Privacy-Preserving Relevance Ranking Scheme and Its Application in Multi-keyword Searchable Encryption

Peisong Shen[1,2], Chi Chen[1,2(✉)], and Xiaojie Zhu[3]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, CAS, Beijing, China
{shenpeisong, chenchi}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences,
Beijing, China
[3] University of Oslo, Oslo, Norway
xiaojiez@ifi.uio.no

Abstract. Searchable Symmetric Encryption (SSE) which enables keyword searches on encrypted data, has drawn a lot of research attention in recent years. However, many SSE schemes do not support privacy-preserving relevance ranking which is a necessary feature for users to quickly locate the needed documents in a large number of retrieved documents. In this paper, we proposed two Privacy-Preserving Relevance Ranking (PPRR) schemes based on RSA encryption and ElGamal encryption. The proposed PPRR schemes preserve rank privacy and reduce storage cost at server side. Furthermore, we integrate PPRR with current multi-keyword SSE algorithm to achieve multi-keyword ranked search on encrypted data. Computation complexity, storage complexity and security of composite schemes are verified with an experiment on real-world dataset.

Keywords: Searchable symmetric encryption
Privacy-preserving relevance ranking · Cloud storage

1 Introduction

With wide deployment of cloud storage services, more and more users outsource their data to cloud server. However, a major concern of cloud storage service is privacy of personal data. On one hand, cloud storage service provider (CSSP) may be malicious and trade personal data for profit. On the other hand, CSSP may be compromised by attackers. Worse still, successive data breach events deepen users' concern about data privacy. To protect confidentiality of data, users usually encrypt their data before outsourcing them to cloud server. However, classical cryptographic algorithms disable information retrieval technique. For example, users cannot perform keyword search query on encrypted data to quickly retrieve the documents they want.

In recent years, many Searchable Symmetric Encryption (SSE) schemes [1–6] have been proposed to solve the problem of keyword search on encrypted data. These schemes used symmetric encryption primitives to protect the keywords and files.

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 128–146, 2018.
https://doi.org/10.1007/978-3-319-78813-5_7

Many SSE schemes support retrieving documents containing query keywords from the cloud server. However, returned documents of these schemes are not ranked by their relevance with search query, which poses a big challenge for users to find their documents from large set of search result. This problem motivates researcher's interest in designing SSE schemes supporting relevance ranking.

Wang et al. [7] proposed ranked search symmetric encryption (RSSE) based on Order-Preserving Encryption. However, due to the limitation of OPE, their scheme cannot be extended to multi-keyword search setting. In [14], A fully homomorphic encryption (FHE) method is used to achieve privacy-preserving relevance ranking at server side. However, their scheme was inefficient due to high computation complexity of FHE algorithm.

Cao et al. [8] proposed the first scheme supporting privacy-preserving Multi-Keyword Ranked Search on Encrypted data (MRSE). Based on the innovative work in [8], many practical schemes [9–12] have been proposed to solve the problems such as accuracy, index updates and search efficiency. These creative achievements promote the application of SSE schemes in real-word cloud storage service. However, the storage cost of encrypted indexes of MRSE-based schemes is proportional to the product of dictionary size and file collection size. The size of encrypted indexes will increase quickly with size of the dictionary and file collection.

Besides, many above-mentioned schemes supporting ranked search do not protect rank privacy, i.e. rank order of search result is disclosed to server. A malicious server can correlate same queries based on rank order of search results, then crack the query based on some background knowledge of dataset, such as statistical distribution of document frequency.

In this paper, we propose two Privacy-Preserving Relevance Ranking (PPRR) schemes which has lower storage overhead and protects the rank privacy of search results. The proposed PPRR schemes utilize Term Frequency(TF)-Inverse Document Frequency(IDF) method to capture the relevance between query and documents. The relevance scores are computed in an encrypted manner on the server side. Result ranking is done at client side to protect rank privacy. In order to keep the value of TF and IDF secret, RSA encryption and ElGamal encryption are used. Based on multiplicative homomorphism of both algorithms, relevance scores are computed securely and accurately at the server side. Randomness is introduced into PPRR-2 scheme to confuse distribution of TF and IDF values. Furthermore, we integrate PPRR schemes with current multi-keyword SSE scheme to support multi-keyword ranked search on encrypted data.

The main contribution of this paper is summarized as follows:

(1) Two privacy-preserving relevance ranking algorithms are proposed. Both of PPRR schemes can protect the rank privacy and resist statistical attack in a strong threat model.
(2) We integrate PPRR schemes with a state-of-art multi-keyword SSE scheme. The composite scheme has sublinear search efficiency, low storage overhead and supports dynamic index updates.

2 Related Work

2.1 Searchable Symmetric Encryption

Song et al. [1] proposed the problem of keyword search on encrypted data for the first time. They designed a SSE scheme based on string matching to solve the problem. However, their scheme needs a sequential scan of all encrypted data to find matched documents. Curtmola et al. [2] proposed a formal definition and security notion of searchable symmetric encryption. They also constructed two SSE schemes based on an inverted-index structure. In order to keep keyword privacy and document privacy, symmetric encryption is used to encrypt the indexes. Following their work, some SSE schemes [3, 4] have been proposed to handle index updates. Kamara et al. [3] used a XOR-based private key encryption to modify encrypted pointer when dealing with linked list node addition. File deletion is handled by using a deletion array which marks deleted files. Stefanov et al. [4] designed a dynamic searchable encryption scheme with a novel hierarchical index structure. Their scheme achieves logarithmic search efficiency. However, these SSE schemes mentioned above only support single-keyword search.

Cash et al. [5] proposed the first SSE schemes supporting multi-keyword search and sublinear search efficiency. The main idea of multi-keyword search of the proposed OXT protocol is that the server firstly retrieves documents containing one keyword in query and then decides whether the other keywords of query occurs in these documents or not. To protect data privacy, they devised an oblivious shared computation protocol between client and server based on blinded exponentiation. Furthermore, in [6] they proposed several efficient single-keyword SSE constructions which can be used as components in OXT protocol. Their scheme used a dictionary structure which supports dynamic updates. They also identified the locality issue of search performance of SSE schemes and gave their solutions to fix this issue.

2.2 Searchable Encryption with Relevance Ranking

Wang et al. [7] designed an order-preserving encryption method which ranks search results based on order-preserving-encrypted TF values in single-keyword ranked search setting. Cao et al. [8] first proposed a privacy-preserving Multi-Keyword Ranked Search on Encrypted data (MRSE). Their scheme is based on vector space model, and utilizes the “coordinate matching” to capture the relevance between documents and queries. Secure kNN algorithm is used to encrypt the indexes. However, their scheme needs to sequentially scan all the encrypted document vectors to find search results. Based on MRSE architecture, many enhanced schemes [9–12, 15] have been proposed in recent years. A multi-dimensional tree is used by Sun et al. [9] to improve the search efficiency. Chen et al. [15] designed a hierarchical cluster index to speed up searches on the cloud server. Xia et al. [10] construct a tree-based index structure and propose a “Greedy Depth-first Search” algorithm to provide efficient multi-keyword ranked search. Li et al. [11] proposed an enhanced MRSE scheme supporting logic search query, they also employed classified sub-dictionaries technique to enhance search efficiency.

In order to achieve accurate relevance evaluation at the server side, Shen et al. [14] used fully homomorphic encryption (FHE) to encrypt TF and IDF values of keywords. They also integrate their FHE scheme with OXT protocol to achieve the multi-keyword ranked search semantics. In 2017, Song et al. [12] proposed a privacy-preserved full-text retrieval algorithm over encrypted data. They used hierarchical bloom filters as their encrypted index and proposed the concept of membership entropies of index words to calculate relevance between query and documents on cloud server. Jiang et al. [13] modified Cash's OXT protocol to support top-k search. They precomputed the multiplication of TF and IDF values in index-building phase, then incorporated the result into the index to support relevance score computation on the server side. In order to protect privacy of TF*IDF values and rank order, they utilize the additive homomorphic property of paillier cryptosystem. However, their scheme doesn't support TF/IDF updates well.

3 Problem Specification and Prerequisite

3.1 Notations and Symbols

We list some notations which will be used in the following sections:

F – File collection
F_j – the j-th file in file collection or the file with an identifier j
$|F_j|$ – the number of unique keywords in F_j
w – keyword
$F(w)$ – identifiers of files containing keyword w
n – number of documents in file collection
D – dictionary composed of all keywords extracted from file collection
m – number of keywords in dictionary
Q – query
T_Q – the trapdoor of query Q
λ – security parameter
K – secret key
$TF_{w,j}$ – the term frequency of keyword w in j-th file
DF_w – the document frequency of keyword w
IDF_w – the inverse document frequency of keyword w
I – encrypted index
R_Q – the search result of query Q
N_Q – number of files in the search result of query Q
PRF – pseudo-random function
a|b – concatenation of string a and string

3.2 System Model and Searchable Encryption Definition

We design our SSE scheme in system model which is depicted in Fig. 1. Two entities are involved in this scenario: cloud server and data owner. Data owner generates index

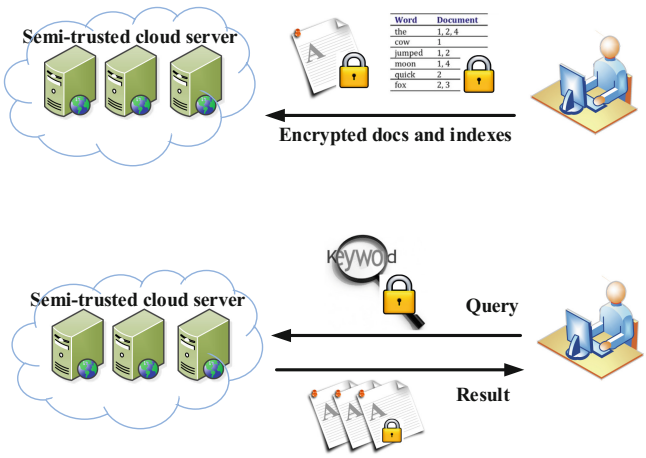

Fig. 1. System model

and encrypt files and index before outsourcing them to the cloud server. To perform a keyword search, the data owner generates the corresponding trapdoor and send it to the cloud server. Once receiving this trapdoor, the cloud server searches index for matched document and calculates relevance score of returning documents. At last, the sorted search results are returned to the data owner. We define searchable encryption as follows:

(1) **Keygen** is a key generation algorithm run by data owner. It takes a security parameter λ, and returns a secret key K.
(2) **Build_Index** is an algorithm run by data owner to generate the encrypted index. It takes a secret key K and file collections F, returns the encrypted index I.
(3) **Trapdoor** is run by data owner to generate a trapdoor for a given query. It takes a secret key K and a query Q, returns trapdoor T_Q.
(4) **Search** is a run by the cloud server in order to find documents containing query keywords. The documents are ranked by their relevance to the query keywords. It takes encrypted index I and trapdoor T_Q, returns the result set of documents.

3.3 Threat Model

In this paper, we suppose cloud server is "semi-trusted" which means the cloud server can dutifully execute the computation and storage operations in daily work. However, it's curious about the file content and index information, so it may try to deduce some information from the encrypted data. In this paper, we adopt the same threat model as [8]. In this model, the server not only knows the content of encrypted index and trapdoors, but also knows some background knowledge about the file collection, such as TF/DF statistical distribution.

3.4 Assessment Criteria

We will evaluate our scheme in three aspects: computation complexity, storage complexity and security.

Computation Complexity: practical multi-keyword SE scheme should achieve logarithmic (sublinear) search efficiency which is essential in real-world scenario.

Storage Complexity: As far as we know, SSE schemes in [5, 6] have the optimal storage overhead $O\left(\sum_{w \in D} DF_w\right)$ which is linear to total number of document-keyword pair.

Security: Security of SSE schemes mainly refers to index privacy and query privacy. Index privacy denotes the privacy of information such as keywords in the document, number of documents, document length and so on. Query privacy refers to privacy of keywords in the search query. If SSE scheme supports TF-IDF based relevance ranking, privacy of TF and IDF should be protected in the construction of SE scheme. Besides, rank privacy should also be considered in threat model.

3.5 TF-IDF Relevance Evaluation Method

In information retrieval community, Term Frequency-Inverse Document Frequency (TF-IDF) method is widely used to calculate the relevance score between a document F_j and a query Q:

$$Score\left(Q, F_j\right) = \sum\nolimits_{w \in Q} TF_{w,F_j} \times IDF_w \tag{1}$$

$$idf = \log \frac{n}{df_w} \tag{2}$$

In detail, TF is abbreviation of Term Frequency which is the number of occurrences of keyword w in document F_j. DF demotes Document Frequency which is the number of documents containing keyword w. IDF is the inverse value of DF.

3.6 Rivest-Shamir-Adlema (RSA) Encryption

RSA encryption is an asymmetric encryption algorithm proposed by Rivest, Shamir and Adlema in 1977. It's widely used in today's information systems and network infrastructure. It can be used for encryption, digital signature method, key distribution and so on. RSA algorithm works as follows:

$$Enc: c = m^{K_{pub}} (mod\, n), Dec: m = c^{K_{pri}} (mod\, n) \tag{3}$$

$$n = p * q,\ K_{pub} * K_{pri} = 1 (mod\, \varphi(n)) \tag{4}$$

In Eq. 4, p and q are big primes. $\varphi(n)$ denotes Euler function. RSA algorithm is homomorphic in multiplication:

$$c_1 * c_2 = m_1^{K_{pub}} * m_2^{K_{pub}} = (m_1 * m_2)^{K_{pub}} (mod\, n) \tag{5}$$

3.7 ElGamal Encryption

ElGamal algorithm is an asymmetric encryption. It works as follows:

$$KeyGen: h = g^d (mod\, p) \tag{6}$$

$$Enc: c_1 = g^r (mod\, p), c_2 = mh^r = (mod\, p) \tag{7}$$

$$Dec: m = c_2 \left(c_1^d\right)^{-1} (mod\, p) \tag{8}$$

In above equations, p is a big prime. g denotes a generator of a multiplicative group $Z_p^* = \{1, \ldots, p-1\}$. $0 < d < p-1$ is secret key, h is public key. m $\in Z_p$ is plaintext, (c_1, c_2) is ciphertext. The ElGamal algorithm is homomorphic in modular multiplication:

$$E(m_1) = (g^{r_1}, m_1 h^{r_1})(mod\, p),\ E(m_2) = (g^{r_2}, m_2 h^{r_2})(mod\, p) \tag{9}$$

$$E(m_1) * E(m_2) = (g^{r_1 + r_2}, m_1 m_2 h^{r_1 + r_2}) = (g^{r_3}, m_1 m_2 h^{r_3}) = E(m_1 * m_2)(mod\, p) \tag{10}$$

4 Privacy-Preserving Relevance Ranking Scheme

4.1 General Idea

In this section, we present the design rationale of privacy-preserving relevance ranking (PPRR) algorithm. Following previous SE schemes supporting relevance ranking, we adopt TF-IDF method to evaluate relevance between query and documents. Our design goal is to construct a privacy-preserving TF-IDF evaluation method in client-server model. As we know, fully homomorphic encryption meets this requirement. However, FHE algorithm is impractical in real-world scenario due to its high computation complexity. Inspired by the fact that a major part of computation complexity is caused by multiplication in TF-IDF algorithm, we calculate the multiplication of TF and IDF values in an encrypted manner at the server-side, leaving the decryption and addition of intermediate results at the client. We choose RSA encryption and ElGamal encryption because of their multiplicative homomorphism.

In our first scheme, TF values are encrypted by RSA while in our second scheme, it is encrypted by the ElGamal. In both schemes, the encrypted TF values are outsourced to cloud server along with the encrypted index. When user submits a search query, IDF values of query keywords are encrypted in the same way as TF values and inserted into

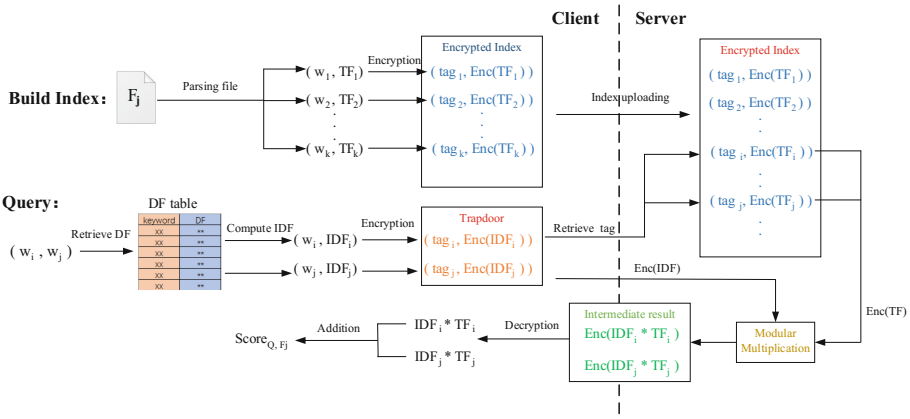

Fig. 2. Illustration of PPRR scheme.

trapdoor. After receiving the trapdoor, the cloud server multiplies encrypted TF with encrypted IDF to get encrypted relevance score for each query keyword. These intermediate results are returned to client. The client decrypts these intermediate scores and sum them up to get the final relevance score. By this way, the result ranking is done by client and rank privacy is protected. In Fig. 2, we demonstrate the architecture and working steps of PPRR scheme.

In PPRR schemes, encrypted TF and IDF are leaked to server. If these values are encrypted deterministically, the distribution of encrypted values remains same as plain values. As a result, a malicious server can deduce plain values based on data collection's statistical knowledge [7, 16]. In order to resist possible statistical attack, the encryption method needs to be a probabilistic one. Because ElGamal encryption is a probabilistic encryption method, PPRR-1 avoids this problem. However, RSA encryption is deterministic. In order to solve this problem, we introduce randomness into PPRR-2 algorithm to protect the privacy of TF/IDF values. In detail, $TF_{w,j}$ is multiplied by a random integer $R[j']$ before encrypted where $j' = j(mod\ T)$ and T is a modular parameter. Therefore, TF values of same keywords in different files are multiplied with different random numbers and TF distribution is confused. In consideration of large size of document collection, we introduce a modular parameter T to restrict the size of random array R. T is a trade-off between storage space and security. Similarly, IDF values of keyword w is multiplied by $R'[w]$ before encrypted where R' is an array which stores a random integer for each keyword in query. R' is reset for each new search request. As a result, query unlinkability is realized and IDF distribution of certain keyword is confused.

4.2 PPRR-1 Description

Detailed description of PPRR-1 scheme is shown in ALGORITHM 1. PPRR-1 uses ElGamal algorithm to encrypt TF values.

ALGORITHM 1. PPRR-1

Keygen:

generate (K_{pub}, K_{pri}) for ElGamal encryption.

generate keys K_T for PRF F_p (with range in Z_p^*)

Build_Index:

Inputs: file collection F, public key K_{pub}

Allocate an map I_{TF}

for each file F_j , **do**

Scan and tokenize F_j

for each keyword w in F_j, **do**

compute $TF_{w,j}$

set $y = ElGamal\_ENC(TF_{w,j})$

set $tag = F_P(K_T,\ w \mid j)$

store $\{tag, y\}$ pair in I_{TF}

Outputs: encrypted index I_{TF}

Trapdoor:

Inputs: file identifier j, query Q, DF table

Allocate an empty array T_Q

for each keyword w in query Q, **do**

get DF(w) from DF table and calculate IDF value IDF_w

set z = $ElGamal\_ENC(IDF_w)$

set $tag = F_P(K_T,\ w \mid j)$

add pair (tag, z) to the array T_Q

Outputs: trapdoor T_Q

Search:

Inputs: encrypted index I_{TF}, trapdoor T_Q

Allocate an empty array S_j

for each pair in T_Q, do

get value $y = ElGamal\_ENC(TF_{w,j})$ from I_{TF} using tag

compute $EScore_w = y * z$

put $EScore_w$ into the array S_j

Outputs: array S_j

Post-Processing:

Inputs: array S_j

set $Score_{Q,f} = \sum_w ElGamal\_Dec_{K_{pri}}(EScore_w)$

Outputs: $\boldsymbol{Score_{Q,f}}$

4.3 PPRR-2 Description

The detailed description of PPRR-2 scheme is shown in ALGORITHM 2 which is based on RSA encryption.

ALGORITHM 2. PPRR-2

Keygen:

generate modulus n and (K_{pub}, K_{pri}) for RSA encryption.
generate keys K_T for PRF F_p (with range in Z_p^*)
initialize an array R and a random integer generator: random()
for integer $i < T$, **do**
set $R[i] = \text{random}()$, $R^{-1}[w] = (R[w])^{-1}(mod\ n)$

Build_Index:

Inputs: file collection F, public key K_{pub}
Allocate an map I_{TF}
for each file F_j , **do**
Scan and tokenize F_j
for each keyword w in F_j, **do**
compute $TF_{w,j}$
set $j' = j\ (mod\ T)$
set y= $RSA\_ENC(TF_{w,j} * R[j'])$
set tag = $F_P(K_T,\ w \mid j)$
store $\{tag, y\}$ pair in I_{TF}
Outputs: encrypted index I_{TF}

Trapdoor:

Inputs: file identifier j, query Q, DF table
Allocate an empty array T_Q, R'
for each keyword w in query Q, **do**
set $R'[w] = \text{random}()$
get DF(w) from DF table and compute IDF_w
set $z = RSA\_ENC(IDF_w * R'[w])$
set $tag = F_P(K_T,\ w \mid j)$
add pair (tag, z) to the array T_Q
Outputs: trapdoor T_Q

Search:

Inputs: encrypted index I_{TF}, trapdoor T_Q
Allocate an empty array S_j
for each pair in T_Q, do
get value y = $RSA\_ENC(TF_{w,j} * R[j'])$ from I_{TF} using tag
calculate $EScore_w = y * z$
put $EScore_w$ into the array S_j
Outputs: array S_j

Post-Processing:

Inputs: array S_j, array R', array R
set $Score_{Q,f} = \sum_{w \in Q} RSA\_Dec_{K_{pri}}(EScore_w) * (R'[w])^{-1} * (R[w])^{-1}$
Outputs: $\boldsymbol{Score_{Q,f}}$

4.4 Index Updates

A practical PPRR scheme should be able to handle index updates in case of document updates. When the file content updates, TF and IDF values needs updates too. TF values are updated in an encrypted manner while IDF values are directly updated at client side. We explain how encrypted TF values are added, deleted or modified:

TF Addition: If a keyword w was added to the document for the first time, we need to generate a pair (tag, y) which stores encrypted TF values of keyword w. then the pair is outsourced to cloud server and inserted into index I_{TF}.

TF Deletion: If all of keyword w was deleted in the document. We need to calculate the tag and send it to cloud server. The server deletes the pair matching tag or uses deletion array I_{TF-D} to mark which TF values have been deleted.

TF Modification: TF modification can be viewed as a combination of TF node deletion and TF node addition.

Because DF values change frequently in case of file updates, we maintain an array which stores DF values for each keyword at the client side. By this way, the trapdoor is generated based on the latest DF values in case of frequent document updates.

4.5 Security Analysis

In PPRR scheme, the cloud server is assumed to be "honest but curious", which means it will execute protocol honestly and try to learn significant information without breaking the protocol. Note that in our security analysis when we say query we mean the encrypted IDF part. Similarly, the ciphertext means the encrypted TF part.

Lemma 1: In PPRR-2, if scalar factors are selected uniformly random for each search query, the query unlinkability is achieved.

Proof Sketch: In the trapdoor generation, each IDF is multiplied by a random number r which is uniformly distributed over range [0, n] where n is modulus of RSA algorithm. So same IDF is encrypted to same ciphertext with possibility of $\frac{1}{n}$ which is negligible. Therefore, lemma 1 is proved.

Lemma 2: If the Elgamal encryption is sematic secure, the adversary has negligible advantage in distinguishing any two ciphertext or queries.

Proof Sketch: Assume there is an adversary that is able to distinguish two ciphertext or queries. Based on the sematic security definition, we can know the encryption algorithm is not sematic secure. However, in our scheme, both TF and IDF are encrypted by Elgamal encryption which is known for semantic secure. Therefore, any two ciphertext or queries are undistinguishable, which is contradict with the assumption. Therefore, lemma 2 is correct.

5 Multi-keyword SSE Scheme Supporting Relevance Ranking

5.1 General Idea

In this section, in order to achieve a practical multi-keyword searchable encryption method supporting relevance ranking, we integrate PPRR algorithms with OXT protocol [5] which is an efficient multi-keyword SSE scheme. The integration is conducted in a keyword search-first, relevance ranking-second manner. A straightforward way of integration is executing OXT protocol firstly and the server returns file identifiers of documents which contains query keywords set. Then the client decrypts the intermediate result and uses it to generate trapdoor of PPRR protocol. The server executes the PPRR search and returns the set of encrypted scores. Finally, the client decrypts these scores and add them up to get the final scores. However, this method needs two round of communication between client and server.

In order to realize the multi-keyword ranked search in one communication round, we move the computation task of tftag from client to server. By this way, the server can retrieve encrypted TF values from I_{TF} with tftag which is computed by himself. In detail, PPRR scheme is changed as follows:

1. Build_Index phase, the client uses PRF function to encrypt the query keyword w. File identifier is encrypted by symmetric encryption in OXT protocol. Then the PRF is applied to both encrypted values to generate the search tag of encrypted TF.
2. In trapdoor phase, client computes PRF-encrypted keywords and put it into trapdoor.
3. In search phase, cloud server computes *tftag* based on PRF-encrypted keyword in trapdoor and Encrypted file identifier in execution result of OXT search.

5.2 OXT-PPRR Scheme Description

In this section, we demonstrate our OXT-PPRR scheme. Because the PPRR-1 and PPRR-2 are integrated in a similar way, we only demonstrate OXT-PPRR-1 scheme in Algorithm 3.

ALGORITHM 3. OXT-PPRR-1 in one communication round

Keygen:

generate λ-bit key K_S

generate keys K_X, K_T, K_I, K_Z for PRF F_p (with range in Z_p^*)

generate key (K_{pub}, K_{pri}) for ElGamal encryption

Build_Index:

Inputs: file collection F, keys $K_{pub}, K_S, K_X, K_I, K_Z, K_T$

Allocate an empty array *TF*, an empty array *T*, an empty array I_X

for each keyword *w* in *D,* **do**

set $K_e = F(K_S, w)$

set $E_w = F_p(K_T, w)$

initialize a counter $c = 0$

for each file F_j containing *w*, **do**

set $xid = F_p(K_I, j),\ z = F_p(K_Z, w|c), y = xid * z^{-1}$

set $e = Enc(K_e, j), stag = F(K_e, c)$

add tuple $\{stag, (e, y)\}$ to array *T*

set $xtag = g^{F_p(K_X, w)*xid}$, add *xtag* to I_X

compute $TF_{w,j}$ and set $etf = ElGamal\_ENC(TF_{w,j})$

set $tftag = F_p(E_w,\ e)$

store $\{tftag, etf\}$ pair in I_{TF}

c ++

Sort array *T* in lexical order, and generate: $I_S = T$

Outputs: outsource encrypted index $I = (I_S, I_X, I_{TF})$ to the cloud server.

Trapdoor:

Inputs: query Q, DF table, keys K_X, K_I, K_Z, K_S, K_T

Allocate an empty array T_Q, T'_Q

Allocate an empty two-dimensional array $xtoken$

Choose the S-term which has least DF value, suppose it is w_1

Set $K_e = F(K_s, w_1)$

for $i = 1,2, \ldots DF_{w_1}$, **do**

set $stag_i = F(K_e, i)$, add it to T_Q;

for *j=2,…,* $|Q|$, **do**

set $xtoken[i,j] = g^{F_p(K_Z, W_1|i)*F_p(K_X, W_j)}$

Set $xtoken[i] = \{xtoken[i,2], xtoken[i,3] \ldots xtoken[i,n]\}$

Merge $xtoken$ into T_Q.

for each keyword *w* in query Q, **do**

get DF(*w*) from DF table and compute IDF value IDF_w

set $eidf = ElGamal\_ENC(IDF_w)$

set $E_w = F_p(K_T, w)$

add pair $(E_w, eidf)$ to the array *A*

add array *A* to T_Q

Outputs: trapdoor T_Q

Search:

Inputs: encrypted index I, trapdoor T_Q

Allocate an empty array *S*

for $c = 1,2, \ldots DF_{W_1}$,**do**
Retrieve *c-th* tuple (e, y) from I_S: $(e, y) = Retrieve(I_S, stag_c)$
initialize counter = 0
for i=2,…,n, **do**
If $(xtoken[c, i])^y \in I_X$, *counter* ++
If $counter = n$
for each pair $(E_w, eidf)$ in A, **do**
set $tag = F_p(E_w, e)$
get value $etf = ElGamal\_ENC(TF_{w,j})$ from I_{TF} using $tftag$
set $EScore_w = etf * eidf$
put $EScore_w$ into S_j
add $tuple(e, S_j)$ *to* S
Outputs: array S
Post-Processing:
Inputs: array S, K_{pri}
Allocate an empty array R
for each $tuple(e, S_j)$ *in* S, **do**
set $j = Dec(K_e, e)$
decrypt values in S_j and add them up:
$Score_j = \sum_w ElGamal\_Dec_{K_{pri}}(EScore_w)$
set $\boldsymbol{R[j] = Score_j}$
rank the search result R based on scores
Outputs: search result R

5.3 Computation Complexity

In this section, we analyze the computation complexity of OXT-PPRR scheme. We take OXT-PPRR-1 as example.

Build_Index:
The complexity of Build_Index is $O\left(\sum_{w \in D} DF_w\right)$. For each keyword-file pair, four PRF encryption, two modular multiplication, one ElGamal encryption, one modular exponentiation and one symmetric encryption are needed to encrypt the file identifier and TF value.

Trapdoor:
The query complexity composes of trapdoor complexity, search complexity and post-processing complexity. Trapdoor complexity is $O(|Q| * DF_w) + O(|Q|)$ where $|Q|$ is number of keywords in query Q and DF_w is number of document containing keyword w. Major part of trapdoor complexity is caused by computing OXT trapdoor. It's proportional to DF_w. For each query keyword and file in $F(w)$, two PRF encryption and one modular exponentiation are needed.

Search:
Search phase composes of OXT search and PPRR relevance computation. Complexity of OXT search is $O(|Q| * DF_w)$. For each query keyword and file in $F(w)$, one modular exponentiation and one Bloom Filter retrieval are executed. Complexity of PPRR relevance computation is $O(|Q| * N_Q)$ where N_Q is number of files in the search result of query Q. For each query keyword w and file F_j in search result, one PRF encryption, one modular multiplication and one array retrieval are needed to compute the relevance between keyword w and file F_j.

Post-precessing:
Complexity of post-processing phase is $O(|Q| * N_Q)$. For each query keyword w and file F_j in search result, one symmetric decryption and one ElGamal decryption are needed to get final relevance scores. The time complexity of addition and scores ranking is negligible.

5.4 Storage Complexity

Encrypted index at server-side is comprised of three parts: $I = (I_S, I_X, I_{TF})$. I_S and I_S is inherited from OXT protocol while I_{TF} is generated by PPRR-1. Encrypted index I_{TF} contains encrypted $TF_{w,j}$ for each file F_j containing keyword w. So storage space of encrypted index is $O\left(\sum_{w \in D} DF_w\right)$. Because storage complexity of I_S and I_X is also $O\left(\sum_{w \in D} DF_w\right)$. Then the overall storage complexity of server in OXT-PPRR-1 scheme is $O\left(\sum_{w \in D} DF_w\right)$.

In OXT-PPRR-1, The client stores DF table. So the storage complexity of client is $O(m)$ where m is number of keywords in dictionary D. In OXT-PPRR-2, the client keeps DF table, an array storing T random factors and an array storing $|Q|$ random factors. So the storage cost of client is $O(m + T + |Q|)$.

5.5 Security Analysis

In the integrated scheme, OXT protocol and PPRR protocol are loose-coupled with each other. The majority of composite scheme remains the same as OXT protocol and PPRR protocol except for moving the calculation of search tag of encrypted TF values to the server side. In order to keep privacy of query keyword, we use PRF-encrypted keyword as trapdoor. The cloud server cannot deduce the keyword information from encrypted keyword. As a result, the security of our composite scheme is based on the security of OXT scheme and PPRR scheme.

6 Experiment

Our experiment is implemented in JAVA on a machine equipped with Intel I7-4790 CPU and 16G memory. We choose 5000 documents from 20 newsgroups [17] as our dataset. The dictionary contains 6000 keywords. We compared OXT-PPRR schemes to EDMRS scheme [10] which is a MRSE-based scheme supporting multi-keyword

ranked search. In OXT-PPRR-1, we implement ElGamal encryption using Elliptic Curve with 224 bit-length key. In OXT-PPRR-2, we use 1024 bit-length key for RSA.

Storage Space Comparison:
Because $DF_w < n$, $O\left(\sum_{w \in D} DF_w\right) < O(mn)$, the index of OXT-PPRR schemes consume less storage space than EDMRS does. This is verified by our experimental result shown in Fig. 3. We can see that our scheme needs less storage space than EDMRS when dictionary size is 1000 and 2000. Because the index size of EDMRS is proportional to dictionary size, we can infer that OXT-PPRR scheme achieves better storage efficiency than EDMRS when dictionary size is larger than 1000.

Trapdoor Comparison:
Time consumption of OXT-PPRR's trapdoor generation is mainly decided by DF_w, which varies based on query keywords and is uncertain. So we won't make comparison.

Index Build Time Comparison:
Time consumption of three schemes are compared in Fig. 4. Index build time of OXT-PPRR schemes is linear to the size of document collection. Due to high computation overhead of matrix multiplication, EDMRS scheme consumes more time than OXT-PPRR schemes.

Search Time Comparison:
From Table 1, we can see that computation complexity of OXT-PPRR search is linear to least document frequency of all query keywords. Computation complexity of OXT-PPRR post-processing is linear to size of search results. Figure 5 shows search time of three schemes with different collection size. It's worth noting that search time of OXT-PPRR scheme includes duration time of search phase and post-processing phase, while search time of EDMRS scheme only includes duration time of search phase.

Table 1. Comparison between OXT-PPRR and EDMRS. T_{EC_E} denotes encryption time of elliptic curve. T_{EC_D} denotes decryption time of elliptic curve.T_{RSA_E} is encryption time of RSA algorithm. T_{RSA_D} is decryption time of RSA algorithm.

Schemes	OXT-PPRR-1	OXT-PPRR-2	EDMRS[10]
Computation complexity of Build_Index	$O(2 * T_{EC\_E} * \sum_{w \in D} DF_w)$	$O((T_{EC_E} + T_{RSA\_E}) * \sum_{w \in D} DF_w)$	$O(m^2 * n)$
Computation complexity of Trapdoor	$O(T_{EC\_E} * \lvert Q \rvert * DF_w)$	$O(T_{EC\_E} * \lvert Q \rvert * DF_w)$	$O(m^2)$
Computation complexity of Search	$O(\lvert Q \rvert * DF_w)$	$O(\lvert Q \rvert * DF_w)$	$O(\theta * m * logn)$
Computation complexity of Post-Processing	$O\left(T_{EC\_D} * \lvert Q \rvert * N_Q\right)$	$O\left(T_{RSA\_D} * \lvert Q \rvert * N_Q\right)$	
Storage complexity (Server)	$O(\sum_{w \in D} DF_w)$	$O(\sum_{w \in D} DF_w)$	$O(mn)$
Storage complexity (Client)	$O(m)$	$O(m+T+\lvert Q \rvert)$	

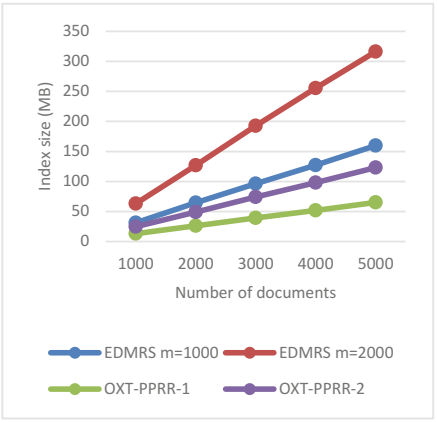

Fig. 3. Index size of three schemes.

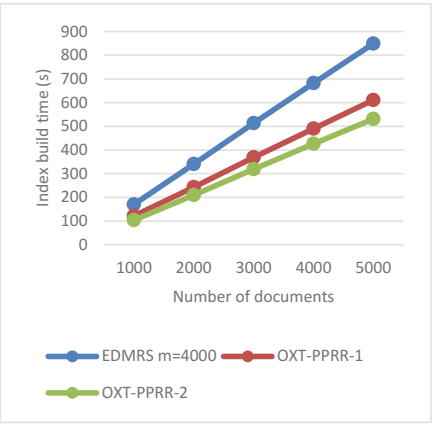

Fig. 4. Index build time

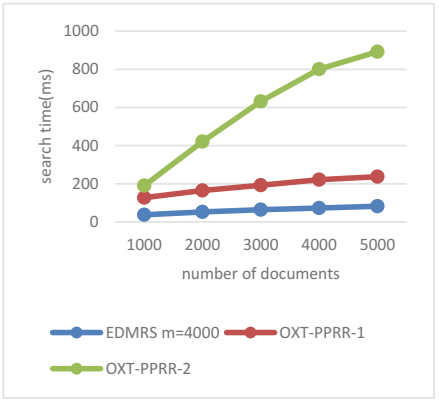

Fig. 5. Search time of three schemes with different size of document collection

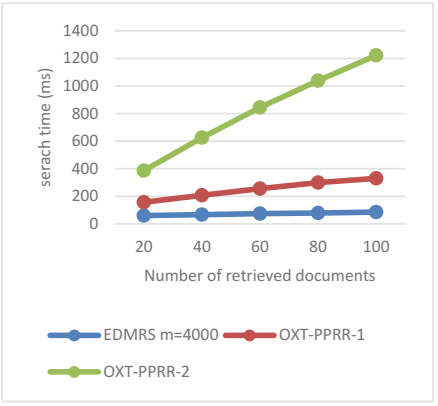

Fig. 6. Search time of three schemes with different size of retrieved documents

Because DF_w and N_Q increase much slower than collection size n, the search time of OXT-PPRR is sublinear to collection size. The tendency of curves in Fig. 5 confirms our judgement. However, due to the fact that RSA decryption and ElGamal decryption is a little time-consuming, OXT-PPRR schemes need more search time than EDMRS does. Besides, Fig. 6 verifies that search time of three schemes are all linear to the size of search results.

7 Conclusion

In this paper, we propose two privacy-preserving relevance ranking (PPRR) algorithms. Both schemes utilize multiplicative homomorphic encryption algorithms to protect TF/IDF and rank relevance scores at client-side in order to protect rank privacy. Besides, randomness is introduced into PPRR-2 to resist possible statistic attack in a strong threat model where attacker may be equipped with TF/IDF distribution knowledge. Furthermore, we incorporate PPRR schemes into Cash's OXT protocol to achieve practical multi-keyword ranked search on encrypted data. Finally, we analyze computation complexity, storage complexity and security of our scheme and experiment result confirms efficiency of our composite scheme.

Acknowledgments. This work was supported by National Science and Technology Major Project (No. 2016ZX05047003).

References

1. Song, D.X.D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of S&P, pp. 44–55, Berkeley, CA (2000)
2. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security (CCS), pp. 79–88, ACM (2006). http://dx.doi.org/10.1145/1180405.1180417
3. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM Conference on Computer and Communications Security (CCS), pp 965–976. ACM (2012). http://dx.doi.org/10.1145/2382196.2382298
4. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS Symposium (2014)
5. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for Boolean queries. In: Canetti, R., Garay, Juan A. (eds.) CRYPTO 2013 Part I. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_20
6. Cash, D., Jager, J., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: data structures and implementation. NDSS **14**, 23–26 (2014)
7. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: International Conference on Distributed Computing Systems (ICDCS), pp. 253–262 (2010). http://dx.doi.org/10.1109/ICDCS.2010.34
8. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: The 30th IEEE International Conference on Computer Communications, pp. 829–837. IEEE Press, New York (2011)
9. Sun, W., Wang, C., Cao, N., Li, M., Lou, W., Hou, Y.T., Li, H.: Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In: 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 79–88. ACM Press, New York (2013)

10. Xia, Z., Wang, X., Sun, X., Wang, Q.: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **27**, 340–352 (2016). https://doi.org/10.1109/TPDS.2015.2401003
11. Li, H., Yang, Y., Luan, T.H., Liang, X., Zhou, L., Shen, X.: Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. IEEE Trans. Dependable Secur. Comput. **13**, 312–325 (2015). https://doi.org/10.1109/TDSC.2015.2406704
12. Song, W., Wang, B., Wang, Q., Peng, Z., Lou, W., Cui, Y.: A privacy-preserved full-text retrieval algorithm over encrypted data for cloud storage applications. J. Parallel Distrib. Comput. **99**, 14–27 (2017)
13. Jiang, X., Yu, J., Yan, J., Hao, R.: Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. Inf. Sci. **403–404**, 23–41 (2017)
14. Shen, P., Chen, C., Tian, X., Tian, J.: A similarity evaluation algorithm and its application in multi-keyword search on encrypted cloud data. In: IEEE Military Communications Conference, pp. 1218–1223. IEEE Press, New York (2015). http://dx.doi.org/10.1109/MILCOM.2015.7357612
15. Chen, C., Zhu, X., Shen, P., Hu, J., Guo, S., Tari, Z., Zomaya, A.Y.: An Efficient privacy-preserving ranked keyword search method. IEEE Trans. Parallel Distrib. Syst. **27**, 951–963 (2016). https://doi.org/10.1109/tpds.2015.2425407
16. Zerr, S., Olmedilla, D., Nejdl, W., Siberski, W.: Zerber+R: top-k retrieval from a confidential index. In: International Conference on Extending Database Technology: Advances Database Technology, pp. 439–449 (2009)
17. NewsGroups dataset. http://qwone.com/~jason/20Newsgroups/