View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Crossref

Hindawi Publishing Corporation
The Scientific World Journal
Volume 2013, Article ID 860621, 13 pages
http://dx.doi.org/10.1155/2013/860621

Research Article

Query-Biased Preview over Outsourced and Encrypted Data

Ningduo Peng, Guangchun Luo, Ke Qin, and Aiguo Chen

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China

Correspondence should be addressed to Ningduo Peng; nindo_academia@163.com

Received 14 June 2013; Accepted 16 July 2013

Academic Editors: T.-Y. Chang and C. L. Hsu

Copyright © 2013 Ningduo Peng et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

For both convenience and security, more and more users encrypt their sensitive data before outsourcing it to a third party such as cloud storage service. However, searching for the desired documents becomes problematic since it is costly to download and decrypt each possibly needed document to check if it contains the desired content. An informative query-biased preview feature, as applied in modern search engine, could help the users to learn about the content without downloading the entire document. However, when the data are encrypted, securely extracting a keyword-in-context snippet from the data as a preview becomes a challenge. Based on private information retrieval protocol and the core concept of searchable encryption, we propose a single-server and two-round solution to securely obtain a query-biased snippet over the encrypted data from the server. We achieve this novel result by making a document (plaintext) previewable under any cryptosystem and constructing a secure index to support dynamic computation for a best matched snippet when queried by some keywords. For each document, the scheme has $O(d)$ storage complexity and $O(\log(d/s) + s + d/s)$ communication complexity, where d is the document size and s is the snippet length.

1. Introduction

Cloud storage provides an elastic, highly available, easily accessible, and cheap data repository to users who do not want to maintain their own storage or just for convenience, and such a way of storing data becomes more and more popular. In many cases, especially when the users want to store their sensitive data such as business documents, it requires the security guarantees against the cloud provider since an internal staff may access to the data maliciously. Directly encrypting the sensitive documents using traditional encryption techniques such as AES is not an ideal solution since the user will lose the ability to effectively search for the desired documents.

One solution for effectively searching over encrypted data is *searchable encryption* technique. It enables a user to securely outsource his private documents to a third party while maintaining the ability to search the documents by keywords. The scenario is simple: the user submits some encrypted keywords to the server, and then the server performs the search and returns the encrypted documents which contain the queried keywords. However, current searchable encryption techniques either directly return the matched documents or return in the first round some limited information (guided mode) which is prestored in metadata, such as the name and a short static abstract for each matched document. The more documents stored, the more possible matched results will be, and finding the desired documents also becomes a problem. Moreover, the bandwidth cost must be taken into consideration such that returning a large amount of matched documents seemed to be impractical.

Another solution for effectively searching for the desired data is through content preview, which is the main topic of this paper. In modern search engine, if a user searches for a web page by keywords, the search engine will return the name, URI, and a small *query-biased snippet* for each matched page. The snippet explains why such page is matched. Then the user could make a final choice and selectively browse the needed pages without opening all matched links. The same way could be used for searching the desired encrypted documents since the scenario is the same. It could also be combined with searchable encryption to improve the user experience.

However, obtaining a query-biased snippet from an encrypted data is quite challenging. For a general search engine, in order to get a query-biased snippet from

a plaintext, it must scan each matched document dynamically, extract the snippets where the keywords occur, then rank the results and finally return the *top-ranking snippet*. While data is encrypted, dynamic scanning becomes quite impossible. Precomputing a snippet file for preview is also impossible because there is no way to know in advance what the queried keywords are, and building all static (keyword, snippet) pairs for each document costs too much storage space even far more than the document itself. Thus, we consider dividing a document to many equal-size encrypted snippets and preconstruct an index to address each snippet. The index stores the information about the keyword frequency in each snippet, which enables the server to dynamically calculate the best snippet for the user when queried by multiple keywords.

There are two major security problems. First, the snippet is the part of a document; therefore the encryption scheme used may affect the snippet retrieval. We use a pad-and-divide scheme to preprocess the document to make it compatible with any cryptosystem such as DES and RSA. Second, the information in the index is private, and no partial information about the document should be leaked to the server. Therefore, we encrypt the index based on the core method of searchable encryption. Since each keyword maps an entry in the index, if queried by some keywords, directly returning the related score information without calculating leaks the information about the number of queried keywords (equals to the number of returned entries) to an eavesdropper, and it also costs multiple communication bandwidth as the number of requested keywords increases. A *homomorphic encryption* scheme could be adopted such that the server could directly operate over the encrypted data and produce a single result, while keeping the ciphertext still secure. However, homomorphic encryption scheme is often costly when dealing with a large amount of data. Observing that all the data are very small, we propose a novel lightweight substitution for homomorphic encryption to construct such secure index.

In this paper, our contributions are the following. (1) To the best of our knowledge, we formalize the problem of securely retrieving query-biased snippet over encrypted data for the first time. We generalize the notion of *secure query-biased preview* (SecQBP) and its security model. (2) We propose a lightweight solution to deal with matrix data with partial homomorphic property, named *matrix additive coding* (Matrix-AC), which could efficiently add two rows of small numbers while keeping the data still encrypted. (3) Based on Matrix-AC and private information retrieval protocol, we construct a *secure additive ranking index* (SecARI) that enables the server to efficiently compute the top-ranking snippet over encrypted data while no partial information about the document is leaked, and then we propose the complete construction to realize SecQBP and prove that it is secure under our security model. (4) We propose a high level solution to combine the preview scheme with searchable encryption technique, which greatly improves the user experience.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the notations and preliminaries. Section 4 presents our proposed additive coding scheme. In Section 5 we formally define the preview scheme and its security model and present the construction in detail. We present the application in searchable encryption and analyze the performance in Section 6. Section 7 concludes this paper.

2. Related Work

We categorize the related work into four topics, and each topic is summarized separately.

2.1. Query-Biased Snippet. Query-biased snippet refers to a piece of the content in a document that contains the queried keywords. Query-biased snippet generation schemes are widely used in modern search engine. It is also named *dynamic summary* or *keyword-in-context* (KWIC) snippet generation. The term was used firstly in [1]. The improvements were introduced in [2–6]. However, as far as we know, all query-biased schemes are focused on dynamically retrieving snippets from the plaintext. If the document is encrypted, dynamic scanning becomes impossible. Static preview refers to a snippet summarizing the content in advance, which is always the same regardless of the query. It is generally composed of either a subset of the content or metadata associated with the document. A lightweight static preview scheme over the encrypted data was introduced in [7]. For more details, please refer to [8] for a survey of the recent preview schemes.

2.2. Searchable Encryption. Our proposed scheme and security model are based on searchable encryption technique. The basic goal of searchable encryption is to enable a user to privately search over encrypted data by keywords. The first scheme was introduced in [9]. Later on, many index-based symmetric searchable encryption schemes were proposed. The first secure index was introduced in [10], and the security model of adaptive chosen keyword attack (IND-CKA) was also introduced. Reference [11] introduced two constructions to realize symmetric searchable encryption: the first is SSE-1 which is nonadaptive and the second is SSE-2 which is adaptive. A generalization for symmetric searchable encryption was introduced in [12]. Another type of searchable encryption schemes is public-key based. The first scheme was introduced in [13], the improved definition was introduced in [14], and the strongest security model was introduced in [15].

There are many functional extensions for the basic searchable encryption schemes. Reference [16] introduced a scheme supporting conjunctive keyword search. References [17–19] introduced ranked keyword search over encrypted data. References [20–22] introduced fuzzy keyword search over encrypted data. Similar to fuzzy keyword search but different, [23, 24] introduced similarity search over encrypted data.

2.3. Homomorphic Encryption. Our proposed additive coding method is based on the core concept of homomorphic encryption. The classical homomorphic encryption schemes

are based on group operation such as the unpadded RSA in [25], the variant of ElGamal introduced in [26], Goldwasser and Micali's bit homomorphic encryption scheme introduced in [27, 28], and Paillier's encryption scheme introduced in [29]. Many improvements have been proposed based on these classical series of schemes. The referred schemes are public-key based, and few symmetric homomorphic schemes have been proposed. The series of symmetric homomorphic schemes which is based on one-time pad was introduced in [30]. Some ring-based homomorphic schemes have been proposed recently, which are also referred to as full homomorphic encryption, such as the one in [31] that is based on ideal lattices and the one in [32] that does not require ideal lattices.

2.4. Private Information Retrieval. We encapsulate a private information retrieval (PIR) protocol and extend the use of it in our scheme. PIR schemes allow a user to privately retrieve the ith bit of an n-bit database. The notion was fist introduced in [33] by Chor et al., and the notion of private block retrieval (PBR) was also introduced. Kushilevitz and Ostrovsky introduced a single-server and single-round computational PIR scheme in [34], which achieves communication complexity of $O(\sqrt{n})$ for the basic scheme and could achieve $O(n^{\epsilon})$ with arbitrary small ϵ theoretically ($2^{O(\sqrt{\log n \log\log n})}$ is achieved assuming security parameter is polylogarithmic in n). In [35], Cashin et al. introduced a single-database PIR scheme with polylogarithmic communication complexity for the first time, about $O(\log^8 n)$ as suggested. Gentry and Ramzan introduced a PBP scheme with $O(k + d)$ communication cost in [36], where $k \geq \log n$ is a security parameter that depends on n, which is nearly optimal.

3. Notations and Preliminaries

3.1. Basic Notations. We write $x \leftarrow_U X$ to represent sampling element x uniformly random from a set X and write $x \leftarrow \mathscr{A}$ to represent the output of an algorithm $\mathscr{A}$. We write $a \parallel b$ to refer to the concatenation of two strings a and b. We write $|A|$ to represent its cardinality when A is a set and write $|a|$ to represent its bit length if a is a string. We write $\oplus$ to represent bitwise exclusive OR (XOR) and "$\ll n$" to represent bitwise shift left for n bits. We write $\lceil x \rceil$ to represent the least integer less than or equal to x. We write $\underline{s}_b$ to represent a bit string that contains either 0 or 1 (e.g., $\underline{001101}_b$). A function $\mu(k) : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if for every positive polynomial $p(\cdot)$ there exists an inter $N > 0$ such that for all $k > N$, $|\mu(k)| < 1/p(k)$. We write poly(k) and negl(k) to denote polynomial and negligible functions in k, respectively.

We write $\Delta = (w_1, \ldots, w_n)$ to present a dictionary of n words in lexicographic order. We assume that all words are of length polynomial in k. We write d to refer to a document that contains poly(k) words. We write $\overline{d}$ to represent the identifier of d that uniquely identifies the document, such as a memory location. We write s to refer to a snippet (50 characters in general) extracted from the document and write $\overline{s}$ to represent the identifier of s, such as the position in the document.

3.2. Cryptographic Primitives. A function $f : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^m$ is pseudorandom if it is computable in polynomial time in k and for all polynomial size adversaries $\mathscr{A}$, it cannot be distinguished from random functions. If f is bijective then it is a pseudorandom permutation. We write the abbreviation PRF for pseudorandom functions and PRP for pseudorandom permutations.

Let ES represent an encryption scheme. Let $\texttt{ES.Gen}(1^k)$ represent the key generation algorithm (k is the secure parameter). Let $\texttt{ES.Enc}_K(d)$ represent the encryption algorithm that encrypts data d using key K, and let $\texttt{ES.Dec}_K(c)$ represent the decryption algorithm that decrypts data c to gain the plaintext d. In our scheme, a lot of data will be encrypted using the same key; therefore the encryption scheme must be at least CPA (chosen plaintext attack) and CCA (chosen ciphertext attack) secure. For example, ECB (electronic codebook) mode in DES or RSA without OAEP (optimal asymmetric encryption padding) should not be used.

3.3. Homomorphism. Let $\mathscr{M}$ denote the set of the plaintexts, let $\mathscr{C}$ denote the set of the ciphertexts, let $\odot$ denote the operation between the plaintexts and $\otimes$ the operation between the ciphertexts, and let "$\leftarrow$" denote "directly compute" without any intermediate decryption. An encryption scheme is said to be homomorphic if for any given encryption key k, the encryption function E or the decryption function D satisfies

$$\forall m_1, m_2 \in \mathscr{M}, \quad E(m_1 \odot m_2) \longleftarrow E(m_1) \otimes E(m_2), \quad (1)$$

$$\forall c_1, c_2 \in \mathscr{C}, \quad D(c_1 \otimes c_2) \longleftarrow D(c_1) \odot D(c_2). \quad (2)$$

Sometimes, property (2) is also referred to as homomorphic decryption. If the operation is upon a group, we say it is a group homomorphism. If the operation is upon a ring, we say it is a ring homomorphism and is also referred to as full homomorphism. If the operator is addition, we say it is additively homomorphic, and if the operator is multiplication, we say it is multiplicatively homomorphic.

3.4. Private Block Retrieval Protocol. Let $B = (B_1, \ldots, B_n)$ represent a database of n blocks; all blocks have equal size d. The user wants to privately retrieve the ith block from the server; therefore he runs a private block retrieval protocol. At a high level, we define the single database and single round computational PBR as follows.

Definition 1 (computational PBR protocol). A computational PBR protocol scheme is a collection of four polynomial-time algorithms $\texttt{CPBR} = (\texttt{Setup}, \texttt{Query}, \texttt{Response}, \texttt{Decode})$ such that we have the following.

$P \leftarrow \texttt{Setup}(B)$ is a probabilistic algorithm that takes as input the database B and outputs a parameter set P. It is run by the database owner, and P is known to all users.

$t \leftarrow \texttt{Query}(i)$ is a probabilistic algorithm that takes as input a block index i and outputs a token t. It is run by the user. t is sent to the server.

$r \leftarrow$Response(t) is a deterministic algorithm that takes as input the requested token t and outputs a result r. It is run by the server. r is sent to the user.

$B_i \leftarrow$Decode(r) is a deterministic algorithm that takes as input the response r from the server and outputs the requested data block B_i. It is run by the user.

In our preview scheme, we adopt the computational PBP scheme as a primitive introduced in [36]. In the setup algorithm, we set the database size as the maximal possible document size (e.g., 10 MB) and reuse prime number set and prime power set in all documents. The communication complexity is $O(\log |d| + |s|)$ where $|d|$ is the document length and $|s|$ is the snippet length.

4. Secure Additive Coding

Before introducing the preview scheme, we first introduce a novel coding method called matrix additive coding (Matrix-AC) that enables addition of two rows in a matrix in a homomorphic fashion, which is very fast and suitable for dealing with small numbers (the integer is coded to a specific bit string) and is especially useful for computing statistical table in encrypted form. Since all operated integers are correlative, it is not a homomorphic encryption scheme which could encrypt data independently.

Matrix-AC is used in the preview scheme to construct the secure additive ranking index (SecARI). Becouse a large number of small numbers will be calculated in the preview scheme, using homomorphic encryption schemes is costly. Therefore, we use Matrix-AC scheme as a substitution for homomorphic encryption scheme to achieve optimal performance.

We note that, for all the schemes (including the preview scheme in the next section), we only consider the confidentiality of the data. Mechanism about protecting data integrity is out of the scope of this paper.

4.1. Basic Idea. The basic idea of coding small integers $\mathbb{S}_N = (0, 1, 2, \ldots, N)$ with homomorphic property is simple: we consider an integer vector $\mathbf{m} = (m_1, \ldots, m_n)$, where $m_i \in \mathbb{S}_N$ and $\sum_{i=1}^{n} m_i \leq N$. We define a "vernier" that has N bits, and each integer m_i is mapped to such vernier for m_i bits in different position. A global cursor g is autoincreased to process the mapping. To code a message, a random string as a one-time-pad key is used and XORed with the mapped data. The decoding is simple: just operate $\oplus$ with the key and count the number of bit-1 to make reverse mapping.

For example (as shown in Figure 1), consider a vernier that has 8 bits, and we map three integers $(2, 1, 3)$ as a vector to three pairs $(2, \underline{00000011}_b)$, $(1, \underline{00000100}_b)$, and $(3, \underline{00111000}_b)$. It is easy to see that the homomorphic property holds as $2 + 3 = 5$, and $\underline{00000011}_b \oplus \underline{00111000}_b = \underline{00111011}_b$ which has exactly 5 bit-1. Thus, let the vector be $(m_1 = 2, m_2 = 1, m_3 = 3)$, and let the keys be (K_1, K_2, K_3) that each key is a random string; then the ciphertext vector will be $(c_1 = \underline{00000011}_b \oplus K_1$, $c_2 = \underline{00000100}_b \oplus K_1$, $c_3 = \underline{00111000}_b \oplus K_1)$. To perform addition for any two plaintexts $m = m_1 + m_2$, the server could directly compute the corresponding ciphertexts $c = c_1 \oplus c_2$ and the decryption key becomes $K = K_1 \oplus K_2$. Using the new key, it is easy to decrypt the ciphertext and count the number of bit-1 to restore the result.

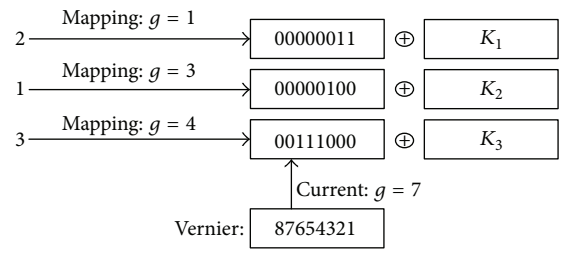


Figure 1: Example of vernier mapping and the basic coding procedure.

The problem of the basic scheme is that the vernier may be used up. That is why we set the restriction that a vernier is just used in a single vector. Another drawback is that, as the max N increases, the length of the vernier also increases in linear $O(N)$. Thus, the targeted data must be small enough to save storage space. A good application is not for dealing with few such integers but for computing a large number of small data in parallel.

4.2. Coding a Matrix. We extend the basic idea to code the data matrix. Let $A_{m \times n}$ represent a matrix with m rows and n columns, let a_{ij} represent the element in row i and column j, and a_i represent the ith row. Matrix-AC scheme is described in Algorithm 1. Note that there are n cursors that control the mapping for each column.

Let us check the homomorphism for decoding: let D represent the decryption algorithm, for arbitrary two ciphertext rows c_1 and c_2, $D(c_1 \oplus c_2) = D(c_1) + D(c_2)$, where the decryption key for $c_1 \oplus c_2$ is $K_1 \oplus K_2$.

There is a problem if the scheme is directly used in the application. In the real world, there is no way to directly represent, for example, data of 5 bits (there is an extended "bitset" class in C++, but it treats the bits as a set, and all operations are performed over set, and it is very slow). In computer, the data is represented by "byte" that a valid number is stored in such a byte. Thus, 5-bit data is stored in one byte (8 bits) as a "character," 12-bit data is stored in two bytes (16 bits) as a "short integer," and a 20-bit data is stored in four bytes (32 bits) as an "integer." Thus, the XOR operation is performed over byte, and the data should be extended to such standard length. However, since all data in Matrix-AC are in fact a bit string, sometimes the data in the same row could be "chained" together. For example, suppose $N = 5$ and there are 6 data in a row; the row could be chained to a 30-bit string and stored in a 32-bit integer. Another problem is that the "bit-counting" algorithm is realized indirectly by "mod 2" operation or setting N masks to see if the masked bit is 1. Therefore, the performance would be improved if some dedicated hardware directly dealing with bits is used.

Input m random strings $\mathbf{K} = (K_1, \ldots, K_m)$ of length $n \cdot N$
Code$_{\mathbf{K}}(X_{m\times n})$:
(1) check the validity of the input, continue if $m_{ij} \in \mathbb{S}_N$ and for all j, $\sum_{k=1}^{m} c_{kj} \le N$, or output Ø otherwise
(2) initialize a ciphertext matrix $C_{m\times n} = (c_{ij})$, each element is set to 0 (N bits), and initialize n cursors $g_1, \ldots, g_n$ and set each $g_j = 1$ (N bits)
(3) **for** $(i = 1; i \le m; i++)$ **do**
(4) **for** $(j = 1; j \le n; j++)$ **do**
(5) **for** $(k = 1; k \le x_{ij}; k++)$ **do**
(6) $c_{ij} \leftarrow c_{ij} \oplus g_j$
(7) $g_j \leftarrow g_j << 1$
(8) **end for**
(9) **end for**
(10) encrypt the ith row as $c_i \leftarrow c_i \oplus K_i$
(11) **end for**
(12) output $C_{m\times n}$
Decode $_{K_i}(c_i)$:
(1) initialize a temporary data row $\mathbf{t} = (t_1, \ldots, t_n)$ with each element be N bits, and set $\mathbf{t} = c_i \oplus K_i$
(2) for each t_j in $\mathbf{t}$, let the binary form of t_j be $b_N b_{N-1} \ldots b_1$, then $x_{ij} = \sum_{k=1}^{N} b_k$
(3) output $\mathbf{x_i} = (x_{i1}, \ldots, x_{in})$

Algorithm 1: Secure additive coding for matrix.

4.3. Proof of Security. Intuitively, the scheme is secure if any two matrices (the numbers of elements are the same) prepared by the adversary are indistinguishable, which also implies that any two elements from the same matrix are indistinguishable. We define the security of Matrix-AC as follows.

Lemma 2. *If $K_1, \ldots, K_n$ are random strings, then Matrix-AC is CPA secure.*

Proof (sketch). We briefly prove the scheme since the mechanism is simple. We describe a PPT simulator $\mathcal{S}$ for all PPT adversaries $\mathcal{A}$. $\mathcal{S}$ generates matrix $\mathbf{X}^*_{m\times n}$ with m random strings of length $n \cdot N$. For any row x_i in the original matrix $\mathbf{X}_{m\times n}$, no matter how it maps, the last computation is a string XORed with a one-time-pad random string K_i; thus the result is indistinguishable from random. For any two rows x_i and x_j from the same matrix, each row is XORed with different random strings such that the results are indistinguishable from each other. For any two rows x_i and x_i^* from two matrices, as discussed previously, x_i is indistinguishable from the random string x_i^*. □

5. Secure Query Biased Preview Scheme

The preview scheme contains two steps: (1) storage at which the data owner prepares the previewable document and a searchable index; (2) retrieval at which the user privately retrieves the snippet from the server.

The basic idea of constructing a query-biased previewable document is as follows: divide the document into n snippets with equal size, extract keywords from each piece to form a keyword set which records the snippet information as (keyword, frequency) pairs, and build an index to address the snippets according to the distinct keywords. The index R is a $m \times n$ two-dimensional matrix of the form R(keyword, snippet index), and the value is the keyword frequency in the corresponding snippet. An example is shown in Table 1. The keyword is represented by w_i, and the snippet index is represented by s_i.

Table 1: Example of a snippet index.

Keyword	s_1	s_2	⋯	s_n
w_1	2	1	⋯	2
w_2	1	2	⋯	1
⋯	⋯	⋯	⋯	⋯
w_i	2	5	⋯	3
⋯	⋯	⋯	⋯	⋯
w_m	3	2	⋯	2

The main process of retrieving the best snippet by multi keywords follows the following steps. The user submits multikeywords to the server. The server retrieves the multirows in the index according to the submitted keywords and adds the rows together. The result is a single entry that contains the information about the best matched snippet. The user decrypts the entry, selects the snippet identifier (index number) with the highest score (for simplicity, the score equals the frequency), and privately retrieves the snippet from the server by running a PBR protocol. In order for the server to perform the "addition" operation over the encrypted data, a homomorphic encryption scheme could be used to encrypt the index. We adopt Matrix-AC as the encryption scheme

instead of a standard homomorphic encryption scheme as discussed previously.

Now we begin to introduce the definition and the security model of the preview scheme. Note that we assume the server is honest but curious. Additional methods could be added to make those solutions robust against malicious attack; however, we restrict our discussion on honest-but-curious fashion. We also note that all documents are treated as text files the same way as search engine does. For example, if a document is a web page, the style tags will be pruned.

5.1. Scheme Definition. The secure-query biased preview (SecQBP) scheme contains two parties: a user U and a remote server S. U encrypts his private document d to D, generates a secure additive ranking index (SecARI) H, and then outsources them to S. S stores the document, performs the computation for the scores when queried by multiple keywords, and returns the result to U. U then selects the best snippet indexed by i and privately retrieves it from S.

Without loss of generality, we consider the construction for a single document. The scheme could be extended to a document collection with ease. Now we define the SecQBP scheme as follows.

Definition 3 (secure query-biased preview scheme). SecQBP scheme is a collection of six polynomial-time algorithms $\mathtt{SecQBP} = (\mathtt{Gen}, \mathtt{Setup}, \mathtt{Query}, \mathtt{ComputeScore}, \mathtt{DecScore}, \mathtt{DecSnip})$ as follows.

$K \leftarrow \mathtt{Gen}(1^k)$ is a probabilistic algorithm that takes as input a security parameter k and outputs the secret key collection K. It is run by the user, and the keys are kept secret.

$(D, H) \leftarrow \mathtt{Setup}_K(d)$ is a probabilistic algorithm that takes as input a document d and outputs a encrypted document D (using any cryptosystem) and an index H. It is run by the user, and D, H are outsourced to the server.

$q \leftarrow \mathtt{Query}_K(\mathbf{w})$ is a deterministic algorithm that takes as input the queried multiple keywords $\mathbf{w} = (w_1, \ldots, w_n)$ and outputs a secret query token q. It is run by the user, and q is sent to the server.

$r \leftarrow \mathtt{ComputeScore}(q, H)$ is a deterministic algorithm that takes as input the secret query q and the index H and outputs the result r that contains the final score information about each snippet. It is run by the server.

$i \leftarrow \mathtt{DecScore}_K(\mathbf{w}, \overline{d}, r)$ is a deterministic algorithm that takes as input the queried keywords $\mathbf{w}$, the document identifier $\overline{d}$, and the query result r and outputs the snippet index number i. It is run by the user.

$s_i \leftarrow \mathtt{DecSnip}_K(D_i)$ is a deterministic algorithm that takes as input the ciphertext D_i and outputs the recovered plaintext snippet s_i. It is run by the user. Note that, if the user retrieves the entire encrypted document, he could decrypt the document by decrypting each snippet.

5.2. Security Model. Informally speaking, SecQBP must guarantee that, first, given the encrypted document c and the index H, the adversary cannot learn any partial information about the document; second, given a sequence of queries $q = (q_1, \ldots, q_n)$, the adversary cannot learn any partial information about the queried keywords and the matched snippet (including the index number and the content). We now present the security definition for adaptive adversaries.

Definition 4 (semantic security against adaptive chosen keyword attack, CKA2-security). Let $\sum$ = (SecQBP algorithm + SecQBP protocol) be the preview scheme. Let $k \in \mathbb{N}$ be the security parameter. one considers the following probabilistic experiments, where $\mathcal{A}$ is an adversary and $\mathcal{S}$ is a simulator.

$\mathrm{Real}_{\Sigma,\mathcal{A}}(k)$: the challenger runs $\mathtt{Gen}(1^k)$ to generate the key K. $\mathcal{A}$ generates a document d and receives $(D, H) \leftarrow \mathtt{Setup}_K(d)$ from the challenger. $\mathcal{A}$ makes a polynomial number of adaptive queries $\mathbf{w}_1, \ldots, \mathbf{w}_n$ (each set $\mathbf{w}_i$ contains multiple keywords in d), and for each queried keyword set $\mathbf{w}_i$, $\mathcal{A}$ receives a query token $q_i \leftarrow \mathtt{Query}_K(\mathbf{w}_i)$ from the challenger. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.

$\mathrm{Sim}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$: $\mathcal{A}$ generates a document d. Given only the size $|d|$, $\mathcal{S}$ generates and sends (D^*, H^*) to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive queries $\mathbf{w}_1, \ldots, \mathbf{w}_n$ (each set $\mathbf{w}_i$ contains multiple keywords in d), and for each queried keyword set $\mathbf{w}_i$, $\mathcal{A}$ receives a query token q_i^* from $\mathcal{S}$. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.

We say that SecQBP is semantic secure against adaptive chosen keyword attack if, for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$\left|\Pr\left[\mathrm{Real}_{\sum,\mathcal{A}}(k) = 1\right] - \Pr\left[\mathrm{Sim}_{\sum,\mathcal{A},\mathcal{S}}(k) = 1\right]\right| \leq \mathrm{negl}(k), \tag{3}$$

where the probabilities are over the coins of $\mathtt{Gen}$ and $\mathtt{Setup}$ (related to the underlying cryptosystem).

Note that, with q_i or q_i^*, $\mathcal{A}$ could run $\mathtt{ComputeScore}(q, H)$ to get the result r, and any internal state is also captured by $\mathcal{A}$. $\mathcal{A}$ could also send query according to the previous result.

5.3. Concrete Construction. Now we describe the concrete construction for SecQBP. We describe the constructions for some core components, and then represent the complete construction.

5.3.1. Encrypting a Document. We consider the problem of extracting keywords from a document. In general, a keyword is followed by a separator. Thus, in a general snippet of 50 characters, no more that 25 keywords are contained. Another problem is that not all words are keywords, and such words do not need indexing, for instance, the words "a," "the," and "and." This kind of words can be found in most of the sentences such that it is useless as a key to index a file. They

Input: a document d, the encryption key K
Output: the encrypted document $D = (D_1, \ldots, D_n)$, a keyword-frequency set collection $e = (e_1, \ldots, e_n)$
Method:
(1) padding d according to snippet length $|s|$ as discussed
(2) treat d as $n = |d|/|s|$ pieces $s_1, \ldots, s_n$
(3) **for** $(i = 1; i \leq n; i++)$ **do**
(4) create a keyword-frequency set e_i for s_i
(5) scan s_i for distinct valid keywords. For each keyword, count the keyword frequency, and add the vector (keyword, frequency) to e_i as set element
(6) encrypting the snippet as $D_i \leftarrow \mathtt{ES.Enc}_K(s_i)$
(7) **end for**

Algorithm 2: Encrypting a document: $\mathtt{ED}_K(d)$.

are called stop-word and firstly researched in [37]. The most classical stop word list used abroad is a list of 425 words suggested in [38].

There is a problem that the last word in a snippet may be cut off. In other words, the last word of a snippet may be not short enough to fit the space, and it cannot be split into two words because neither of them is a valid keyword. In a general search engine, such overflowed word is omitted. However, in the scenario of precomputing snippets, if the word is omitted, a keyword may be lost. It means that, when querying the omitted keyword, there will be no matched snippet returned, where actually there is a match for the document. Thus, we add the full word to both the keyword sets of the snippets which contain part of the keyword.

The basic idea for encrypting a document is dividing the document with equal size; therefore, a padding scheme is needed when the last piece of the document is not long enough. We modify the CBC plaintext padding scheme introduced in [39] to meet our goal. Let $|s|$ represent the length of the snippet; the snippet is treated as a sequence of bytes. If the last snippet is a bytes, then pad the snippet with $|s| - a$ bytes with value $|s| - a$. After decryption, the padding will be deleted to recover the original plaintext. For instance, suppose $|s| = 50$; if the final snippet has 15 plaintext bytes, then pad the snippet as

$$\text{byte}_1 \parallel \text{byte}_2 \parallel \cdots \parallel \text{byte}_{15} \parallel 35 \parallel 35 \parallel \cdots \parallel 35, \quad (4)$$

where there are 35 bytes that have the number 35. If the snippet is divisible by $|s|$, here is 50, then add a new snippet with all bytes being 50:

$$50 \parallel 50 \parallel \cdots \parallel 50. \quad (5)$$

Let d represent a document, and D is the encrypted form of d. We introduce the scheme for encrypting a document, shown in Algorithm 2. In the algorithm, "a valid keyword" means the token is not a separator, not a stop word, and not a random-looking string. A word dictionary could be used to check its validity.

Table 2: Example of a SecARI.

Index	s_1	s_2	$\cdots$	s_n
$\pi_K(w_2)$	c_{21}	c_{22}	$\cdots$	c_{2n}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
PAD	PAD	PAD	PAD	PAD
$\pi_K(w_1)$	c_{11}	c_{12}	$\cdots$	c_{1n}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\pi_K(w_m)$	c_{m1}	c_{m2}	$\cdots$	c_{mn}
PAD	PAD	PAD	PAD	PAD
$\pi_K(w_3)$	c_{31}	c_{32}	$\cdots$	c_{3n}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

5.3.2. Constructing the Secure Index. The secure additive ranking index (SecARI) is the encryption form of the snippet index, as shown in Table 1 (PAD denotes the padding with a random string), and each row is an encrypted entry. For security reason, the number of entries of SecARI must be padded to a certain amount which is independent of the actual number of keywords in the content, or it will leak the information about the number of distinct keywords in the document (it equals the number of rows). An example of a SecARI is shown in Table 2. In the table, π is a pseudorandom permutation which randomizes the order of the keywords, and the value c_{ij} is the encrypted score.

Let us consider the secure amount of the entries. If the document d is small, let a keyword occupy only one byte; then the maximum possible number of keywords is $|d|/2$ (as discussed, a valid keyword is at least 2 bytes); thus, the number of entries must be set to $|d|/2$ (the fractional part is ignored). If the document d is large, the maximum possible number of keywords equals the total number of words in the dictionary. Reference [40] made a detailed word statistical analysis based on 450 million words on Corpus of Contemporary American English (1990–2012). The statistics show that the total words used are about 60000. We set the dictionary used as Δ and define the maximum keyword amount as $|\Delta| = 60000$. Thus, we define the number of entries as follows.

Input:
(1) $\overline{d}$: the document identifier
(2) $e = (e_1, \ldots, e_n)$: the keyword-frequency set collection
(3) $K = (K_m, K_p)$: consists of the master key K_m for row encryption and the permutation key K_p for π
Output: A secure additive ranking index H
Method:
(1) scan e, extract m distinct keywords $(w_1, \ldots, w_m)$
(2) create an $m \times n$ data matrix $X = (x_{ij})$, the value of each data x_{ij} is the frequency of the keyword w_i in the jth snippet
(3) for each row i, the encryption/decryption key is $k_i \leftarrow f_{K_m}(w_i \,||\, \overline{d})$. Thus the keys for all rows form a vector $\mathbf{K}_X = (k_1, \ldots, k_m)$
(4) create an $m \times n$ matrix C, each cell c_{ij} has length N
(5) compute $C \leftarrow \texttt{Code}_{\mathbf{K}_X}(X)$ using Matrix-AC
(6) for all w_i, set $H[\pi_{K_p}(w_i)] = c_i$ where c_i is the ith row of the encrypted matrix C
(7) if $m < N_{\text{ent}}$, set remaining $N_{\text{ent}} - m$ entries of H to random values

Algorithm 3: Constructing SecARI: $\texttt{Index}_K(\overline{d}, e)$.

Definition 5 (number of entries). To guarantee security, the number of entries N_{ent} for a SecARI is

$$N_{\text{ent}} = \begin{cases} \dfrac{|d|}{2}, & \dfrac{|d|}{2} < |\Delta|, \\ |\Delta|, & \dfrac{|d|}{2} \geq |\Delta|. \end{cases} \tag{6}$$

SecARI is in fact a sparse look-up table, and we use indirect addressing method to manage it. Indirect addressing method is also called *FKS dictionary* introduced in [41], which is also adopted in symmetric searchable encryption scheme in [11]. It manages sparse table of the form (address, value). The address is a *virtual address* that could locate the value field. Given the address, the algorithm will return the associated value in constant look-up time and return Ø otherwise.

In addition, we make use of a pseudorandom permutation π to index an entry and a pseudorandom function to generate the one-time-pad keys for Matrix-AC:

$$\begin{aligned} \pi &: \{0,1\}^k \times \{0,1\}^{|w|} \longrightarrow \{0,1\}^{|w|}, \\ f &: \{0,1\}^k \times \{0,1\}^{|w|+|\overline{d}|} \longrightarrow \{0,1\}^{N \cdot n}, \end{aligned} \tag{7}$$

where $|w|$ is the keyword length and $|\overline{d}|$ is the length of the document identifier. N is the upper bound discussed in Matrix-AC and n is the number of snippets that is calculated from the document size. The submitted keyword is encrypted by π such that the server cannot figure out what the keyword the user queries.

Let H be a $\{0,1\}^{|w|} \times \{0,1\}^{N \cdot n} \times N_{\text{ent}}$ data matrix managed by indirect addressing technique as discussed previously. Now we describe SecARI in Algorithm 3.

5.3.3. The Complete Scheme. In order to hide the information about the number of queried keywords, a SecARI is not enough. When the user submits the queried multiple keywords, each query should be of the same length so that an eavesdropper cannot learn the information about the number of keywords in a query. Let the maximum number of keywords allowed in a single query be $W_{\max}$; the remaining space must be padded. The user and the server should initiate a secure channel such as SSL to transport such message, or the padding may be discovered by an eavesdropper. Since the size of a keyword is small, the bandwidth waste of the padding is rather negligible.

We also determine the upper bound N for Matrix-AC. As discussed, a general snippet contains at most 25 keywords; thus we set $N = 32$ (stored as a standard integer).

Let f be the pseudorandom function, and π is the pseudorandom permutation as described previously. Now we describe the complete scheme in Algorithm 4, and describe the storage and retrieval protocol in Protocol 1. The retrieval protocol describes the retrieval of a query-biased snippet from document d by submitting a multikeyword query $\mathbf{w} = (w_1, \ldots, w_n)$.

Note that it is a scenario for a single document. The protocol also works for a document collection. Thus, the user could retrieve multiple snippets for multiple documents in the same round.

5.4. Proof of Security. The server stores the SecARI, performs homomorphic computation for a query, and returns to the user the score information as a single entry. We prove the security by introducing a theorem as follows.

Theorem 6. *If f is a pseudorandom function, if π is a pseudorandom permutation, and if* ES *is CPA and CCA secure, then SecQBP is CKA2 secure.*

Proof. We describe a polynomial-size simulator $\mathcal{S}$, for all polynomial-size adversaries $\mathcal{A}$, $\text{Real}_{\sum,\mathcal{A}}(k)$ and $\text{Sim}_{\sum,\mathcal{A},\mathcal{S}}(k)$

$\mathtt{Gen}\left(1^k\right)$:
(1) sample index keys $\left(K_m, K_p\right) \leftarrow_U \{0,1\}^k$, generate document encryption key $K_d \leftarrow \mathtt{ES.Gen}\left(1^k\right)$
(2) output $K = \left(K_m, K_p, K_d\right)$
$\mathtt{Setup}_K(d)$:
(1) invoke $\mathtt{ED}_{K_d}(d)$ to get e and the encrypted document D
(2) invoke $\mathtt{Index}_{(K_m,\ K_p)}(\overline{d}, e)$ to get the secure index H
(3) output D, H
$\mathtt{Query}_K(\mathbf{w})$:
(1) for each keyword w_i in $\mathbf{w} = (w_1, \ldots, w_n)$, compute $t_i \leftarrow \pi_{K_p}(w_i)$
(2) put $(t_1, \ldots, t_n)$ into query q and pad it to length $W_{\max}$
(3) output the query q
$\mathtt{ComputeScore}(q, H)$:
(1) let k represent the snippet amount, unpack q to get the queried tokens $t = (t_1, \ldots, t_n)$, set a flag $F = 1$
(2) select a subset of $t : t' = (t'_1, \ldots, t'_m)$ where t'_i is in H. If no element is in H, then set $F = 0$
(3) create the result $h = H[t'_1] \oplus \cdots \oplus H[t'_m]$ if $F == 1$, or else randomly select an index i and set $h = H[i]$
(4) put t' into query q' and pad it to length $W_{\max}$
(5) output $r = (h, q', F, k)$
$\mathtt{DecScore}_K\left(\mathbf{w}, \overline{d}, r\right)$:
(1) unpack $r = (h, q', F, k)$ and get $t' = (t'_1, \ldots, t'_m)$
(2) **if** the flag $F == 0$ **then**
(3) randomly select an index $i \in [1, k]$
(4) **else**
(5) according to t', generate the decryption key $k_j \leftarrow f_{K_m}(w_j \,||\, \overline{d})$ for each matched keyword w_j in $\mathbf{w}$
(6) compute the decryption key $k_r = k_1 \oplus k_2 \oplus \cdots \oplus k_m$
(7) invoke $a \leftarrow \mathtt{Decode}_{k_r}(h)$ using Matrix-AC
(8) choose a snippet number i in a with the highest score
(9) **end if**
(10) output the snippet index i
$\mathtt{DecSnip}_{K_d}(D_i)$: output the plaintext snippet $D_i \leftarrow \mathtt{ES.Dec}_{K_d}(D_i)$

Algorithm 4: SecQBP algorithm.

Storage:
(1) the user U runs $\mathtt{Gen}\left(1^k\right)$ to generate the key K
(2) U runs $\mathtt{Setup}_K(d)$ to get the encrypted document D and the index H, and sends (D, H) to the server S
Query:
(1) U runs $\mathtt{Query}_K(\mathbf{w})$ to get a token q and sends it to S
(2) S runs $\mathtt{ComputeScore}(q, H)$ to produce the score result r, and sends it to U along with document identifier $\overline{d}$
(3) U runs $\mathtt{DecScore}_K\left(\mathbf{w}, \overline{d}, r\right)$ to get the index number i (best matched snippet)
(4) U runs a CPBR protocol as discussed, generates a query token t from $\mathtt{CPBR.Query}(i)$ and sends t to S
(5) S responses with o from $\mathtt{CPBR.Response}(t)$
(6) U runs $\mathtt{CPBR.Decode}(o)$ to get the encrypted snippet D_i
(7) U runs $\mathtt{DecSnip}_{K_d}(D_i)$ to get the plaintext s_i

Protocol 1: SecQBP protocol.

Table 3: Comparisons of preview schemes.

	Data type	Preview mode	Round	Communication	Storage	Computation
General search engine [5]	Plaintext	Query biased	1	$O(s)$	d	$O(d)$
Content mask [7]	Plaintext or ciphertext	Static	1	$O(s)$	$O(d)$	$O(1)$
Our scheme	Ciphertext	Query biased	2	$O(\log(d/s) + s + d/s)$	$O(d)$	$O(1)$

are indistinguishable. Consider the simulator that given the size of the document $|d|$, $\mathcal{S}$ generates the data as follows.

(1) (Simulating H^*) $\mathcal{S}$ computes $m = N_{\text{ent}}$, $n = \lceil |d|/|s| \rceil$. For $1 \le i \le m$, $\mathcal{S}$ generates a string $a_i^* \parallel c_i^*$ such that each a_i^* is a distinct string of length $|w|$ chosen uniformly at random, and each c_i^* is a string of length $N \cdot n$ bits chosen uniformly at random. All strings form H^*.

(2) (Simulating q_i^*) $\mathcal{S}$ prepares a query list L that stores the query history. The value in L is of the form (w, a^*). When queried by a keyword set $\mathbf{w}_i$, for each keyword w_k in $\mathbf{w}_i$, $\mathcal{S}$ first scans L to see if there is a match. If not, $\mathcal{S}$ randomly chooses a distinct a_k^* which is not in L and stores the pair (w_k, a_k^*) into L. $\mathcal{S}$ gets $(a_1^*, \ldots, a_{|\mathbf{w}_i|}^*)$ according to $\mathbf{w}_i$ and sets $q_i^* = (a_1^*, \ldots, a_{|\mathbf{w}_i|}^*)$.

(3) (Simulating D_i^*) $\mathcal{S}$ sets D_i^* to a $|D_i|$-bit string chosen uniformly at random. Note that $|D_i|$ is a global parameter known by the user and the server.

We claim that no polynomial-size distinguisher $\mathcal{D}$ could distinguish the following pairs.

(1) (H and H^*) recall that H consists of N_{ent} values. Each value consists of either a string of the form $(\pi_{K_p}(w_i) \parallel c_i)$ or a random string. In any case, with all but negligible probability, the PRP key K_p is not included; therefore the pseudorandomness of π guarantees that $\pi_{K_p}(w_i)$ is indistinguishable from random. The PRF key K_m is also not included; therefore the pseudorandomness of f guarantees that the derived key k_i for each data row is indistinguishable from random, and then the underlying Matrix-AC is CPA-secure, which means that c_i is indistinguishable from random. H^* contains N_{ent} random values. Therefore, as discussed, H and H^* are indistinguishable.

(2) (q_i and q_i^*) recall that q_i is the evaluation of the PRP π. In any case, with all but negligible probability, the PRP key K_p is not included; therefore the pseudorandomness of π guarantees that all $\pi_{K_p}(w_i)$ in q_i are indistinguishable from random, and q_i^* is a random string of the same length of q_i.

(3) (D_i and D_i^*) recall that D_i is encrypted by a CPA and CCA secure encryption scheme. Since the encryption key K_d is not known by the adversary, the security of the encryption scheme guarantees that D_i and D_i^* are indistinguishable. □

6. Comparison, Application, and Performance Analysis

First, we compare the functionalities and performance of our work with previous works. Then, as a significant example, we discuss how to combine the preview scheme with symmetric searchable encryption to improve the user experience. We also discuss the performance of the preview scheme in the concrete application example.

6.1. Scheme Comparison. Let s denote the snippet length and d the document size; the comparisons of our work with other representative works are shown in Table 3.

The query-biased preview mode is widely used in general search engine, as introduced in [5]. In the scheme, the search engine dynamically scans the document line by line to find the top-ranking snippet. Therefore, the computation complexity is $O(d)$. In [7], Mithal and Tayebi proposed a static preview scheme over encrypted data based on content mask technique. In the scheme, some segments of the plaintext are extracted in advance and are masked with noise in such a way that the so called "masked preview content" could be sent to the user as a preview when queried. The static scheme is fast and informative but does not explain why a document is matched by a query. Note that our scheme costs one extra round of communication since the score results have to be returned to the user in the first round.

6.2. Symmetric Searchable Encryption Extension. We review the generalized definition of symmetric searchable encryption (SSE) introduced in [12]. We assume that the searchable encryption scheme is in guided mode. In other words, the server will first return to the user the identifiers of the matched documents, and the user makes a final choice to select some document identifiers and sends them to the server to retrieve the selected ones.

Definition 7 (extended symmetric searchable encryption). In guided mode, a symmetric searchable encryption scheme is a collection of six polynomial-time algorithms $\mathtt{SSE} = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Token}, \mathtt{Search}, \mathtt{Retrieve}, \mathtt{Dec})$ such that we have the following.

$K \leftarrow \text{Gen}(1^k)$ is a probabilistic algorithm that takes as input a security parameter k and outputs a secret key K. It is run by the user, and the output key is kept secret by the user.

$(\gamma, C) \leftarrow \text{Enc}_K(D)$ is an algorithm that takes as input a secret key K and a document collection $D = (D_1, \ldots, D_n)$ and outputs a searchable structure γ and a sequence of encrypted documents $C = (C_1, \ldots, C_n)$.

It enables a user to query some keywords, and the server returns the matched documents. For instance, in an index-based searchable symmetric encryption scheme, γ is the secure index. It is run by the user, and (γ, C) is sent to the storage server.

$t \leftarrow \text{Token}_K(\mathbf{w})$ is a deterministic (possibly probabilistic) algorithm that takes as input a secret K and a set of some keywords $\mathbf{w} = (w_1, \ldots, w_n)$ and outputs a search token t (also named trapdoor or capacity). It is run by the user.

$I \leftarrow \text{Search}(\gamma, t)$ (guided mode) is a deterministic algorithm that takes as input the query token t and the searchable structure γ and outputs the matched document identifiers $I = (I_1, \ldots, I_m)$. It is run by the server, and the result I is sent to the user. Note that, if not in guided mode, this algorithm returns the matched documents directly. It is run by the server.

$C' \leftarrow \text{Retrieve}(C, I')$ is a deterministic algorithm that takes as input the encrypted documents and the selected document identifiers $I' \subseteq I$ and outputs the selected documents corresponding to the identifiers. It is run by the server.

$D_i \leftarrow \text{Dec}_K(C_i)$ is a deterministic algorithm that takes as input a secret key K and the returned encrypted document C_i and outputs the recovered plaintext D_i. It is run by the user.

The preview scheme is applied in SSE as follows. The user runs `SSE.Gen`, `SecQBP.Gen`, `SSE.Enc`, and `SecQBP.Setup`, respectively. The server stores the outsourced structure generated by SSE and the encrypted documents generated by SecQBP scheme. To search for some documents, the user runs `SSE.Token` and `SecQBP.Query`, respectively, and sends them to the server. The server produces the identifiers of the matched documents, runs `SecQBP.ComputeScore` for the corresponding documents one by one, and returns the document identifiers and the score results together. The user decodes the score, retrieves the preview snippets from the server, then makes the choice, and sends the selected document identifiers to the server to retrieve the interested documents.

6.3. Performance Analysis. We adopt SSE-2 introduced in [11] as an instance of a SSE scheme. Table 4 shows the time complexity and storage complexity for single SSE-2 scheme and SSE-2 plus SecQBP in detail.

Let C represent the encrypted document collection, so the total size is $|C|$ bytes. Other than the returned encrypted documents, the extrastorage cost for SSE-2 is $|C|/8$ bytes; thus the storage cost is $O(n)$. The extrastorage cost for SecQBP is H for each document. By definition, the storage cost is $O(n)$. For SSE, the server searches the matched documents and decrypts the identifier list. For SecQBP, the server searches the indices for all matched documents, returns score results for all matched documents, and finally returns the snippets. They are both in time complexity of $O(1)$. The number of rounds for SSE is two (guided mode). First, the server returns the identifiers of the matched documents and next returns the selected documents. SecQBP adds extra round for retrieving snippets from the snippet server. Moreover, for each matched document, the size of the messages for SEE is $O(1)$. SecQBP is $O(\log(d/s) + s + d/s)$, where d is the document length, and s is the snippet length, since the user will receive a score result of size d/s and a snippet of size s.

Table 4: Properties of SSE-2 + SecQBP.

Properties	SSE-2	SSE-2 + SecQBP
Adaptive adversaries	Y	Y
Number of servers	1	1
Server storage	$O(n)$	$O(n)$
Server computation	$O(1)$	$O(1)$
Number of rounds	2	3
Extracommunication	$O(1)$	$O(\log(d/s) + s + d/s)$

The detailed performance of SSE is analyzed in [42]; therefore we just analyze the performance of the SecQBP part. The content of a document d is varied in the real world. By observation in [40], the number of keywords in a document increases along with the document size which satisfies log model, and the worst case satisfies linear model (each word in the document is keyword, such as a dictionary). However, the design for security in our scheme guarantees that the encrypted indices generated from any document are indistinguishable. Therefore, the computation for the server is independent from the models (i.e., the computations for all documents are the same). To simulate the reality, we design the data generator that simulates documents using log model.

In order to demonstrate the optimization for the server, we compare our suggested Matrix-AC scheme with the simplest and, as far as we know, the fastest symmetric homomorphic encryption scheme [30] denoted by SHE and a well-known homomorphic cryptosystem [29] denoted by Paillier cryptosystem. We consider that 100 users submit queries simultaneously. Each query contains 5 keywords, and the score computation is over 100 matched documents (SSE generates the identifiers of the matched documents). The size of each document increases from 50 KB to 1 MB (the sizes for all stored documents are the same), and the computation cost is described by millisecond.

The algorithms are coded in C++ programming language and the server is a Pentium Dual-Core E5300 PC with 2.6 GHz CPU. The result is shown in Figure 2. It demonstrates that:the following. (1) The scheme is secure. The figure shows a linear computation cost, which means the computation is independent of the document content. In other words, the server does not see any differences for all documents while performing the search. (2) In cloud environment, computation for 100 users simultaneously on a single server becomes a burden as the size of the document increases. In other words, the number of servers run as services is determined by the size of the stored documents and the accepted queries. (3) The performance is improved as we adopt Matrix-AC to substitute the homomorphic encryption schemes. From the data, Matrix-AC is about 30% faster than

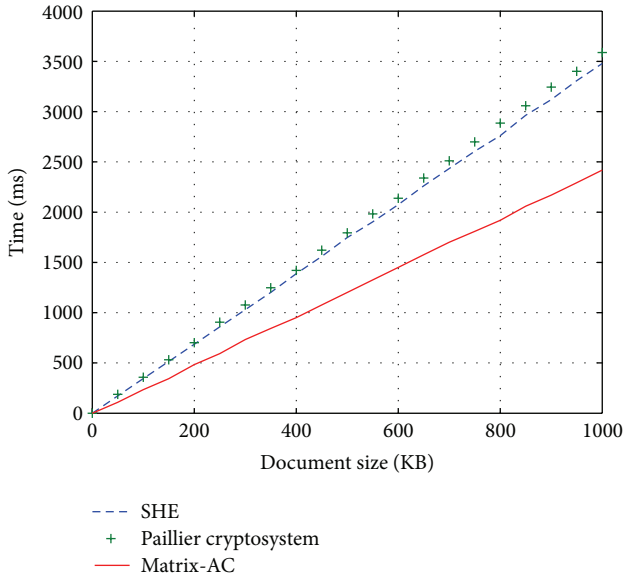


Figure 2: Time cost for computing scores (single server, 100 users).

using SHE or Paillier cryptosystem. We assume that the user does not modify the document frequently, and the main operation is just searching for some documents. Therefore, the performance improvement is significant since it could save about 30% virtual machines in the cloud.

7. Conclusions

In this paper, we propose a generalized method of securely retrieving query-biased snippet over outsourced and encrypted data, which allows the users to take a sneak preview over their encrypted data. The preview scheme has strong security and privacy guarantees with relatively low overhead, and it greatly improves the user experience.

Acknowledgments

Part of this work is supported by the Fundamental Research Funds for New Century Excellent Talents in Chinese Universities (Grant no. NCET-10-0298) and Ministry of Science and Technology of Sichuan province (no. 2012HH0003).

References

[1] A. Tombros and M. Sanderson, "Advantages of query biased summaries in information retrieval," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 2–10, ACM, 1998.

[2] D. Christopher and P. R. H. S. Manning, *Introduction To Information Retrieval*, Cambridge University Press, 2008.

[3] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell, "Summarizing text documents: sentence selection and evaluation metrics," in *Proceedings of 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 121–128, 2013.

[4] T. Sakai and K. Sparck-Jones, "Generic summaries for indexing in information retrieval," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 190–198, Association for Computing Machinery, 2001.

[5] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams, "Fast generation of result snippets in web search," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '07)*, pp. 127–134, July 2007.

[6] R. W. White, I. Ruthven, and J. M. Jose, "Finding relevant documents using top ranking sentences: an evaluation of two alternative schemes," in *Proceedings of the 25th Annual International Conference on Research and Development in Information Retrieval (ACM SIGIR '02)*, pp. 57–64, Association for Computing Machinery, 2002.

[7] A. K. Mithal and A. Tayebi, Method and System For Facilitating Search, Selection, Preview, Purchase Evaluation, Offering For Sale, Distribution, and/or Sale of Digital Content and Enhancing the Security Thereof, 2009.

[8] A. Nenkova and K. McKeown, "A survey of text summarization techniques," in *Mining Text Data*, pp. 43–76, Springer, 2012.

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

[10] E. J. Goh, "Secure indexes," Tech. Rep., 2003, IACR ePrint Cryptography Archive, http://eprint.iacr.org/2003/216.

[11] 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*, pp. 79–88, ACM Press, November 2006.

[12] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '10)*, pp. 577–594, Springer, 2010.

[13] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*, pp. 506–522, Springer, 2004.

[14] M. Abdalla, M. Bellare, D. Catalano et al., "Searchable encryption revisited: consistency properties, relation to anonymous ibe, and extensions," in *Proceedings of the 25th Annual International Cryptology Conference (CRYPTO '05)*, pp. 205–222, Springer, 2006.

[15] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III, "Public key encryption that allows pir queries," in *Proceedings of the 27th Annual International Cryptology Conference (CRYPTO '07)*, pp. 50–67, Springer, 2007.

[16] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proceedings of the 2nd International Conference (ACNS '04)*, pp. 31–45, Springer, 2004.

[17] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *IEEE International Conference on Computer Communications (INFOCOM '11)*, pp. 829–837, April 2011.

[18] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS '10)*, pp. 253–262, June 2010.

[19] A. Swaminathan, Y. Mao, G.-M. Su et al., "Confidentiality-preserving rank-ordered search," in *Proceedings of the ACM*

Workshop on Storage Security and Survivability (StorageSS '07), pp. 7–12, October 2007.

[20] C. Bosch, R. Brinkman, P. Hartel, and W. Jonker, "Conjunctive wildcard search over encrypted data," in *Proceedings of the 8th VLDB Workshop on Secure Data Management*, pp. 114–127, Springer, 2011.

[21] J. Bringer and H. Chabanne, "Embedding edit distance to allow private keyword search in cloud computing," in *Proceedings of the 8th FTRA International Conference on Secure and Trust Computing, Data Management, and Application*, pp. 105–113, Springer, 2011.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *IEEE Conference on Computer Communications (INFOCOM '10)*, March 2010.

[23] W. Cong, R. Kui, Y. Shucheng, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *IEEE Conference on Computer Communications (INFOCOM ' 12)*, pp. 451–459, IEEE, 2012.

[24] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *IEEE 28th International Conference on Data Engineering (ICDE '12)*, pp. 1156–1167.

[25] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 26, no. 1, pp. 96–99, 1983.

[26] A. K. Singh and P. Chandran, "A secure and efficient multi-authority proactive election scheme," in *Proceedings of the 3rd International Conference Information Systems Security (ICISS '07)*, pp. 208–218, 2007.

[27] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in *Annual ACM Symposium on Theory of Computing*, pp. 365–377, ACM, 1982.

[28] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[29] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '99)*, pp. 223–238, 1999.

[30] C. Castelluccia, E. Mykletun, and G. Tsudik, "Efficient aggregation of encrypted data in wireless sensor networks," in *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems-Networking and Services*, pp. 109–117, July 2005.

[31] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09)*, pp. 169–178, June 2009.

[32] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Lecture Notes in Computer Science*, pp. 24–43, 2010.

[33] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," *Journal of the ACM*, vol. 45, no. 6, pp. 965–982, 1998.

[34] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval," in *Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science*, pp. 364–373, October 1997.

[35] C. Cashin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '99)*, pp. 402–414, Springer, 1999.

[36] C. Gentry and Z. Ramzan, "Single-database private information retrieval with constant communication rate," in *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, pp. 803–815, July 2005.

[37] H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information," in *Pioneer of Information Science, Selected Works*, pp. 94–104, 1969.

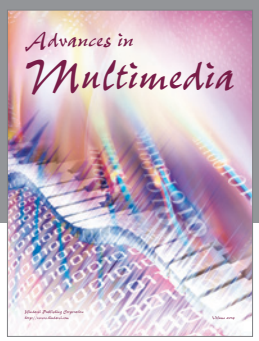
[38] C. Fox, "Lexical analysis and stop-lists," in *Information Retrieval: Data Structures and Algorithms*, 1992.

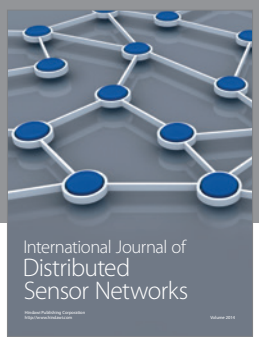
[39] R. L. Rivest and R. W. Baldwin, "The rc5, rc5-cbc, rc5-cbc-pad, and rc5-cts algorithms," in *The Internet Engineering Task Force Request For Comments*, 1996.

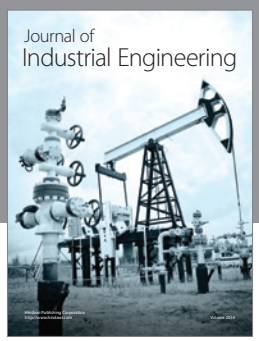
[40] M. Davies, Word frequency data, 2012, http://www.wordfrequency.info/.

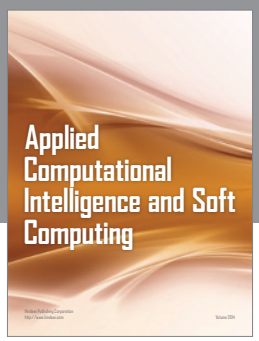
[41] M. L. Fredman, E. Szemeredi, and J. Komlos, "Storing a sparse table with o(1) worst case access time," *Journal of the ACM*, vol. 31, no. 3, pp. 538–544, 1984.

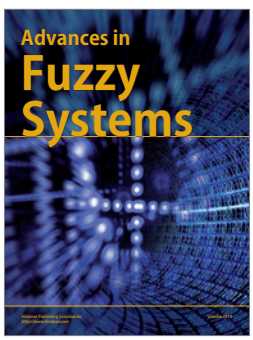
[42] S. Kamara, C. Papamanthou, and T. Roeder, "Cs2: a searchable cryptographic cloud storage system," Tech. Rep. MSR-TR-2011-58, Microsoft Research, 2011.

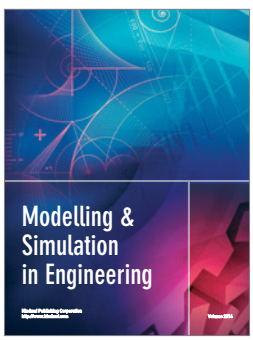
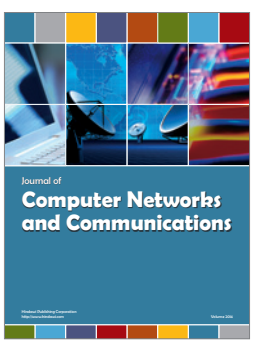
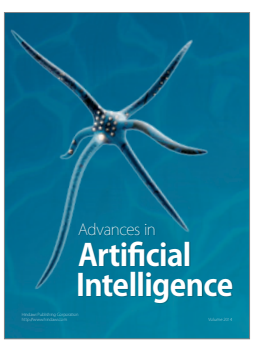
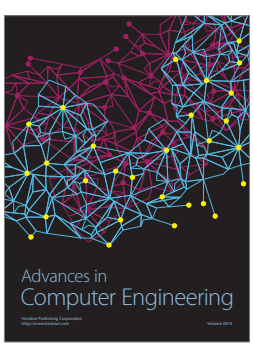
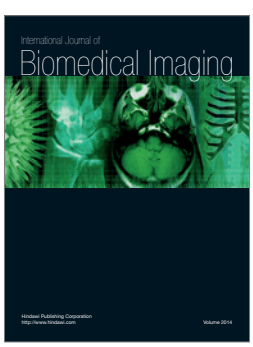
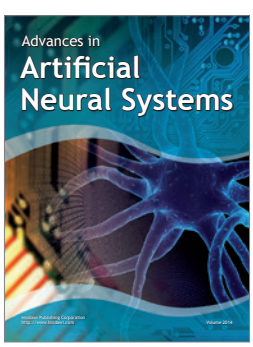
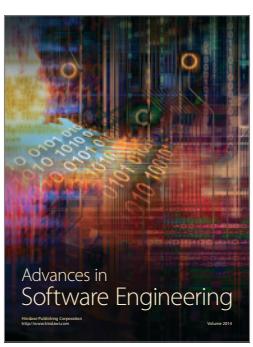
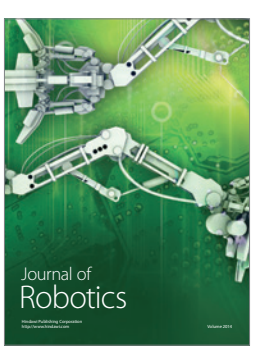
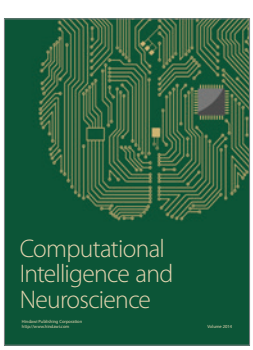










Hindawi

Submit your manuscripts at
http://www.hindawi.com