Inverted Index Based Multi-Keyword Public-key Searchable Encryption with Strong Privacy Guarantee

Bing Wang* Wei Song*† Wenjing Lou* Y. Thomas Hou*
*Virginia Polytechnic Institute and State University, Blacksburg, VA, USA
†Wuhan University, Wuhan, Hubei, China

***Abstract*—With the growing awareness of data privacy, more and more cloud users choose to encrypt their sensitive data before outsourcing them to the cloud. Search over encrypted data is therefore a critical function facilitating efficient cloud data access given the high data volume that each user has to handle nowadays. Inverted index is one of the most efficient searchable index structures and has been widely adopted in plaintext search. However, securing an inverted index and its associated search schemes is not a trivial task. A major challenge exposed from the existing efforts is the difficulty to protect user's query privacy. The challenge roots on two facts: 1) the existing solutions use a deterministic trapdoor generation function for queries; and 2) once a keyword is searched, the encrypted inverted list for this keyword is revealed to the cloud server. We denote this second property in the existing solutions as one-time-only search limitation. Additionally, conjunctive multi-keyword search, which is the most common form of query nowadays, is not supported in those works. In this paper, we propose a public-key searchable encryption scheme based on the inverted index. Our scheme preserves the high search efficiency inherited from the inverted index while lifting the one-time-only search limitation of the previous solutions. Our scheme features a probabilistic trapdoor generation algorithm and protects the search pattern. In addition, our scheme supports conjunctive multi-keyword search. Compared with the existing public key based schemes that heavily rely on expensive pairing operations, our scheme is more efficient by using only multiplications and exponentiations. To meet stronger security requirements, we strengthen our scheme with an efficient oblivious transfer protocol that hides the access pattern from the cloud. The simulation results demonstrate that our scheme is suitable for practical usage with moderate overhead.**

I. INTRODUCTION

With the growing popularity of cloud computing, people begin to embrace the benefits of cloud storage services by outsourcing their data into the cloud. To ensure the data privacy in the cloud, data must be encrypted before being outsourced into the cloud. However, data utilization, e.g. keyword search, arises as a challenging problem due to the data encryption. Downloading the entire encrypted data set first then searching over the decrypted data is obviously counterproductive and a waste of valuable computation and communication resources. Therefore, the search operation must be done at the cloud side and over the encrypted data.

Searchable encryption is an essential technique that allows search directly over encrypted data. Similar to search over plaintext documents, the common approach of searchable encryption schemes is to build a secure index for the entire document set. Then in the search stage, which is done on the cloud serve side, only the secure index is referenced. Secure index-based searchable encryption not only enhances the search scalability, but also allows the encryption of each document to be independent of the searchable encryption scheme used. In the searchable encryption literature, some of the works [1]–[13] design their own secure index structures while others [14]–[16] build upon the inverted index, which is one of the most popular index structures for plaintext search. Compared with building searchable encryption schemes using self-designed indexes, using the inverted index as the underlying index structure enjoys several benefits. First, search on the inverted index can be very efficient especially for a large dataset. Search is directly pointed to the related documents, i.e. the inverted lists that match the query keyword. Secondly, the inverted index has been widely adopted in large dataset search schemes. Therefore, secure inverted indexes can be applied incrementally over the already built inverted indexes.

The first effort of building a secure inverted index comes from Curtmola *et al.* [14], and followed by others [15], [16]. There are two major limitations of the existing inverted index based searchable encryption schemes.

- First of all, the keyword privacy is compromised once a keyword is searched. As a result, the index must be rebuilt for the keyword once it has been searched. Obviously, such a solution is counterproductive due to the high overhead suffered.
- Secondly, the existing inverted index based searchable schemes do not support conjunctive multi-keyword search, which is the most common form of queries nowadays.

In this paper, we explore the problem of building a searchable encryption scheme based on the inverted index to overcome the aforementioned limitations. We construct our scheme through a series of novel designs based on the private set intersection protocol in [17]. We achieve secure and private matching between the query trapdoor and the secure index. We design a novel trapdoor generation algorithm so that the query related inverted lists are combined together secretly without letting the cloud server know which inverted lists are retrieved. Our contributions are summarized as follows:

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

1) We propose a practical inverted index based public-key searchable encryption scheme. Our scheme overcomes the one-time-only search limitation in the existing schemes. Our scheme supports conjunctive multi-keyword search using only one trapdoor while the existing invert index based searchable encryption schemes only support single keyword search.
2) We design a probabilistic trapdoor generation algorithm to break the trapdoor linkability. Our scheme preserves the index and trapdoor privacy. To provide stronger security guarantee, we strengthen our scheme with an efficient oblivious transfer protocol to hide the access pattern.
3) Comparing with the existing public-key searchable encryption schemes which use expensive pairing operations, our scheme is more efficient because we only need multiplication and exponentiation.
4) Unlike most of the existing public-key searchable encryption schemes, we provide both a theoretical analysis and a simulation study using a real-world dataset to evaluate the performance. The simulation results show that our scheme is suitable for practical usage and the overhead is moderate.

We organize our paper as follows. In section II, we discuss the related works. Section III briefly introduces the preliminary concepts. We present our system model, threat model, and notations in section IV. Our solution is described in section V, followed by the security and the computation complexity analysis in section VI. We report the simulation results of our scheme in section VII. The conclusion of the paper is drawn in section VIII.

II. Related Works

Searchable encryption was first studied by Song et al. in [18]. The proposed scheme supports single keyword search without an index which means the server must scan the whole document to find the search result. Follow-ups on searchable encryption usually build a secure searchable index such that certain trapdoors generated via secret keys could match with the index to get the search result while the content of the index is hidden from the cloud. Some of the works [1]–[13] design their own index structures while others [14]–[16] build their secure searchable indexes upon the inverted index.

Among the self-designed secure indexes, Goh et al. [1] first proposed a Bloom filter based index which supports single keyword search. Chang et al. [3] used a vector index of which the length is the same as the cardinality of the dictionary for each document. Other works [5]–[13] focus on enrich the search functionality including result ranking, multi-keyword search and fuzzy search. In public key settings, Boneh et al. [2] proposed the first public key based searchable encryption scheme. Bellare et al. [4] introduced an as-strong-as-possible privacy definition for the public-key based searchable encryption and constructed a solution that satisfies their definition. One major disadvantage of using self-designed indexes is that their index structures are not compatible with each other. As a result, it is impossible to provide a service that includes all the useful functions. Additionally, for users who have already built their inverted indexes for plaintext search, they need to re-generate their encrypted searchable index which could be expensive if the data volume is huge.

Curtmola et al. [14] proposed the first inverted index based encrypted searchable index. The document list for each keyword is encrypted and obfuscated into an array. However, according to the design, the position and the content of the inverted list will be disclosed to the cloud server once the keyword is searched. As a result, one keyword can only be searched once before re-generating the index for the keyword. Based on [14], Kamara et al. [15] put forward the concept of dynamic searchable encryption and constructed an encrypted inverted index that supports dynamic operations such as document updates. Naveed et al. [16] designed a primitive named blind storage upon which they implemented the encrypted searchable index scheme of [14]. As these works are based on [14], they share the same limitation. Moreover, these schemes do not support conjunctive multi-keyword search which is the most common query type nowadays.

Our scheme falls into the public-key searchable encryption category which includes [2], [4], [7], [12], [19]–[21]. While those works all adopt a paring based crypto-system to construct the indexes, our scheme only requires multiplication and exponentiation, which are a magnitude less expensive in computation comparing to pairing operations. Additionally, most of these schemes did not provide an implementation evaluation of the algorithm performance under real-world application scenario (we only found [7], [12] performed the experimental studies). In this work, we not only provide a complexity analysis of our algorithm, but also perform a simulation study using a real-world dataset to evaluate the performance. The simulation results shows that our scheme is suitable for practical usage and is more efficient than existing public-key based solutions.

III. Preliminary

A. Inverted Index

Inverted index is one of the most popular data structures used in document retrieval systems [22]. An inverted index contains multiple inverted lists. One inverted list I_{ω_i} corresponds to one keyword ω_i which is contained in all the documents of I_{ω_i}. Additional information can be included in the inverted list such as the numerical statistics of the keyword (to support result ranking) and the positions of the keyword within a document (to support phrase search). The biggest advantage of using the inverted index comes from the search efficiency especially for large volume of documents. The search operation is performed on a much smaller document set which is consisted with the inverted lists that match with the query keywords.

B. Private Set Intersection

Private set intersection (PSI) is a cryptography primitive that allows two or more parties to calculate the intersection

of their datasets privately. The output of the PSI reveals no additional information other than the intersection itself. In [17], Freedman et al. proposed an efficient PSI protocol, denoted as the FNP protocol, based on the Paillier homomorphic cryptosystem [23]. A brief description of the Paillier algorithm is shown in Fig.1. The Paillier cryptosystem features 1) additive homomorphic, i.e. given the ciphertexts $E(a_1), E(a_2)$, the ciphertext of $a_1 + a_2$ can be calculated as $E(a_1 + a_2) = E(a_1)E(a_2)$, and 2) one-time multiplicative homomorphic, i.e. given the ciphertext $E(a_1)$, the ciphertext of $a_1 \times a_2$ is $E(a_1)^{a_2}$.

Key generation: $pk = (n, g)$, where
$n = pq, \gcd(pq, (p-1)(q-1)) = 1, g \in \mathbb{Z}^*_{n^2}$
$sk = (\lambda, \mu)$, where
$\lambda = \text{lcm}(p-1, q-1), \mu = \left(\frac{g^\lambda \mod n^2 - 1}{n}\right)^{-1} \mod n$
Encryption: To encrypt a message m to its ciphertext c
$c = g^m \cdot r^n \mod n^2, r \in \mathbb{Z}_n$
Decryption: $m = \frac{c^\lambda \mod n^2 - 1}{n} \cdot \mu \mod n$

Fig. 1. The scheme of the Paillier homomorphic cryptosystem

The FNP protocol works as follows. 1) Alice represents her set $\mathcal{A}$ as a polynomial $f(x) = \prod_{a_i \in \mathcal{A}}(x - a_i)$. Clearly, the set of the roots of $f(x) = 0$ is Alice's dataset. 2) Alice encrypts the coefficients of the polynomial using the Paillier homomorphic encryption and sends the encrypted polynomial $f'(x) = Enc(f(x))$ to Bob. 3) Bob calculate $\mathcal{R} : \{r_j = f'(b_j) +_h b_j\}$ with his data $b_j \in \mathcal{B}$, where $+_h$ is the Paillier homomorphic addition. Then Bob sends $\mathcal{R}$ back to Alice. 4) Alice decrypts $\mathcal{R}$ as $\mathcal{R}'$, and the intersection $\mathcal{A} \cap \mathcal{B}$ is the intersection $\mathcal{A} \cap \mathcal{R}'$.

C. Blind Storage

Blind storage proposed in [16] is an efficient oblivious transfer protocol implementation under the *honest-but-curious* adversary model. An overview of the process of blind storage is shown in Fig.2. We refer the readers to [16] for more details of blind storage.

The security property of the blind storage comes from the fact that the storing locations of the documents are independent of each other from the server's point of view. A blind storage only uses block ciphers and collision resistant hash functions so that it is computationally efficient. The communication and storage overhead to ensure a negligible probability of information leakage is moderate (by a factor of 4).

IV. System Model

The system model we consider in this work is shown in Fig.3. There are three entities in the system, a cloud server, a data owner and multiple users. The data owner generates the encrypted index and outsources it along with the encrypted data into the cloud. An authorized user submits a query request to the server in the form of a trapdoor which he gets from the data owner through a secure channel. After receiving the trapdoor, the cloud server matches the encrypted index with

Setup: a pseudorandom function Φ, a pseudorandom generator Γ, and F is a collection of $f = (ID_f, data_f)$
BuildStorage:
- D is the storage which is arranged as an array of n_D blocks. All of the blocks are marked as free at the beginning.
- For each f, the data owner generates a seed $s = \Phi(ID_f)$ and a pseudorandom sequence $\Lambda = \Gamma(s)$.
- The server stores $data_f$ along with its ID_f into the first $|data_f|$ free blocks in the sequence Λ.

Access:
- Given the ID_f, the user generates the seed $s = \Phi(ID_f)$ and the sequence $\Lambda = \Gamma(s)$ and retrieves a certain number of the blocks from D.
- After decryption, the user finds the target file blocks by the ID information.

Fig. 2. The scheme of the blind storage

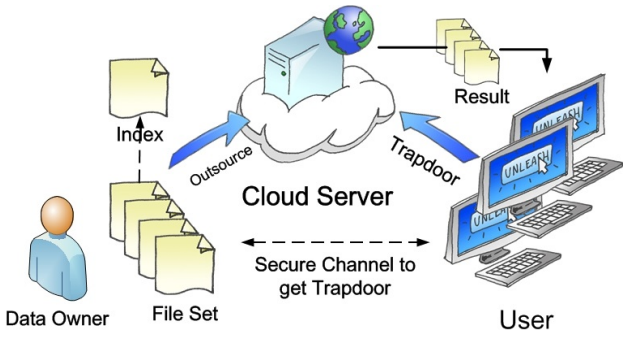


Fig. 3. System architecture of search over encrypted data in cloud computing

the trapdoor. Finally, the cloud server returns the matching documents as the search result.

We model the cloud server as a *honest-but-curious* adversary. We assume that the authorized users are fully trusted by the data owner and they are authorized to access all the documents through search. The access control between data owner and the users can be achieved using existing protocols such as [12], [24], and therefore, we will not discuss it in this paper.

A. Threat Model

A searchable encryption scheme should protect data owner's *data privacy*. The data privacy violation could come from three aspects. The first one is the confidentiality of the document set. In most scenarios, the data owner will encrypt his document contents using a block cipher such as AES. Therefore, it is safe to claim that the privacy of the document set itself is well protected. So we focus on the other two aspects, which are the *encrypted index privacy* and the *trapdoor privacy*.

Index privacy: The index privacy is twofold. First, the cloud server should not learn the content of the index since the content of the index directly reflects the content of the documents. Second, the cloud server should deduce no information about the document through analyzing the encrypted

index. Such information includes 1) whether a document contains certain keyword(s), and 2) whether different documents contain a common keyword.

Trapdoor privacy: A trapdoor is generated for each query request to allow the cloud server to search over the encrypted index. Intuitively, the trapdoor contains the query information but in an encrypted form. Given a trapdoor, the cloud server should learn nothing about the user's query from it. We consider the protection of the following information for trapdoor privacy: the content of the query, the number of the keywords in the query, and the fact that whether the same query has been searched before.

Access pattern refers to the accessed documents, i.e. the search results. As pointed out by [25], the adversary could further deduce the private information of the index and the trapdoor from the access pattern. To avoid such leakage, the search results of queries must be indistinguishable from each other.

B. Definition and Notation

We will use the following notations through the rest of the paper.

- $\Sigma = (\sigma_1, \sigma_2, \cdots, \sigma_n)$ is a finite set of document collection, where σ_i is the ID of the ith document.
- $\Omega = (\omega_1, \omega_2, \cdots, \omega_m)$ is a finite set of keyword collection from Σ, which we denote as dictionary.
- $\mathcal{I} = (I_{\omega_1}, I_{\omega_2}, \cdots, I_{\omega_m})$, an inverted index for the document set Σ. Each I_{ω_i} is a list which contains $\Sigma_i = (\sigma_{i1}, \sigma_{i2}, \cdots, \sigma_{ip}) \subset \Sigma$ where $\omega_i \in \sigma_{ij}, 1 \leq j \leq p$.
- $\tilde{\mathcal{I}}$ is the encrypted searchable index based on $\mathcal{I}$.
- $\mathcal{Q} \subset \Omega$ is a query request which is a subset of the dictionary.
- $\mathcal{T}_{\mathcal{Q}}$ is the trapdoor for the query $\mathcal{Q}$.
- $[n]$ means an integer set from 1 to n.
- $|\mathcal{S}|$ refers to the cardinality of $\mathcal{S}$ which can be a set, a list or a vector.

Before delve into the detail of our scheme, we first present the definitions of searchable encryption scheme given in [2], [14].

***Definition** 1 (Searchable Encryption Scheme):* A searchable encryption scheme consists of the following probabilistic polynomial time algorithms.

- **Setup**(k) takes as input a security parameter k. It outputs a master key MK.
- **IndexGen**($MK, \mathcal{I}$) takes as input the master key MK and an index $\mathcal{I}$ for the document set Σ. It outputs the encrypted searchable index $\tilde{\mathcal{I}}$.
- **TrapdoorGen**($MK, \mathcal{Q}$) takes as input the master key MK and a query $\mathcal{Q}$. It outputs the trapdoor $\mathcal{T}_{\mathcal{Q}}$ for the query.
- **Query**($\tilde{\mathcal{I}}, \mathcal{T}_{\mathcal{Q}}$) takes as input the encrypted index $\tilde{\mathcal{I}}$ and the trapdoor $\mathcal{T}_{\mathcal{Q}}$. It outputs $\mathcal{R} \subset \Sigma$ as the search result.

V. Our Inverted Index based Public-key Searchable Encryption Scheme

We present our inverted index based public-key searchable encryption scheme in this section. We assume the data owner already has an inverted index available for his dataset. Thus, we skip the process of building an inverted index from a document set.

A. Overview

As discussed in section III, an inverted index consists of two parts: 1) a keyword dictionary and 2) a document list for each keyword. When performing a search over plaintext, the server matches the query keyword(s) to the dictionary to locate the target document list(s) first. Then the server gets the document candidates by integrating the document list(s) together. Finally, the document candidates are returned to the user as the search result.

When searching over encrypted inverted indexes, we have to solve three challenging tasks. We first need a privacy preserving method to determine the match between the query keyword(s) and the dictionary. Then we need to select the related inverted lists without letting the cloud server know which ones are retrieved. Finally, the access pattern must be concealed. We address these challenges through a series of novel designs based on the FNP PSI protocol. To make the presentation clear, we use a simple example in Fig.4 to illustrate our scheme. We now present the details of our scheme as follows.

B. Scheme Details

Besides the four essential algorithms in Definition 1, our scheme features an extra algorithm to hide the access pattern from the cloud server. We store the documents in a blind storage in the cloud. Correspondingly, we retrieve the documents from the cloud storage following the access process of blind storage. We give a brief description of blind storage in Fig.2. More detail of blind storage is discussed in [16].

- **Setup**(k): The data owner first chooses two k-bit prime numbers p, q such that $\gcd(pq, (p-1)(q-1)) = 1$. Then the data owner follows the key generation process in Fig.1 to generate the key pair for the Paillier algorithm, i.e. $pk = (n, g), sk = (\lambda, \mu)$. The data owner keeps sk, a pseudorandom permutation (PRP) f and an invertible matrix M as the master key MK. The degree of M is determined by the size of the dictionary m. At last, the data owner publishes the public key to the cloud server.
- **IndexGen**($MK, \mathcal{I}$): Given an inverted index, the data owner takes the following steps to convert it to the encrypted version.
  1) For each keyword $\omega_i \in \Omega$ and its corresponding inverted list I_{ω_i}, the data owner performs the following steps.
     a) The data owner generates a tag $t_{\omega_i} = f(\omega_i)$. Similarly, the data owner generates a tag for each document $\sigma_i \in \Sigma$, where $t_{\sigma_i} = f(\sigma_i)$.

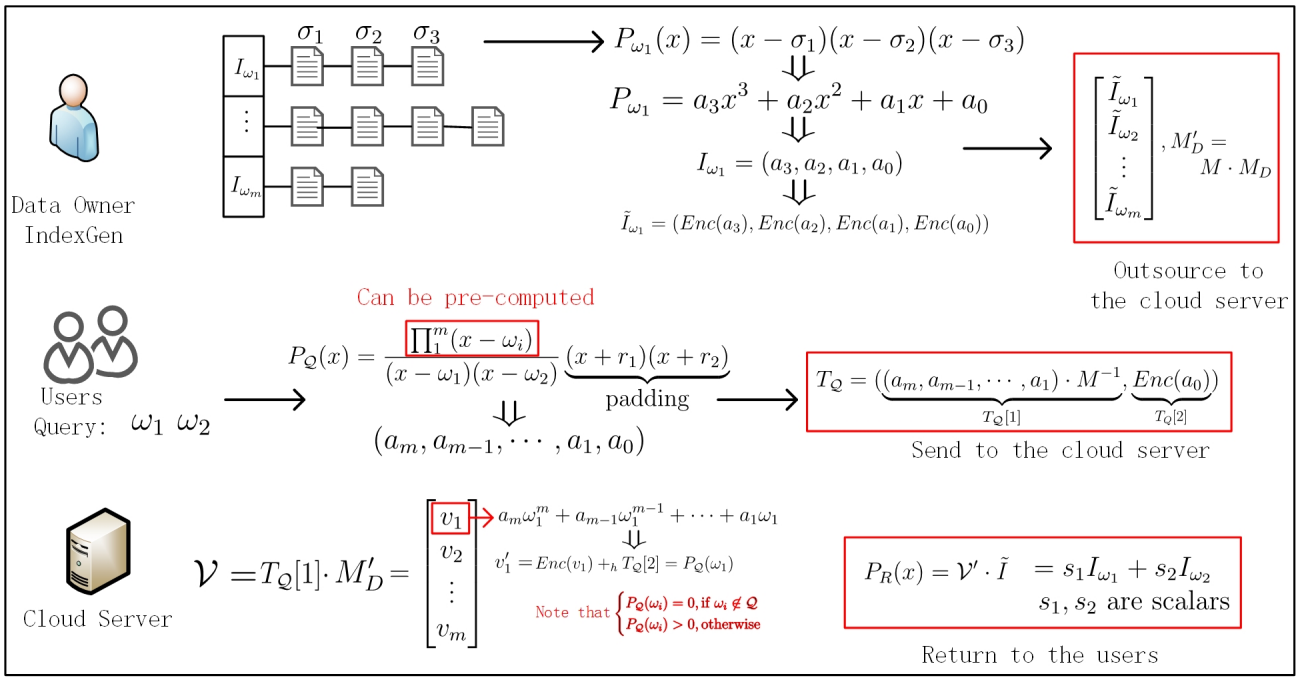


Fig. 4. In this figure, we show the whole process using a simple example. 1) The data owner transforms each inverted list to a polynomial. The polynomial is represented using its coefficients. To secure the index, the data owner encrypts the coefficients of each polynomial. Note that padding is not shown in the figure for simplicity. The data owner generates a dictionary matrix M_D following the algorithm. Note that here we use ω_i directly instead of the tag for simplicity. 2) When generating a trapdoor for a query, the data owner first calculates $P_\Omega/P_\mathcal{Q}$, then adding randomness to hide the number of the keywords in the query. The trapdoor is a 2-tuple. 3) When the cloud server receives the trapdoor, he calculates $\mathcal{V}'$ as shown in the figure. Each v_i' in $\mathcal{V}'$ is the ciphertext of $P_R(\omega_i)$. Because $P_R(\omega_i) = 0$ if $\omega_i \notin \mathcal{Q}$, the result polynomial $P_R(x)$ only contains the inverted lists corresponding to the query keywords.

We denote the set of the keyword tags and the document tags as $f(\Omega), f(\Sigma)$ respectively.

b) Let L be the maximum length of all the inverted list. The data owner generates a set of random numbers $\mathcal{R}_i = \{r_j\}$ for I_{ω_i}, where $r_j \in \mathbb{Z}_n^*, r_j \notin f(\Omega), |\mathcal{R}_i| = L - |I_{\omega_i}|$. Then, the data owner pads $\mathcal{R}_i$ to the inverted list to hide the length.

c) After getting all the polynomials of the inverted lists, the data owner generates a polynomial $P_{\omega_i}(x)$ for I_{ω_i} as

$$P_{\omega_i}(x) = \prod_{\sigma_j \in I_{\omega_i}} (x - t_{\sigma_j}) \prod_{r_j \in \mathcal{R}_i} (x - r_j).$$

2) The data owner calculates a polynomial vector as follows,

$$\mathcal{I} = (P_{\omega_1}, P_{\omega_2}, \cdots, P_{\omega_m})^T$$

3) The data owner encrypts the coefficients of each polynomial P_{ω_i} using the public key (n, g) of the Paillier encryption algorithm, and sets the encrypted index as $\tilde{\mathcal{I}} = Enc_{(n,g)}(\mathcal{I})$.

4) The data owner constructs a dictionary matrix M_D as

$$M_D = \begin{pmatrix} t_{\omega_1}^m & t_{\omega_2}^m & \cdots & t_{\omega_m}^m \\ t_{\omega_1}^{m-1} & t_{\omega_2}^{m-1} & \cdots & t_{\omega_m}^{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_{\omega_1} & t_{\omega_2} & \cdots & t_{\omega_m} \end{pmatrix}$$

Then he encrypts it with the matrix M as $M_D' = M \cdot M_D$.

5) Finally, the data owner outsources the matrix M_D' and the index polynomial vector $\tilde{\mathcal{I}}$ to the cloud.

- **TrapdoorGen**$(MK, \mathcal{Q})$: Before generating the trapdoors, the data owner first generates a polynomial for the entire dictionary P_D as

$$P_D(x) = \prod_{\omega_i \in \Omega} (x - t_{\omega_i})$$

Note that this is a one-time cost.
When the data owner receives a query request $\mathcal{Q}$ from a user, he constructs a polynomial $P_\mathcal{Q}(x)$ where $P_\mathcal{Q}(x) = P_D / \prod_{\omega_i \in \mathcal{Q}}(x - t_{\omega_i})$. Then, the data owner generates $P_\mathcal{Q}'(x)$ by padding random terms to $P_\mathcal{Q}(x)$.

$$P_\mathcal{Q}'(x) = P_\mathcal{Q}(x) \prod_{q+1}^{m} (x - r_j), q = |\mathcal{Q}|, r_j \notin f(\Omega)$$

We represent $P_\mathcal{Q}'(x)$ using its coefficients (a_m, a_{m-1}, a_1, a_0). Finally, the data owner returns

the trapdoor which is a 2-tuple to the user as

$$\mathcal{T}_{\mathcal{Q}} = \{(a_m, a_{m-1}, \cdots, a_1) * M^{-1}, Enc_{(n,g)}(a_0)\}$$

- **Query**$(\tilde{\mathcal{I}}, \mathcal{T}_{\mathcal{Q}})$: After receiving the trapdoor, the cloud server first calculates

$$V = T_{\mathcal{Q}}[1] * M'_D = (v_1, v_2, \cdots, v_m).$$

For each $v_i, i \in [m]$, the cloud server calculates $v'_i = Enc_{(n,g)}(v_i) +_h T_{\mathcal{Q}}[2]$ where $+_h$ is homomorphic addition of the Paillier encryption algorithm. Then all the values are organized as a vector:

$$\mathcal{V}' = (v'_1, v'_2, \cdots, v'_m).$$

After that, the cloud server calculates

$$P_R(x) = \mathcal{V}' \cdot \tilde{\mathcal{I}}^T \tag{1}$$

and returns $P_R(x)$ back to the user.

- **OT**(P_R): After receiving P_R, the user decrypts the polynomial with the assistance of the data owner. Then the user factors the polynomial to find the roots of $P_R(x) = 0$. Note that these roots are the tags of the document candidates. Finally, the user launches the blind storage fetch protocol to get the encrypted documents back from the cloud server.

VI. Security and Performance Analysis

In this section, we first prove the correctness and the security of our scheme. Then we analyze the complexity of our scheme.

A. Security Analysis

***Theorem** 1 (Completeness):* Our scheme returns all the documents that contain the query keyword(s).

Proof: We represent the query trapdoor as a polynomial $P'_{\mathcal{Q}}(x)$ so that the roots of $P'_{\mathcal{Q}}(x) = 0$ are the tags of the keywords except those in the query, i.e. $\{x | x = t_{\omega_i}, \omega_i \notin \mathcal{Q}\}$.

Because $v'_i = v_i +_h Enc_{(n,g)}(a_0)$ is the ciphertext of $P'_{\mathcal{Q}}(t_{\omega_i}), i \in [m]$, and $P'_{\mathcal{Q}}(\omega_i) = 0$ for those keywords $\omega_i \notin \mathcal{Q}$.

Then, the result polynomial P_R which is

$$P_R(x) = \mathcal{V}' \cdot \tilde{\mathcal{I}}^T = \sum_{\omega_j \in \mathcal{Q}} v'_j P_{\omega_j} \tag{2}$$

consists of the query-related inverted lists only. Note that the scalar, i.e. v_j, won't affect the root of the polynomials. The roots of equation 2 actually is $\cap_{\omega_j \in \mathcal{Q}} \Sigma_{\omega_j}$, which is the set of the documents that contain all the query keywords. Therefore, the user will fetch the correct documents from the cloud. ■

Before we prove the security of our basic scheme, we first review the definition of *semantic security*.

***Definition** 2 (Semantic security):* A cryptosystem is *semantically secure* if given the ciphertext of a message Msg, any probabilistic polynomial-time algorithm (PPTA) cannot deduce any partial information about Msg computationally with a high non-negligibly probability.

***Definition** 3 (Semantic security of searchable encryption):* A searchable encryption scheme is semantic secure if the encrypted indexes and the trapdoors are indistinguishable under Chosen Plaintext Attack (IND-CPA).

The underlying security significance of Definition 3 is that given the encrypted index and the trapdoors any probabilistic polynomial-time adversary (PPTA) cannot determine any partial information of the documents. As proved in [23], the Paillier homomorphic algorithm, which is based on the quadratic residuosity problem, is a semantically secure cryptosystem. Now we are ready to prove the security of our scheme.

***Theorem** 2 (Security):* Our inverted index based public-key searchable encryption scheme is semantically secure.

Proof: *(sketch)* Suppose a PPTA $\mathcal{A}$ chooses two document set Σ_0, Σ_1 that $|\Sigma_0| = |\Sigma_1|$ and $|\Omega_{\Sigma_0}| = |\Omega_{\Sigma_1}|$. $\mathcal{A}$ submits the datasets to the data owner. The data owner randomly choose $b = \{0, 1\}$ and generates the encrypted inverted index for Σ_b. After that, we allow $\mathcal{A}$ to access the trapdoor generation algorithm and the query algorithm. But we don't allow $\mathcal{A}$ to know the search result. This simulates the view of the cloud server in our scheme. If $\mathcal{A}$ cannot guess b correctly with a probability higher than $1/2$, then our scheme is semantically secure.

- **Security of the tags:** We use the PRF to produce the tags in our scheme. Because $|\Sigma_0| = |\Sigma_1|$ and $|\Omega_{\Sigma_0}| = |\Omega_{\Sigma_1}|$, $\mathcal{A}$ clearly cannot gain advantage from the tags. In practice, we can use a symmetric key encryption scheme like AES as the PRF.
- **Security of the index:** The index is represented as polynomials in our scheme, and the polynomials are encrypted using Paillier homomorphic encryption. Because the Paillier cryptosystem is semantically secure, the cloud server learns nothing from the encrypted polynomials. Furthermore, the cloud server cannot learn useful information from the length of the inverted list because of the padding.
- **Linkability of the trapdoors:** In $T_{\mathcal{Q}}[1]$, random terms are introduced to hide the length of the query. Because of the random terms, the coefficients of $P'_{\mathcal{Q}}$ will be different each time. In $T_{\mathcal{Q}}[2]$, a_0 is encrypted using the Paillier encryption. Recall with the Paillier encryption algorithm, a random number is used each time when performing the encryption, which means a plaintext will be encrypted differently with the same key. As a result, a plaintext query will be transfered to different trapdoors every time due to the introduced randomness.
- **Access pattern:** The matching result is a polynomial of which the coefficients are encrypted using the Paillier encryption. Therefore, the matching result is semantically secure as well. Furthermore, our blind storage based oblivious transfer process is secure under the ideal/real paradigm as shown in [16]. As a result, $\mathcal{A}$ cannot gain advantage to guess b correctly by observing the matching result. ■

B. Computation Complexity Analysis

In the *Setup* phrase, the data owner needs to generate the key pair for the Paillier algorithm. It takes two exponentiations

plus the computation of finding the prime numbers.

In the *IndexGen* process, there are m polynomials need to be encrypted. Each of them has degree of L. Since the polynomial is represented using its coefficient, we need $m \times L$ encryption operations in total. The data owner also needs m^3 multiplications to generate the M'_D. It is worth mentioning the matrix multiplication can be optimized using existing methods such as Strassen's algorithm [26].

In the *TrapdoorGen* process, the data owner needs to perform: 1) a polynomial division which can be efficiently done through the synthetic division method; 2) one exponentiation to encrypt a_0; and 3) m^2 multiplications to generate $T_{\mathcal{Q}}[1]$.

In the *Query* process, the cloud server needs to perform m^2 multiplications to calculate $\mathcal{V}$, m exponentiations to encrypt $\mathcal{V}$, and m multiplications to do the homomorphic summation.

Comparison with the existing public key based schemes: We compare the computational load of our scheme with the existing public-key based schemes [7], [12], [19]–[21] in Table I. Let E denote an exponentiation operation, M denote a multiplication and e denote a pairing operation and P denote a map-to-point hash function which hashes any input to the bilinear paring group G_1 (such operations are not efficient).

In most cases, the number of the documents are greater than the number of the keywords, i.e. $n \gg m$. Pairing operations are far more expensive than multiplication and exponentiation. The comparison table shows that our scheme is lighter-wright than the existing solutions. It is worth mentioning that although our scheme requires more computation when generating a trapdoor than other schemes, we support multi-keyword conjunctive search while none of them has this functionality.

C. One round communication vs. two round communication

Our scheme requires two rounds of communication between the user and the cloud server to complete a search request. The main reason is to protect the access pattern since a significant amount of information can be learned from it [25]. It is worth mentioning that our scheme can be modified to take only one round communication by letting the cloud server evaluate the result polynomial P_R using the document tags. However, the modified scheme will leak the access pattern to the cloud server. Based on the different application scenarios, one may choose the one-round scheme without the access pattern protection, depending on the security and privacy requirements demanded by the application.

VII. Simulation

In this section, we evaluate the performance of our proposed scheme. A series of real-world dataset based simulations are carried out to evaluate the computation cost of our scheme. Most of the public-key based searchable encryption schemes did not report a simulation study on algorithm efficiency. We only found Li et al. [7] reported a simulation studies on their proposed scheme so we will compare our results with theirs.

A. Environment Setting

We use Python to implement the prototype of our scheme on a Windows 8.1 PC with Intel Core i3 3.3 GHz and 4 Gigabyte memory. We use a simple pure Python implementation of the Paillier homomorphic cryptosystem[1].

Dataset: we use part of the Enron dataset [2] as our dataset. The Enron email data contains a large number of documents. The statistic of the dataset is shown in Table II (the keyword set excludes the stop words and has been processed using Porter's stemming algorithm).

B. Data owner computation

Before outsourcing the data into the cloud, the data owner needs to initialize the parameters of the system and generates the encrypted inverted index. We assume the data owner has built the inverted index already. In our simulation, we use Lucene [27] to generate the inverted index for our dataset.

1) System Setup: During the system initialization, the data owner first generates the key pair for the Paillier homomorphic system. This process includes finding two big prime numbers p, q and computing the public key and the private key. In our experiment, the key generation process takes 0.40 second for 512-bit security and 3.03 second for 1024-bit security. Although 768-bit RSA module has been broken [28], in our experiments, we report the computation costs for both 512-bit and 1024-bit keys to demonstrate the trade-off between the security and the efficiency.

2) Inverted Index Encryption: Although index generation process is the most computationally expensive step, it is only a one-time process. There are two steps of converting an inverted index to its encrypted version. The first step is to calculate the corresponding polynomial for each keyword list. The second step is to encrypt the polynomials with the master key which includes a matrix multiplication and a Paillier encryption. The computation cost will increase as the dictionary size grows large. It is worth to mention that when using 512-bit keys for the Paillier crypto-system, the matrix multiplication costs much more than the Paillier encryption while the encryption consumes more computation when we use 1024-bit keys. The computation of the matrix multiplication can be further reduced by using multiple smaller size matrices instead of a large size matrix. The computation cost of the index generation respect to different key size and matrix size is shown in Fig.5.

3) Trapdoor Generation: We show the computation cost for a trapdoor with different dictionary size in Fig.6. The most computationally expensive step is the matrix multiplication. Because only one encryption is involved, using different key sizes shows little difference. The computation time is less than one seconds when we use smaller matrices (128 by 128) to perform the permutation. Note that even with a relatively large matrix (1024 by 1024), the trapdoor generation time is still under two seconds. It is worth mentioning that the number of query keyword doesn't affect the trapdoor generation time

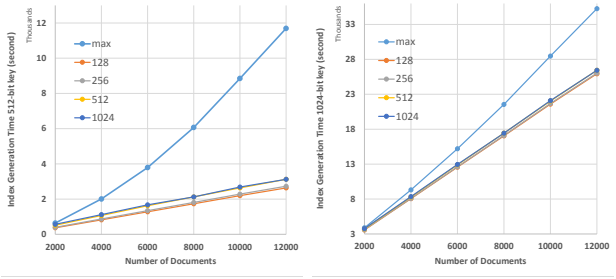
[1]The source code available at https://github.com/mikeivanov/paillier

[2]Enron dataset, http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/

TABLE I
PERFORMANCE COMPARISON

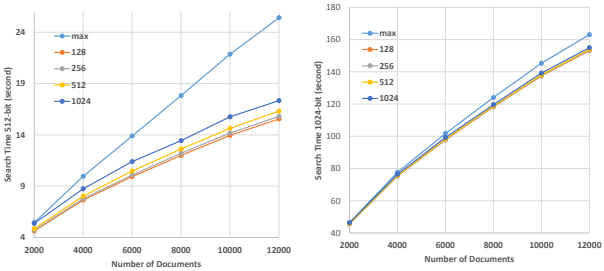
	Baek et al. [19]	Rhee et al. [20]	Zhao et al. [21]	Li et al. [7]	Sun et al. [12]	Ours
Setup	M	2E	M	9mM	(3n+1)E+e	2E
Index	nm(E+M+P+2e)	nm(2E+P+e)	nm(4M+P+2e)	n(m^2+3m)M	m(n+2)E	mLE+m^3M
Trapdoor	P+M	2E+2P	3M+4P+e	(m^2+3m)M	(2n+1)E	m^2M+E
Search	nm(M+e)	nm(2E+P+e)	nm(2M+P+4e)	(m+3)e	(n+1)e+(n+2)M+E	(m^2+1)M+mE

m is the size of the dictionary, n is the size of the document set, L is a constant decided by the dataset.



(a) 512-bit key (b) 1024-bit key

Fig. 5. Index generation time for different key size and matrix size



(a) 512-bit key (b) 1024-bit key

Fig. 7. Search time for different key sizes and matrix sizes

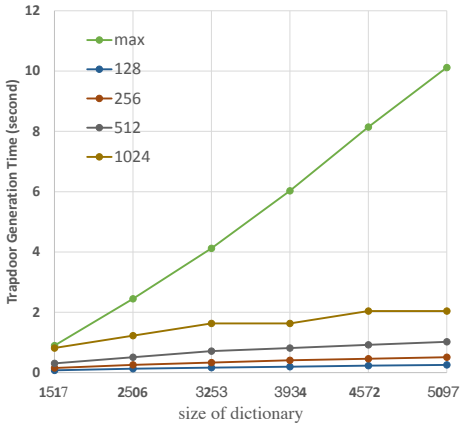


Fig. 6. Trapdoor generation time respect to the size of the dictionary

because we hide the degree of the trapdoor polynomial by padding random terms.

C. Cloud server computation

After receiving the trapdoor from the user, the cloud server needs to multiply the trapdoor with the dictionary matrix. The consumed time is shown in Fig.7. The computation includes a matrix multiplication and m encryptions. As shown in the figure, using smaller matrices can improve the efficiency but the improvement is not much especially when using 1024-bit keys encryptions. This is because the encryption takes a large part of the computation. However, the encryption can be accelerated through parallel processing. Therefore, the search time can be further reduced if multiple threads are used by the cloud server which also has more computation power than our proof-of-concept environment.

D. Comparison between Li et al's scheme and ours

We compare the index generation time and the search time with Li et al's scheme [7]. Li et al's scheme focuses on matching queries with the attributes of documents of which the number is supposed to be small. Therefore, their scheme cannot handle general text keyword search which is the object of our scheme. As reported in their original paper, the index generation time for each index, i.e., each document, is around 30 seconds with 70 attributes which equivalents to 70 keywords in our scheme. Compared with theirs, our scheme is extreme efficient because their scheme heavily relies on pairing operation and the operations over elliptical curve. Although their scheme has different objective, the comparison demonstrates the same technique is not suitable for generating secure index of general keyword search tasks.

TABLE II
DATASET STATISTIC

# of Documents	2000	4000	6000	8000	1000	12000
# of Keywords	1517	2506	3253	3934	4572	5097

VIII. CONCLUSION

In this paper, we proposed a novel construction of a public-key searchable encryption scheme based on inverted index. Our scheme overcomes the one-time-only search limitation in the previous schemes. Our probabilistic trapdoor generation algorithm prevents the cloud server from linking the trapdoors. Our scheme also hides the number of keywords in the query. Additionally, our scheme supports multi-keywords conjunctive search. Because the access pattern has been identified as a serious security threat in searchable encryption schemes, our scheme can be integrated with blind storage technique to

further protect the access pattern. Compared with the exiting public-key based schemes that heavily rely on expensive pairing operations, our scheme is more efficient by using only multiplications and exponentiations. Finally, we validated the practicality of the proposed scheme by implementing a prototype of the scheme and evaluated the performance using the Enron email dataset. The results showed that our scheme features reasonable index construction for a large document set. The trapdoor generation time and the search efficiency outperform the pairing based solutions. Furthermore, our scheme scales well when handling large document set, which makes our scheme ideal for real-world scenario.

ACKNOWLEDGMENT

This work was supported in part by US National Science Foundation under grant CNS-1217889 and National Natural Science Foundation of China under grants 61202034 and 61232002.

REFERENCES

[1] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[2] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds. Springer Berlin Heidelberg, 2004, vol. 3027, pp. 506–522.

[3] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Springer Berlin Heidelberg, 2005, vol. 3531, pp. 442–455.

[4] M. Bellare, A. Boldyreva, and A. ONeill, "Deterministic and efficiently searchable encryption," in *Advances in Cryptology - CRYPTO 2007*, ser. Lecture Notes in Computer Science, A. Menezes, Ed. Springer Berlin Heidelberg, 2007, vol. 4622, pp. 535–552.

[5] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–5.

[6] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 829–837.

[7] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 383–392.

[8] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 393–402.

[9] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud." in *NDSS*. The Internet Society, 2012.

[10] M. Li, S. Yu, W. Lou, and Y. Hou, "Toward privacy-assured cloud data services with flexible search functionalities," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, June 2012, pp. 466–470.

[11] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 71–82.

[12] W. Sun, S. Yu, W. Lou, Y. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 226–234.

[13] B. Wang, S. Yu, W. Lou, and Y. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 2112–2120.

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 79–88.

[15] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 965–976.

[16] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage." *IACR Cryptology ePrint Archive*, vol. 2014, p. 219, 2014.

[17] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds. Springer Berlin Heidelberg, 2004, vol. 3027, pp. 1–19.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, Proceedings. 2000 IEEE Symposium on*, 2000, pp. 44–55.

[19] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Computational Science and Its Applications ICCSA 2008*, ser. Lecture Notes in Computer Science, O. Gervasi, B. Murgante, A. Lagan, D. Taniar, Y. Mun, and M. Gavrilova, Eds. Springer Berlin Heidelberg, 2008, vol. 5072, pp. 1249–1259.

[20] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems and Software*, vol. 83, no. 5, pp. 763 – 771, 2010.

[21] Y. Zhao, X. Chen, H. Ma, Q. Tang, and H. Zhu, "A new trapdoor-indistinguishable public key encryption with keyword search," *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications*, vol. 3, no. 1/2, pp. 72–81, 2012.

[22] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[23] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology EUROCRYPT 99*, ser. Lecture Notes in Computer Science, J. Stern, Ed. Springer Berlin Heidelberg, 1999, vol. 1592, pp. 223–238.

[24] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[25] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." in *NDSS*, 2012.

[26] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[27] Lucene, "The Lucene search engine," 2005.

[28] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. Te Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit rsa modulus," in *Proceedings of the 30th Annual Conference on Advances in Cryptology*, ser. CRYPTO'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 333–350.