)
12. Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction* **4**(2), 81–173 (2011)
13. Erkin, Z., Veugen, T., Toft, T., Lagendijk, R.L.: Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE transactions on information forensics and security* **7**(3), 1053–1066 (2012)
14. Ge, Z., Liu, X., Li, Q., Li, Y., Guo, D.: Privitem2vec: A privacy-preserving algorithm for top-n recommendation. *International Journal of Distributed Sensor Networks* **17**(12), 15501477211061250 (2021)
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning. pp. 201–210 (2016)
16. Jeckmans, A., Peter, A., Hartel, P.: Efficient privacy-enhanced familiarity-based recommender system. In: European Symposium on Research in Computer Security. pp. 400–417. Springer (2013)
17. Jumonji, S., Sakai, K., Sun, M.T., Ku, W.S.: Privacy-preserving collaborative filtering using fully homomorphic encryption. *IEEE Transactions on Knowledge and Data Engineering* pp. 1–1 (2021). <https://doi.org/10.1109/TKDE.2021.3115776>
18. Lai, S., Yuan, X., Sun, S.E., Liu, J.K., Liu, Y., Liu, D.: Graphse²: An encrypted graph database for privacy-preserving social search. p. 41–54. Asia CCS '19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3321705.3329803>
19. Lindell, Y.: Secure multiparty computation for privacy preserving data mining. In: *Encyclopedia of Data Warehousing and Mining*, pp. 1005–1009. IGI global (2005)
20. Narumanchi, H., Goyal, D., Emmadi, N., Gauravaram, P.: Performance analysis of sorting of the data: integer-wise comparison vs bit-wise comparison. In: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA). pp. 902–908. IEEE (2017)
21. Ogunseyi, T.B., Avoussoukpo, C.B., Jiang, Y.: Privacy-preserving matrix factorization for cross-domain recommendation. *IEEE Access* **9**, 91027–91037 (2021)
22. 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)
23. Quijano, A.: Homomorphic encryption (2020), https://github.com/AndrewQuijano/Homomorphic_Encryption/
24. Rosinosky, G., Da Silva, S., Ben Mokhtar, S., Négru, D., Réveillère, L., Rivière, E.: Pprox: efficient privacy for recommendation-as-a-service. In: Proceedings of the 22nd International Middleware Conference. pp. 14–26 (2021)
25. Samanthula, B.K., Chun, H., Jiang, W.: An efficient and probabilistic secure bit-decomposition. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. pp. 541–546 (2013)
26. Shmueli, E., Tassa, T.: Secure multi-party protocols for item-based collaborative filtering. In: Proceedings of the eleventh ACM conference on recommender systems. pp. 89–97 (2017)
27. Veugen, T.: Comparing encrypted data. Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and TNO Information and Communication Technology, Delft, The Netherlands, Tech. Rep (2011)

Input : Dataserver's input: $E(x), \langle E(x_0), E(x_1), \dots, E(x_{m-1}) \rangle$
Output: Dataserver receives: $\gamma = 0$ if the decomposition was correct or 1 otherwise

1. Dataserver:
 - $U = \sum_{i=0}^{m-1} E(x_i)2^i \pmod N$
 - $V = U + E(x)(N-1) \pmod N$
 - $r' \in_r \mathbb{Z}_N$ and $W = Vr' \pmod N$
 - Send W to keyserver
2. Keyserver:
 - If $D(W) = 0$: set $\gamma = 0$
 - Else: set $\gamma = 1$
 - Send γ to dataserver

Algorithm 3: Secure bit decomposition sub-protocol: SVR()

28. Wang, J., Chao, J., Tang, Q., Liu, Z., Khin, A.M.M.: Cryptorec: Novel collaborative filtering recommender made privacy-preserving easy. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2021). <https://doi.org/10.1109/TDSC.2021.3065752>

A SBD: Secure verification of result phase

The second half of the SBD protocol is to verify if the bit decomposition is correct or not from the step 5 to 8 in Algorithm 1. The sub-protocol: secure verification of result (SVR()) is used to perform this verification. Basically what the dataserver does here, it reconstructs the integer from the decomposed bits, masks it with some random noise, and send it to the keyserver for decryption. If the bit decomposition is correct, the keyserver will receive encryption of 0 otherwise some random encrypted number. The result is conveyed to the dataserver; if the decomposition is incorrect, the dataserver starts over from step 2 of Algorithm 1.