IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 22, 2018, accepted July 22, 2018, date of publication July 31, 2018, date of current version August 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2861424

Verifiable Public Key Encryption With Keyword Search Based on Homomorphic Encryption in Multi-User Setting

D. N. WU, Q. Q. GAN, AND X. M. WANG
Department of Computer Science, Jinan University, Guangzhou 510632, China
Corresponding author: X. M. Wang (twxm@jnu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61070164 and Grant 61272415, in part by the Natural Science Foundation of Guangdong Province, China, under Grant S2012010008767, in part by the Science and Technology Planning Project of Guangdong Province, China, under Grant 2013B010401015, and in part by the Zhuhai Top Discipline–Information Security

ABSTRACT Data security and privacy concerns are important issues in cloud storage. In this paper, we propose a verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting. By employing van Dijk, Gentry, Halevi, and Vaikuntanathan homomorphic encryption, the proposed scheme enables the cloud server to generate an inverted encryption index structure without using a query trapdoor, which significantly improves the efficiency of search. On the other hand, the proposed scheme presents a new authenticated data structure based on the inverted encryption index structure, and shows how to apply it to verify the correctness and completeness of search results. Moreover, the proposed scheme allows multiple users to perform encrypted keyword search over encrypted data. Finally the proposed scheme is proved secure based on the approximate-GCD problem. The experiment results demonstrate the proposed scheme has less computation overhead than the existing schemes.

INDEX TERMS Searchable encryption, homomorphic encryption, verification, multi-user setting.

I. INTRODUCTION

Cloud has been widely used not only by individuals but also by entrepreneurs because it allows people to manage their data conveniently and at low cost. But meanwhile it incurs some problems in security. The outsourced data have strong privacy and business value, while cloud service provider is semi-trusted. To protect data from leakage, data owners encrypt the data and then store them in cloud. However, the encryption greatly restricts the ability of cloud servers to handle users' requests, such as searching over encrypted data.

To resolve this problem, the notion of the public key encryption with keyword search (PEKS) was proposed by Boneh *et al.* [1]. In the PEKS scheme, a sender uploads an encrypted email to an email server along with an encrypted list of keywords. The receiver sends the desired keyword (denoted as a trapdoor) to the email server, which then tests the encrypted emails for the presence of this trapdoor. Soon afterwards, many intuitions have been proposed to improve upon this construction, and expanded it to the cloud environment (e.g., [2]–[12]). However, most of the existing PEKS schemes cannot guarantee the completeness of the search results done by the cloud server. If no verification for the completeness of search results is guaranteed, the cloud server might return incomplete search results to save computational resources. The consequence of making decisions based on incompleteness search results could be very serious or even catastrophic.

Because the files and indexes with keywords are encrypted, the cloud server can simply store the encrypted files and indexes in the order as they are received. Considering that a file has more than one keyword, data owners typically encrypt each keyword for each file and upload it. When a user submits a query trapdoor, the cloud server needs to traverse the entire indexes to find the target files, thus its time complexity is $O(n * m)$, where n represent the number of files, m represent the number of keywords for all files in the system. As a result, the efficiency of search is quite low, and it may incur the cloud server not to traverse the entire indexes and return the incomplete search results to save computational cost. Therefore, the index structure should be optimized at the cloud server to improve the efficiency of search. However, in the PEKS

schemes, the user must give the server a query trapdoor, the server can test whether the one of the encrypted indexes associated with the files is equal to the query trapdoor. In the absence of a trapdoor, the server is unable to test whether the keywords contained in the two encrypted index are consistent, so that the server cannot rearrange the encrypted index structure.

In the environment of cloud storage, the data owner is eager to share his data with multiple users. In this case, the searchable encryption works for multiple users. However, the most existing PEKS schemes are constructed in single-user setting. It is very clear that the schemes proposed for using in single-user setting cannot be directly and effectively used in multi-user setting because of the increased requirements of the latter. In PEKS schemes that are constructed in single-user setting, data owner can only shares his data with a single user and also only permits a single user to perform encrypted keyword search over encrypted data. While in multi-user setting, such as cloud storage, data owners hope to share their data with multiple users and also permit the multiple users to perform encrypted keyword search over encrypted data.

To tackle above problems, we propose a verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting. Specifically, our main contribution can be summarized as follows:

(1) We optimize the encryption index structure by employing DGHV(van Dijk, Gentry, Halevi and Vaikuntanathan) homomorphic encryption. In our scheme, the cloud server can generate an inverted encryption index structure without using a query trapdoor, which significantly improves the efficiency of search.

(2) We propose a new authenticated data structure for verifying the completeness of search results based on the inverted encryption index structure, and apply it to generate verification proof for verifying the correctness and completeness of search results.

(3) Our scheme is constructed in multi-user setting, and allows multiple users to perform encrypted keyword search over encrypted data.

(4) Our scheme is proved to be secure based on the Approximate-GCD problem.

(5) We evaluate the performance of our scheme and compare our scheme with the previous schemes in terms of time complexity and functions. The time complexity of single keyword search is reduced to $O(m)$ in our scheme. Experiment results demonstrate the efficiency of search in our scheme is higher than the existing schemes.

The following sections are described briefly as follows. We introduce some important prior works in this domain in Section 2. After that, we introduce some preliminaries used in the paper in Section 3. In the Section 4, we give the formal model of our scheme and a security model as well as the specific construction about our scheme. Then in Section 5, we prove our scheme is secure under the security model. The scheme performance is evaluated in Section 6. Finally, we conclude this paper in Section 7.

II. PRIOR WORKS

To enable users search over encrypted outsourced data through keywords without decrypting the data at first, the notion of public key encryption with keyword search (PEKS) was first put forth by Boneh *et al.* [1] and its construction makes use of the construction of identity-based encryption (IBE). Subsequently Boneh *et al.* [2] presented a more practical scheme which supported arbitrary conjunctive queries (such as comparison search, subset search, etc.). Baek *et al.* [3] proposed a PEKS scheme with a designated server to remove a secure channel. Camenisch *et al.* [4] proposed oblivious generation of the keyword search trapdoor to maintain the privacy of the keyword against a curious trapdoor generator. Cao *et al.* [5] presented ranked searches using multi-keyword over encrypted cloud data and established a variety of privacy requirements. So far, a lot of work has been done to enhance the security and the efficiency of PEKS scheme (e.g. [6]–[13]).

With the development of encryption search technology, the risk of privacy leakage in the outsourced data has been improved. However the problem of providing secure inquiry service has become another new challenge. Due to the system may occur malfunction or the cloud server might return an incomplete search results to save computational resources, the receiver may receive some incorrect and incomplete search results, so that the receiver may make a wrong decision based on the incorrect and incomplete search results. It could be very serious or even catastrophic. Therefore, we should consider the verifiability of search result. So far, there are many schemes to address this issue, such as the schemes [14]–[22]. However, most of the existing schemes mainly focus on the integrity verification of plaintext data, while there are few verifiable search schemes over encrypted data. When the schemes are migrated to the ciphertext, the schemes are no longer applicable since the ciphertext contains the data owner's private key and the random number. To the best of our knowledge, there are not many verifiable search schemes over encrypted data, and these schemes are devised to verify the correction of search results in single-user setting, and do not consider the completeness verification of search result in multi-user setting. Chai and Gong [23] gave the first verifiable keyword search in symmetric setting. Schemes [24], [25] presented the fine-grained keyword search schemes through utilizing attribute-based encryption. Sun *et al.* [26] presented a search result verification scheme in the multi-keyword text search scenario by turning the proposed secure index tree into an authenticated one. Guo *et al.* [27] put forward a multi-phrase ranked search scheme over encrypted data, which can verify the ranked results and support dynamic update operations. Specially, the aforementioned works are devised in single-user setting.

III. PRELIMINARIES

A. APPROXIMATE-GCD PROBLEM

Let λ be a security parameter, $\rho = \lambda$, $\eta = O(\lambda^2)$, and $\gamma = O(\lambda^5)$. The (ρ, η, γ)-Approximate-GCD problem is defined as follows [28]:

- ***The*** (ρ, η, γ)***-Approximate-GCD problem***: Given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chosen η-bit odd integer p, output p, where

$$\mathcal{D}_{\gamma,\rho}(p) = \{choose\ q \xleftarrow{R} \mathbb{Z} \cap [0, 2^{\gamma}/p), \\ r \xleftarrow{R} \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) : \\ output\ x = pq + r\}. \quad (1)$$

B. DGHV HOMOMORPHIC ENCRYPTION

The DGHV homomorphic encryption algorithm is described as follows [28]:

- ***ParamGen***(λ): Input λ as a security parameter, the parameter generation algorithm sets $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = O(\lambda^2)$, $\gamma = O(\lambda^5)$, $\tau = \gamma + \lambda$, and outputs public parameters as $params = \langle \rho, \rho', \eta, \gamma, \tau \rangle$.
- ***KeyGen***(λ): The key generation algorithm first chooses a random odd η-bit integer p where $p \in [2^{\eta-1}, 2^{\eta}]$. Then it draws $(\tau + 1)$ samples $x_0, \ldots, x_\tau$ from $\mathcal{D}_{\gamma,\rho}(p)$, relabels so that x_0 is the largest and restarts unless x_0 is odd. Finally the secret key is $sk = p$ and the public key is $pk = \langle x_0, x_1, \ldots, x_\tau \rangle$.
- ***Encrypt***(pk, m): To encrypt a bit $m \in \{0, 1\}$, the encryption algorithm chooses a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and sets the ciphertext as $c = [m + 2r + 2\sum_{i \in S} x_i]_{x_0}$.
- ***Evaluate***$(pk, C, c_1, \ldots, c_t)$: For a (binary) circuit C_ε with t inputs, and t ciphertexts c_i, the evaluate algorithm applies the (integer) addition and multiplication gates of C_ε to the ciphertexts, performs all the operations over the integers, and outputs the resulting integer.
- ***Decrypt***(sk, c): Input the secret key sk and ciphertext c, the decryption algorithm outputs $((c \mod p) \mod 2)$.

C. HOMOMORPHIC HASH FUNCTION

Let G be a multiplicative cyclic group of order p, and $(g_1, g_2, \ldots, g_n)$ be generators. For a vector $b = (b_1, b_2, \ldots b_n)$, its homomorphic hash function is defined as $H(b) = \prod_{i=1}^{n} g_i^{b_i}$. Then $H(b)$ satisfies the following properties [29]:

Homomorphic: For any two vectors b_1, b_2, and random integers r_1, r_2, then $H(r_1 b_1 + r_2 b_2) = H(b_1)^{r_1} H(b_2)^{r_2}$.

Collision Free: For any polynomial time algorithm, it is hard to find b_1, b_2, b_3, r_1 and r_2 $(b_3 \neq r_1 b_1 + r_2 b_2)$, which satisfies $H(b_3) = H(b_1)^{r_1} H(b_2)^{r_2}$.

IV. A VERIFIABLE PUBLIC KEY ENCRYPTION SCHEME WITH KEYWORD SEARCH IN MULTI-USER SETTING

A. SYSTEM MODEL

The system model is depicted in Figure 1. There are four major entities in this system: key-generation center, data owners, cloud servers, and data users.

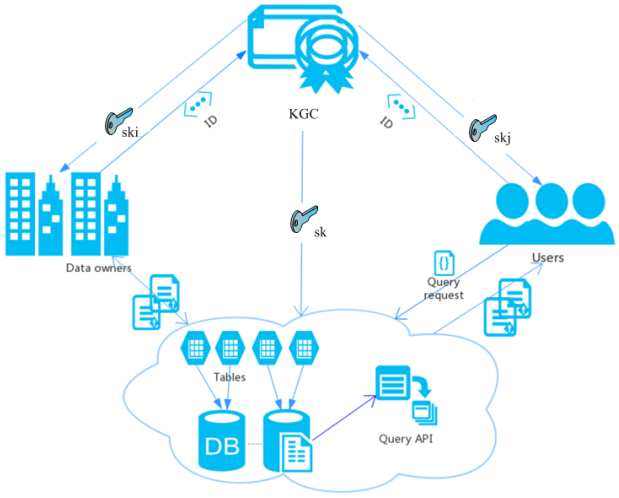

FIGURE 1. System model.

- ***Key-generation center***. The key-generation center (KGC) refers to a fully-trusted center who is responsible for generating public/secret keys and then sending them to data owners, cloud servers and users.
- ***Data owners***. The data owners refer to a special type of users who create the private/confidential data and then outsource them to cloud servers in an encrypted form so that it can be shared with authorized users.
- ***Cloud servers***. The cloud servers have a huge storage space and a strong computing power to handle and maintain data owner's data. The cloud servers are responsible for producing search results over the encrypted data according to the users' search requirements and then sending the search results to the users.
- ***Users***. Users generally refer to those who have registered to search for encrypted keywords in the encrypted data. Note that every user in this paper has an identity belonging to the public domain.

B. THREAT MODEL

We define that KGC and authorized users are honest but regard the cloud server as "honest-but-curious," which is adopted in related schemes on secure cloud data search [5], [26]. "honest-but-curious" means that the cloud server executes the scheme honestly, but it also tries to analyze the receiving data to obtain extra information with curiosity. According to the available information that the cloud server can gain, we consider the threat model as follows:

- ***Known Ciphertext Model***. In this model, the cloud server can only access the ciphertexts, which is intended to protect the keywords against the cloud server. To be specific, cloud server achieve keyword search on ciphertexts for data users. As for our scheme, the server uses the ciphertexts to build a *Z-index* structure based on the inverted encryption index structure, which can be

employed for keyword search and verification. However, the server do not know the user's secret key so that the server cannot obtain the keywords from the ciphertexts.

C. DEFINITION

When a data owner wants to outsource his data to cloud server, he will first encrypt his file ($E(file)$) using a standard proxy re-encryption public key algorithm. After appending to the ciphertext $PEKS(w_i, pk)$ of each keyword, the data owner will send the following message to cloud server:

$$E(file)||PEKS(w_1, pk)||\ldots||PEKS(w_m, pk),$$

Where $PEKS$ is an encrypted algorithm with properties discussed below. This paper focuses on addressing how the cloud server searches all files containing a keyword-search query $w = (w_1, w_2, \ldots, w_t)$, and user verifies the correctness and completeness of the search result. We omit the discussion of proxy re-encryption.

Definition 1: A verifiable public key encryption scheme with keyword search in multi-user setting consists of the following algorithms:

- ***Setup***(1^λ): After input a secure parameter λ, ***Setup***(1^λ) algorithm outputs a pair of secret key sk and public key pk.
- ***KeyGen***$(1^\lambda, id)$: The ***KeyGen*** algorithm takes as input a user identity id and 1^λ, then generates a secret key sk_{id} for the user.
- ***PEKS***(w_i, pk): The ***PEKS*** algorithm produces a searchable ciphertext CT_i with keyword w_i by a pubic key pk.
- ***Test***(CT_i, CT_j): After input two searchable ciphertexts CT_i and CT_j, the ***Test*** algorithm verifies whether the keywords contained in the two ciphertexts are the same. If they are the same, output 1; Otherwise, output 0.
- ***Z-Index***$(CT_i(i = 1, 2, \ldots, m))$: After input a set of searchable ciphertext $CT_i(i = 1, 2, \ldots, m)$, the ***Z-Index*** algorithm outputs a Z-$Index$ structure.
- ***Query***$(w_1, w_2, \ldots, w_t, sk_{id})$: Given a keyword-search query $w = (w_1, w_2, \ldots, w_t)$, the ***Query*** algorithm calls the ***PEKS*** algorithm to generate a ciphertext for each keyword using the secret keys (sk_{id}), and outputs the ciphertext $CT_i(i = 1, 2, \ldots, t)$.
- ***RPGen***$(CT_i(i = 1, 2, \ldots, t))$: Given the ciphertexts $CT_i(i = 1, 2, \ldots, t)$ from a user, the server returns a set of encrypted files Rf, where each encrypted file $E(file_i)) \in Rf$ contains all keywords from $w = (w_1, w_2, \ldots, w_t)$. Afterwards, the server computes a $proof$ so that a user can verify that all encrypted files included in Rf contain $(w_1, w_2, \ldots, w_t)$ and ensure that no encrypted files that satisfies query keywords $(w_1, w_2, \ldots, w_t)$ is omitted from Rf.
- ***Verify***$(Rf, proof)$: The ***Verify*** algorithm takes as input the Rf and $proof$, and checks the correction and completeness of the search results. If the results are correct and complete, output 1; Otherwise output 0.

D. SCHEME CONSTRUCTION

Let ***ParamGen***$'$ and ***KeyGen***$'$ be the parameter generation algorithm and the key generation algorithm from the DGHV homomorphic encryption [28]. A verifiable public key encryption scheme with keyword search in multi-user setting is constructed as follows.

- ***Setup***(1^λ). The key-generation center (KGC) firstly runs ***ParamGen***$'$ to obtain the public parameters $params' = \langle \rho, \rho', \eta, \gamma, \tau \rangle$, and runs ***KeyGen***$'$ to obtain $sk' = p, pk' = \langle x_0, x_1, \ldots, x_\tau \rangle$. Then the KGC picks a homomorphic hash H and a collision-resistant hash $H_1 : \{0,1\}^* \rightarrow \{0,1\}^Q (Q \ll \eta)$. Finally the KGC sets the public key $pk = (params', pk', H, H_1)$ and sends the secret key $sk = sk' = p$ to server through a secure channel.
- ***KeyGen***$(1^\lambda, id)$. When a user submit his identity id for registration, KGC will choose a random number k_i and computer $q_{id} = H_1(id \oplus k_i)$ and sends $sk_{id} = q_{id}$ to the user through a secure channel.
- ***PEKS***(w_i, pk). Given a keyword w_i of the W-bit file, satisfying $W \ll \eta$, a user chooses a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r_i, and computes a searchable ciphertexts of w_i by using the public key pk and the user's secret key q_{id} as follows.

$$\begin{aligned} C_{i1} &= [w_i + r_i q_{id} + r_i q_{id} \sum_{i \in S} x_i]_{x_0}, \\ C_{i2} &= H(r_i q_{id}). \end{aligned} \tag{2}$$

  Thus, the searchable ciphertexts of the keyword w_i is $CT_i = (C_{i1}, C_{i2})$.
- ***Test***(CT_i, CT_j). After receiving the two ciphertexs (CT_i, CT_j), the server verifies whether the two ciphertexts contain the same keyword by the following way.

$$\frac{H(C_{i1} \mod p) \times C_{j2}}{H(C_{j1} \mod p) \times C_{i2}} = 1. \tag{3}$$

  If $w_i = w_j$, it outputs 1; Otherwise it outputs 0.

PEKS(w_1)	E(file$_1$)	E(file$_2$)			
PEKS(w_2)	E(file$_3$)	E(file$_4$)	E(file$_5$)		
PEKS(w_3)	E(file$_1$)	E(file$_2$)	E(file$_3$)	E(file$_4$)	E(file$_5$)
...	...	...	...	...	...
PEKS(w_{m-2})	E(file$_6$)	E(file$_7$)	E(file$_{10}$)		
PEKS(w_{m-1})	E(file$_2$)	E(file$_3$)	E(file$_5$)	E(file$_8$)	E(file$_9$)
PEKS(w_m)	E(file$_1$)	E(file$_2$)	E(file$_{10}$)		

FIGURE 2. An inverted encryption index structure.

- ***Z-IndexBuild*** $(CT_i(i = 1, 2, \ldots, m))$. Given the ciphertexts $CT_i(i = 1, 2, \ldots, m)$, the server calls ***Test***(CT_i, CT_j) algorithm to determine whether the keywords in the two ciphertexs (CT_i, CT_j) are consistent, where $PEKS(w_i) = CT_i$, so that the server can put

FIGURE 3. A *Z-Index* structure.

the ciphertexts with the same keyword in one line, and build an inverted encryption index structure (see Figure 2). Based on the inverted encryption index structure, an authenticated data structure is constructed and initially empty, shown as Figure 3. The row of the structure denotes the encrypted keyword $PEKS(w_i)$ (where $i \in [1, m]$, $PEKS(w_i) = CT_i$). The column of the structure denotes the encrypted file $E(file_i)(i \in [1, n])$. Each row has a vector $\mathbf{v}_{w_i} = [v_{i1}, v_{i2}, \ldots, v_{in}]$ $(i \in [1, m])$ and a verification proof $preproof(\mathbf{v}_{w_i})$. If the $E(file_j)(j \in [1, n])$ contains the keyword w_i, then set $v_{ij} = 1(j \in [1, n])$. Otherwise, set $v_{ij} = 0$ (see Figure 3). Namely, each "1" in the $\mathbf{v}_{w_i}$ is mapped to a encrypted file that contains the keyword w_i. And for a vector $\mathbf{v}_{w_i}$, then $preproof(\mathbf{v}_{w_i}) = H(value(\mathbf{v}_{w_i}))$, where $value(x)$ is a function that converts x to a decimal number. Finally the authenticated data structure is formed, which is called *Z-IndexBuild*, and published as shown in Figure 3.

- ***Query***$(w_1, w_2, \ldots, w_t, sk_{id})$: Given a keyword-search query $w = (w_1, w_2, \ldots, w_t)$, a user executes the ***PEKS*** algorithm to generate the ciphertexts $CT_i(i = 1, 2, .., t)$ for each search keyword $w_i(i = 1, 2, \ldots, t)$ by the user's secret key sk_{id}, and sends the ciphertexts $CT_i(i = 1, 2, .., t)$ to the server.
- ***RPGen***$(CT_i(i = 1, 2, \ldots, t))$: After receiving the ciphertexts $CT_i(i = 1, 2, \ldots, t)$, the server tests which $PEKS(w_j)$ in the *Z-Index* structure is equal to CT_i by running the ***Test***(CT_i, CT_j) algorithm, where $PEKS(w_j) = CT_j$. Once the server finds t correspondent $PEKS(w_i)$ in the *Z-Index* structure, the server obtains the subset vectors $\mathbf{v}_{w_i}(i = 1, 2, \ldots, t)$, and uses the bit operation & to compute completeness witnesses

$$\mathbf{v}_{result} = \mathbf{v}_{w_1} \& \mathbf{v}_{w_2} \& \ldots \& \mathbf{v}_{w_t}. \tag{4}$$

According to the $\mathbf{v}_{result}$, the server returns a set of encrypted files Rf, where each encrypted file $E(file_i)) \in Rf$ is mapped to 1 in the $\mathbf{v}_{result}$. Namely, each encrypted file $E(file_i)) \in Rf$ contains all keywords $w = (w_1, w_2, \ldots, w_t)$. If $Rf = \{E(file_1), E(file_2), \ldots, E(file_l)\}(l \leq t)$, then the server returns a verification $proof = value(\mathbf{v}_{w_1})||value(\mathbf{v}_{w_2})|| \ldots ||value(\mathbf{v}_{w_l})$ to the user.

- ***Verify***$(Rf, proof)$. After receiving Rf and $proof$, the user finishes the following verification steps.

1) Check whether each subset vector $value(\mathbf{v}_{w_i})$ is correct as follows.

$$H(value(\mathbf{v}_{w_i})) = preproof(w_i)(i = 1, 2, \ldots l). \tag{5}$$

If the equation holds, move to next step. If not, abort.

2) Use $Rf = E(file_i)(i = 1, 2, \ldots, l)$ to build a vector $\mathbf{v}'$. If $i \in [1, l]$, $v_i = 1$; otherwise, $v_i = 0$.

3) Check the completeness of search result:

$$\mathbf{v}' = binary(value(\mathbf{v}_{w_1})) \,\&\, \ldots \,\&\, binary(value(\mathbf{v}_{w_l})). \tag{6}$$

If the equation holds, output 1. If not, output 0. Where given a decimal number, $binary(x)$ is a function to output its binary format.

Example: A user sends the query ciphertexts (CT_1, CT_2) of corresponding keywords (w_1, w_2) to the server, the server finds the corresponding $PEKS(w_1)$, $PEKS(w_2)$ in the *Z-IndexBuild* by running the ***Test***(CT_i, CT_j) algorithm. Thus the server obtains the corresponding vectors $\mathbf{v}_{w_1} = 101 \ldots \ldots 011$ and $\mathbf{v}_{w_2} = 110 \ldots \ldots 111$, and uses the bit operation & to compute $\mathbf{v}_{result} = \mathbf{v}_{w_1} \& \mathbf{v}_{w_2} = 100 \ldots ..011$. In the $\mathbf{v}_{result}$, the first bit is 1, the last two bits are 1, so the corresponding $Rf = \{E(file_1), E(file_{n-1}), E(file_n)\}$ and the verification $proof = value(\mathbf{v}_{w_1})||value(\mathbf{v}_{w_2})$ are returned to the user.

After receiving $(Rf, poof)$, the user can obtain $value(\mathbf{v}_{w_1})$ and $value(\mathbf{v}_{w_2})$ from $proof$, and $preproof(w_1) = H(value(101 \ldots \ldots 011))$ and $preproof(w_2) = H(value(110 \ldots \ldots 111))$ from the published *Z-Index* structure (see Figure 3). Thus the user can check

$$H(value(\mathbf{v}_{w_1})) = prepoof(w_1),$$
$$H(value(\mathbf{v}_{w_2})) = prepoof(w_2).$$

If the above equations hold, then the user builds $\mathbf{v}' = 100 \ldots \ldots 011$ according Rf, and checks

$$\mathbf{v}' = binary(value(\mathbf{v}_{w_1})) \,\&\, binary(value(\mathbf{v}_{w_2})).$$

If the above equation holds, output 1. If not, output 0.

E. CORRECTNESS AND COMPLETENESS

According to the definition in [30]: For sets $S_1, S_2, S_3, \ldots, S_l$, if $I = S_1 \cap S_2 \cap S_3 \cap \ldots \cap S_l$ is correct and complete, if and only if the following two conditions hold.

- ***subset condition***: $I \subseteq S_1 \wedge I \subseteq S_2 \wedge \ldots \wedge I \subseteq S_l$;
- ***completeness***: $(S_1 - I) \cap (S_2 - I) \cap \ldots \cap (S_l - I) = \varnothing$.

Thus, the two conditions in our scheme can be described as follows:

- ***subset condition***: $\mathbf{v}_{result} \subseteq \mathbf{v}_{w_1} \wedge \ldots \wedge \mathbf{v}_{result} \subseteq \mathbf{v}_{w_t}$;
- ***completeness***: $(\mathbf{v}_{w_1} - \mathbf{v}_{result}) \cap \ldots \cap (\mathbf{v}_{w_1} - \mathbf{v}_{result}) = \varnothing$.

Subset Condition: In our scheme, the verification of the equation (3) ensures the satisfaction of subset condition. Because if the equation(3) is equal to 1, then the two ciphertexts CT_i and CT_j contain the same keyword, so that the number of "1" in the subset vector $\mathbf{v}_{w_i}$ is the number of encrypted file with the same keyword w_i. Meanwhile the

number of "1" in $\mathbf{v}_{result}$ is the number of encrypted file with all keywords from $w = (w_1, w_2, \ldots, w_t)$.

Specially, assuming CT_i and CT_j are the ciphertexts for the keywords w_i and w_j respectively. If $w_i = w_j$, then

$$\begin{aligned}\frac{H(C_{i1} \mod p) \times C_{j2}}{H(C_{j1} \mod p) \times C_{i2}} &= \frac{H(w_i + r_i q_i) \times H(q_j)^{r_j}}{H(w_j + r_j q_j) \times H(q_i)^{r_i}} \\ &= \frac{H(w_i) \times H(q_i)^{r_i} \times H(q_j)^{r_j}}{H(w_j) \times H(q_j)^{r_j} \times H(q_i)^{r_i}} \\ &= \frac{H(w_i)}{H(w_j)} \end{aligned} \tag{7}$$

Obviously, the equation (3) is true.

Completeness: In our scheme, given a query keyword $w = (w_1, w_2, \ldots, w_t)$ from a user, the server returns a set of encrypted files according to the $\mathbf{v}_{result}$. Namely the server returns all encrypted files whose corresponding positions in the $\mathbf{v}_{result}$ are equal to 1, and does not return the encrypted files whose corresponding positions in the $\mathbf{v}_{result}$ are equal to 0. If the $\mathbf{v}_{result}$ is correct, then the completeness is satisfied. The correction of the $\mathbf{v}_{result}$ is guaranteed by the equations (3) and (4). Meanwhile, the verification of the equations (5) and (6) ensures that a user can verify the completeness of search results. Because if the equation (5) holds, it shows that the returned $value(\mathbf{v}_{w_i})$ is correct since the $preproof(w_i)$ is published in the *Z-IndexBuild*. Meanwhile according to the $Rf = E(file_i)(i = 1, 2, \ldots, l)$ returned by the server, the user can reconstruct $\mathbf{v}^{'}$. If the equation (6) holds, it shows that the reconstructed $\mathbf{v}^{'}$ is correct, which is equal to the $\mathbf{v}_{result}$. Therefore, our scheme can verify the completeness of search results.

V. SECURITY ANALYSIS

As for security aspect, we reduce our scheme from Section 4 to the hardness of the Approximate-GCD problem. In other words, randomly chosen a set of integers $x_0, x_1, \ldots, x_\tau$, which are all close to multiples of a large integer p, try to find this "common near divisor" p. In order to get a reliable oracle for the least-significant bit through the promised adversary, we describe a random-self-reduction and accuracy-amplification step as in [28]. Therefore, a Binary-GCD algorithm can employ the obtained reliable oracle to find p.

Considering the technical details, our random self-reduction implies a loss in parameters. In particular, the obvious advantage in guessing the encrypted bit in a random "high ρ'-bits noise ciphertext" can be transformed into the ability to predict reliably the parity bit of the quotient in an arbitrary "low ρ-bits noise integer". By adding extra noise we can "wipe out the traces" of the non-random noise in the arbitrary input integer. That means the security of our scheme in "high-noise" can be reduced to the hardness of Approximate-GCD problem in "low-noise," where the difference between "high noise" and "low noise" is quite small.

Theorem 1: Fix the parameters $(\rho, \rho', \eta, \gamma, \tau)$ as in the proposed scheme from Section 4 (all polynomial in the security parameter λ). Any attack $\mathcal{A}$ with advantage ε on the proposed scheme can be converted into an algorithm $\mathcal{B}$ for solving (ρ, η, γ)-Approximate-GCD with success probability at least $\varepsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, and in λ and $1/\varepsilon$.

Proof: Now we use the same way as [28] to show how the challenger $\mathcal{B}$ to recover p with the success probability. We use $q_p(z)$ and $r_p(z)$ to denote the quotient and remainder of z with respect to p, hence $z = q_p(z)p + r_p(z)$.

- ***Step 1***. First the challenger $\mathcal{B}$ draws $(\tau + 1)$ samples $x_0, \ldots, x_\tau$ from $\mathcal{D}_{\gamma,\rho}(p)$. It relabels so that x_0 is the largest. It restarts unless x_0 is odd. $\mathcal{B}$ outputs a public key $pk =< x_0, x_1, \ldots, x_\tau >$ to the adversary $\mathcal{A}$.
- ***Step 2***. $\mathcal{B}$ produces a sequence of integers, and attempts to recover p by utilizing $\mathcal{A}$ to learn the least significant bit of the quotients of these integers with respect to p. For this, $\mathcal{B}$ uses the following ***Subroutine Learn-LSB Algorithm***:

Algorithm 1 Subroutine Learn-LSB(z, pk)

Input: $z \in (0, 2^\lambda)$ with $|r_p(z)| < 2^\rho$ and $pk =< x_0, x_1, \ldots, x_\tau >$
Output: The least-significant-bit of $q_p(z)$

1. For j = 1 to ploy$(\lambda)/\epsilon$ do:
2. choose noise $r_j \xleftarrow{R} (-2^{\rho'}, 2^{\rho'})$, a bit $w_j \xleftarrow{R} \{0, 1\}$ and a random subset $S_j \subseteq_R \{1, 2, \ldots, \tau\}$
3. set $CT_j \leftarrow [z + w_j + r_j + r_j \Sigma_{k \in S_j} x_k]_{x_0}$
4. $a_j \leftarrow \mathcal{A}(pk, CT_j)$
5. set $b_j \leftarrow a_j \oplus parity(z) \oplus w_j$
6. Output the majority vote among the b_j's

- ***Step 3***. Once we turned $\mathcal{A}$ into an oracle for the least-significant bit of $q_p(z)$, recovering p is rather straightforward. Perhaps the simplest way of doing it is using the ***Binary GCD Algorithm***: Given any two integers $z_1 = q_p(z_1) \cdot p + r_p(z_1)$ and $z_2 = q_p(z_2) \cdot p + r_p(z_2)$, $\mathcal{B}$ uses the following ***Binary GCD Algorithm***:

Algorithm 2 Binary GCD(z_1, z_2)

Input: $z_1 = q_p(z_1) \cdot p + r_p(z_1)$ and $z_2 = q_p(z_2) \cdot p + r_p(z_2)$
Output: The odd part of GCD$(q_p(z_1), q_p(z_2))$

1. If $z_2 > z_1$ then $z_1 \leftrightarrow z_2$
2. call Learn-LSB( ) output $b_1 = [q_p(z_1)]_2$ and $b_2 = [q_p(z_2)]_2$
3. If both $q_p(z_1)$ and $q_p(z_2)$ are odd then replace z_1 by $(z_1 - z_2)$ and set $b_1 \leftarrow 0$
4. For each z_i with $b_i = 0$, replace z_i by $z_i \leftarrow (z_i - parity(z_i))/2$

- ***Step 4***. To recover p, $\mathcal{B}$ draws a pair of elements $z_1^*, z_2^* \xleftarrow{R} \mathcal{D}_{\gamma,\rho}(p)$. According to [28], with probability at least $\Pr[\mathrm{GCD}(q_p(z_1^*), q_p(z_2^*)) = 1] = \pi^2/6 \approx 0.6$. That is, $\mathcal{B}$ will get a element $\widetilde{z} = 1 \cdot p + r$. Finally, $\mathcal{B}$ recovers $p = \lfloor z_1^*/q_p(z_1^*) \rceil$.

This is contradictory to the condition that the approximate-gcd problem is difficult.

Although the server has the secret key p, it can learn nothing else about the keyword. For the ciphertexts $\{C_{i1} = [w_i + r_i q_{id} + r_i q_{id} \sum_{i \in S} x_i]_{x_0}, C_{i2} = H(r_i q_{id})\}$, since the server does not know the secret key $sk_{id} = q_{id}$, it cannot obtain any information about the keyword w_i. Furthermore, r_i is uniformly random and independent over Z_p^* from server's view, which can ensure the ciphertexts' indistinguishability and enhance the keywords' privacy one step further. Therefore, the server learns nothing more about the keyword w_i.

VI. PERFORMANCE ANALYSIS

This section mainly evaluates the performance of our scheme inculding the functions, computational cost, index-based searchable encryption schemes comparison and experiment results. Suppose $|DO|$ represent the number of the authorized data owners, n represent the number of data files, and m represent the number of search keywords, t represent the number of queried keywords, d denote the number of search results.

Firstly, we show the functions and computational complexity of our scheme through comparing with other analogous schemes [31]–[33] in Table 1 and Table 2.

TABLE 1. Functionality comparison.

Schemes	Multiple Keywords	Verifiable	OIB	Data Update
Re-dPEKS [31]	×	×	×	✓
Re-dtPECK [32]	✓	×	×	✓
VMKDO16 [33]	✓	✓	×	✓
Our scheme	✓	✓	✓	✓

Notation. OIB: Outsourced Index Building

TABLE 2. Computational cost comparison.

Schemes	$KeyGen$	$PEKS$	$Test$	$Verify$
Re-dPEKS [31]	$(2\|DO\|+2)E$	$(2m+1)E+mP$	$(t+1)E+H+tP$	\
Re-dtPECK [32]	$(\|DO\|+2)E$	$(m+5)E+2P$	$(t+4)E+(t+2)P$	\
VMKDO16 [33]	$(\|DO\|+1)E$	$\varsigma E+nH+3P$	$2P+3E$	$(d+1)E+dH+2P$
Our scheme	$(\|DO\|)H^{\ddagger}$	$m(H+2A+2M)$	$m(D+2H+2M)t$	tL

Notation. E: Exponentiation; H: Hash operation; P: Pairing operation; A: Addition; M: Multiplication; D: Division; L: Logic operation. $\varsigma = m + 5 + 2n$. ‡: cost of $\oplus$ operation which is usually regarded as negligible.

Obviously, our scheme enriches the search functionalities over encrypted data. As illustrated in Table 1, it can achieve aforementioned functionalities simultaneously, while the other three cannot. Our scheme allows the cloud server to generate an inverted encryption index structure without a query trapdoor, which significantly improves the efficiency of searching.

From Table 2, we can see that ***KeyGen***, ***PEKS***, ***Test*** and ***Verify*** algorithms in our scheme have lower computational overhead than those of other schemes. Because our scheme adopts a homomorphic encryption and some simple addition, multiplication and division operations, while other three schemes used the cryptography technology, such as bilinear pairing operations.

TABLE 3. Index-based searchable encryption schemes comparison.

Schemes	Index Structure	Index Size	Search Complexity	Data Update Structure
Z-IDX [34]	File-Keyword	$O(n)$	$O(n \cdot m)$	Bloom Filter
PPSED [35]	File-Keyword	$O(n \cdot m)$	$O(n \cdot m)$	Masked Index
MKPSE [9]	Keyword-File	$O(n \cdot m)$	$O(m^2)$	Index $\tilde{I}$
DEPKS [36]	Keyword-File	$O(m^2)$	$O(m \log m)$	Index C
Our scheme	Keyword-File	$O(n \cdot m)$	$O(m)$	Z-Index

Notation. Data Update Structure: updating the structure when data update operation happens.

To demonstrate the search efficiency with index structure, we make comparisons with several related searchable encryption schemes [9], [34]–[36] and the results are displayed in Table 3. In general, the index structure can be divided into two types: file-keyword and keyword-file, the latter one also named the inverted index. From Table 3, we can see the search complexity in schemes [9], [36] and our scheme is smaller than schemes [34], [35] due to the index structure. In our scheme, we introduce an index structure named *Z-Index*, which can support keyword search and results verification to ensure correctness and completeness of search results. By comparison, it shows that our scheme has better efficiency when weighing all the factors in Table 3.

To access the actual performance of our scheme in computational time, we perform the comparison experiments with VMKDO16 [33] over a real-world dataset named Enron email dataset. The experiments are implemented on windows 10 with Inter Core i5 and Processor 2.6 GHz. We set $E(F_q) : y^2 = x^3 + x$ and G_1 is a subgroup of $E(F_q)$, where q is a large prime number. The group order of G_1 is 160-bit, and the base field is 512-bit. The experiment results are shown in Figure 4, Figure 5 and Figure 6.

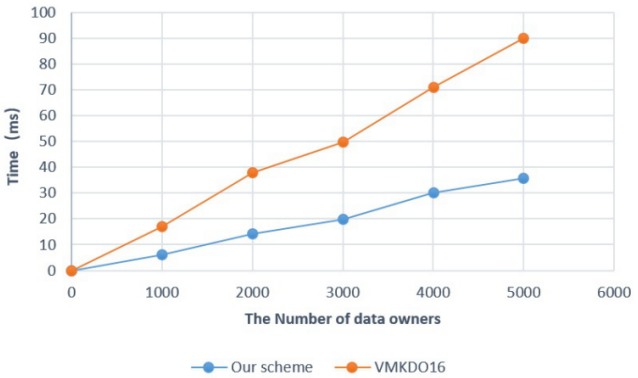

FIGURE 4. KeyGen algorithm comparison.

From Figure 4, we can see that the computational time of ***KeyGen*** algorithm in two schemes almost linearly increases with the number of *DO*s (Here we set $|DO| \in [1, 5000]$). We also see that our scheme needs less computational time than the scheme VMKDO16 since our scheme only needs a

Hash operation to generate the private key, while the scheme VMKDO16 needs to select 2 random numbers for each data owner and do 2 exponentiation operations to obtain a pair of public-private key of the data owner.

In the ***PEKS*** phase, the scheme VMKDO16 firstly needs to encrypt file set F through the traditional public key encryption algorithm, and generate the signatures for the encrypted file set(Here we set $m \in [1, 1000]$). The computational cost of generating the signature for each encrypted file block is an exponential operation. Following the index is built for file set according to the given keyword set. The process mainly involves multiple bilinear pairing and exponential operations. However, our scheme only needs 2 multiplications, 2 additions and a Hash operation to finish ***PEKS*** algorithm. Therefore, the computational burden of ***PEKS*** algorithm in the scheme VMKDO16 is much heavier than our schemes. The experiment result showed in Figure 5 is consistent with our analysis. Actually, the ***PEKS*** algorithm is affected by the keyword number m, and its computational burden becomes heavier with increasing m. However, the performance of our scheme is almost unaffected because the computation burden of addition and multiplication is negligible.

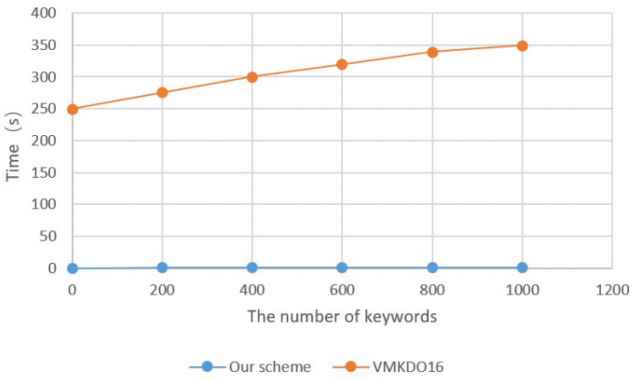

FIGURE 5. PEKS algorithm comparison.

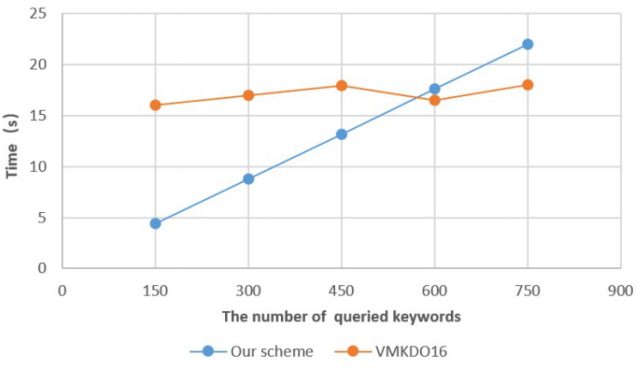

FIGURE 6. Test algorithm comparison.

In ***Test*** phase, Figure 6 shows that our scheme has much less computational overhead than the scheme VMKDO16 when there are fewer search keywords(Here we set $t \in [150, 750]$). But the computational time in our scheme linearly increases with the number of t, while that of the scheme VMKDO16 is almost constant. This is because the scheme VMKDO16 is constructed based on attribute encryption algorithm, so that the number of searching keywords mainly affects the time of trapdoor generation. In the ***Test*** phase, no matter the number of keywords, the ***Test*** algorithm only needs 3 power exponential operations and 2 bilinear pairing operations. Our scheme is constructed based on Homomorphic encryption algorithm, and the encrypted indexes include keywords, so it will increase linearly with the increase of the number of queried keywords. Therefore, when t is large enough, the scheme VMKDO16 will perform better than our scheme. Fortunately, users generally submit the number of searching keyword is not big. Thus, our scheme is still acceptable in practice, and perform better than the scheme VMKDO16 when there are fewer search keywords.

VII. CONCLUSION

A verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting is proposed. Our scheme allows the server to build an inverted encryption index structure without a query trapdoor, so that the time complexity of single keyword search is reduced to $O(m)$. Experiments demonstrate it has an obvious advantage than others. Moreover, our scheme can verify the correctness and completeness of searching results in multi-user setting and allow multiple users to perform encrypted keyword queries over encrypted data. Security analysis show it is secure based on the Approximate-GCD problem under random oracle.

REFERENCES

[1] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2004, pp. 506–522.
[2] D. Boneh and B. Water, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography*. Berlin, Germany: Springer, 2007, pp. 535–554.
[3] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Computational Science and Its Applications—ICCSA*. Berlin, Germany: Springer, 2008, pp. 1249–1259.
[4] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy, "Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data," in *Public Key Cryptography—PKC*. Berlin, Germany: Springer, 2009, pp. 196–214.
[5] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.
[6] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *Proc. IEEE 31st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2011, pp. 383–392.
[7] C. Gu, Y. Guang, Y. Zhu, and Y. Zheng, "Public key encryption with keyword search from lattices," *Int. J. Inf. Technol.*, vol. 19, no. 1, pp. 1–10, 2013.
[8] C. Hou, F. Liu, H. Bai, and L. Ren, "Public-key encryption with keyword search from lattice," in *Proc. IEEE 8th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. (3PGCIC)*, Oct. 2013, pp. 336–339.
[9] B. Wang, W. Song, W. Lou, and Y. T. Hou, "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 2092–2100.
[10] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Jan. 2016.

[11] R. Chen *et al.*, "Server-aided public key encryption with keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.
[12] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 715–725, Sep./Oct. 2017.
[13] K. Emura, G. Hanaoka, K. Nuida, G. Ohtake, T. Matsuda, and S. Yamada, "Chosen ciphertext secure keyed-homomorphic public-key cryptosystems," *Des., Codes Cryptogr.*, vol. 86, no. 8, pp. 1623–1683, 2018.
[14] R. C. Merkle, "A certified digital signature," in *Proc. Int. Conf. Adv. Cryptol.*, 1989, pp. 218–238.
[15] L. Che, L. Xu, S. Zhou, and X. Huang, "Data dynamics for remote data possession checking in cloud storage," *Comput. Elect. Eng.*, vol. 39, no. 7, pp. 2413–2424, 2013.
[16] X. Wang and D. Yuan, "A query verification scheme for dynamic outsourced databases," *J. Comput.*, vol. 37, no. 1, pp. 156–160, 2013.
[17] Y. Yu, Y. Zhang, J. Ni, M. H. Au, L. Chen, and H. Liu, "Remote data possession checking with enhanced security for cloud storage," *Future Gener. Comput. Syst.*, vol. 52, pp. 77–85, Nov. 2015.
[18] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1717–1726, Sep. 2013.
[19] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Integrity for distributed queries," in *Proc. IEEE Conf. Commun. Netw. Secur.*, Oct. 2014, pp. 1–9.
[20] C. Guo *et al.*, "Key-aggregate authentication cryptosystem for data sharing in dynamic cloud storage," *Future Gener. Comput. Syst.*, vol. 84, no. 7, pp. 190–199, 2018.
[21] M. Sookhak, A. Gani, M. K. Khan, and R. Buyya, "Dynamic remote data auditing for securing big data storage in cloud computing," *Inf. Sci.*, vol. 380, pp. 101–116, Feb. 2017.
[22] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017.
[23] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 917–922.
[24] C. Guo, R. Zhuang, Y. Jie, Y. Ren, T. Wu, and K. Choo, "Fine-grained database field search using attribute-based encryption for e-healthcare clouds," *J. Med. Syst.*, vol. 40, no. 11, pp. 235–242, 2016.
[25] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: verifiable attribute-based keyword search over outsourced encrypted data," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 522–530.
[26] W. Sun *et al.*, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, Nov. 2014.
[27] C. Guo, X. Chen, Y. Jie, F. Zhang, M. Li, and B. Feng, "Dynamic multi-phrase ranked search over encrypted data with symmetric searchable encryption," *IEEE Trans. Services Comput.*, pp. 1–12, Oct. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8089767/, doi: 10.1109/TSC.2017.2768045.
[28] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2010, pp. 24–43.
[29] M. N. Krohn, M. J. Freedman, and D. Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 226–240.
[30] M. T. Goodrich *et al.*, "Efficient verification of Web-content searching through authenticated Web crawlers," *Proc. VLDB Endowment*, vol. 5, no. 10, pp. 920–931, 2012.
[31] L. Guo, B. Lu, X. Li, and H. Xu, "A verifiable proxy re-encryption with keyword search without random oracle," in *Proc. Int. Conf. Comput. Intell. Secur.*, 2013, pp. 474–478.
[32] Y. Yang and M. Ma, "Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 746–759, Apr. 2016.
[33] Y. Miao, J. Ma, X. Liu, Z. Liu, L. Shen, and F. Wei, "VMKDO: Verifiable multi-keyword search over encrypted cloud data for dynamic data-owner," *Peer-Peer Netw. Appl.*, vol. 11, no. 2, pp. 287–297, 2016.
[34] E.-J. Goh, "Secure indexes," Cryptol. ePrint Arch., Tech. Rep. 2003/216, 2003. [Online]. Available: http://eprint.iacr.org/2003/216
[35] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2005, pp. 442–455.
[36] R. Zhang, R. Xue, T. Yu, and L. Liu, "Dynamic and efficient private keyword search over inverted index–based encrypted data," *ACM Trans. Internet Technol.*, vol. 16, no. 3, 2016, Art. no. 21.

D. N. WU received the M.Sc. degree in computer engineering from Jinan University, China, in 2017. Her research interests include security and privacy in cloud computing.

Q. Q. GAN received the M.Sc. degree in software engineering from Jinan University, China, in 2016, where she is currently pursuing the Ph.D. degree. Her research interests include security and privacy in cloud computing.

X. M. WANG received the B.Sc. degree from the Harbin Institute of Technology, China, and the Ph.D. degree from Nankai University, China. She is currently a Professor with the Department of Computer Science, Jinan University, China. Her research interests include security and privacy in network and distributed systems, such as wireless sensor networks and cloud computing with a focus on security protocol designs and access control.