Achieving secure, universal, and fine-grained query results verification for secure search scheme over encrypted cloud data

Hui Yin, Zheng Qin, Jixin Zhang, Lu Ou, and Keqin Li, *Fellow, IEEE*

Abstract—Secure search techniques over encrypted cloud data allow an authorized user to query data files of interest by submitting encrypted query keywords to the cloud server in a privacy-preserving manner. However, in practice, the returned query results may be incorrect or incomplete in the dishonest cloud environment. For example, the cloud server may intentionally omit some qualified results to save computational resources and communication overhead. Thus, a well-functioning secure query system should provide a query results verification mechanism that allows the data user to verify results. In this paper, we design a secure, easily integrated, and fine-grained query results verification mechanism, by which, given an encrypted query results set, the query user not only can verify the correctness of each data file in the set but also can further check how many or which qualified data files are not returned if the set is incomplete before decryption. The verification scheme is loose-coupling to concrete secure search techniques and can be very easily integrated into any secure query scheme. We achieve the goal by constructing secure *verification object* for encrypted cloud data. Furthermore, a short signature technique with extremely small storage cost is proposed to guarantee the authenticity of *verification object* and a *verification object* request technique is presented to allow the query user to securely obtain the desired *verification object*. Performance evaluation shows that the proposed schemes are practical and efficient.

Index Terms—Cloud computing, Query results verification, Secure query, Verification object.

1 Introduction

1.1 Motivation

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. Driven by the abundant benefits brought by the cloud computing such as cost saving, quick deployment, flexible resource configuration, etc., more and more enterprises and individual users are taking into account migrating their private data and native applications to the cloud server. A matter of public concern is how to guarantee the security of data that is outsourced to a remote cloud server and breaks away from the direct control of data owners [2]. Encryption on private data before outsourcing is an effective measure to protect data confidentiality [3]. However, encrypted data make effective data retrieval a very challenging task.

To address the challenge (i.e., search on encrypted data), Song *et al.* first introduced the concept of searchable encryption and proposed a practical technique that allows users to search over encrypted data through encrypted query keywords in [4]. Later, many searchable encryption schemes were proposed based on symmetric key and public-key setting to strengthen security and improve query efficiency [5], [6], [7], [8], [9], [10], [11], [12]. Recently, with the growing popularity of cloud computing, how to securely and efficiently search over encrypted cloud data becomes a research focus. Some approaches have been proposed based on traditional searchable encryption schemes in [13], [14], [15], [16], [17], [18], [19], [20], [21], which aim to protect data security and query privacies with better query efficient for cloud computing. However, all of these schemes are based on an ideal assumption that the cloud server is an "honest-but-curious" entity and keeps robust and secure software/hardware environments. As a result, correct and complete query results always be unexceptionally returned from the cloud server when a query ends every time. However, in practical applications, the cloud server may return erroneous or incomplete query results once he behaves dishonestly for illegal profits such as saving computation and communication cost or due to possible software/hardware failure of the server [22].

Therefore, the above fact usually motivates data users to verify the correctness and completeness of query results. Some researchers proposed to integrate the query results verification mechanisms to their secure search schemes [23], [24], [25], [26], (e.g., embedding verification information into the specified secure indexes or query results). Upon receiving query results, data users use specified verification information to verify their correct-

- *Hui Yin, Zheng Qin, Lu Ou, and Jixin Zhang are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China. Hui Yin is also with the Department of Mathematics and Computer Science, Changsha University, Changsha 410022, China. Zheng Qin is the Corresponding Author. E-mail: zqin@hnu.edu.cn*
- *Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, New York 12561, USA. Email: lik@newpaltz.edu*

ness and completeness. There are two limitations in these schemes:

1) These verification mechanisms provide a coarse-grained verification, i.e., if the query result set contains all qualified and correct data files, then these schemes reply *yes*, otherwise reply *no*. Thus, if the verification algorithm outputs *no*, a data user has to abort the decryption for all query results despite only one query result is incorrect.

2) These verification mechanisms are generally tightly coupled to corresponding secure query constructions and have not universality.

In a search process, for a returned query results set that contains multiple encrypted data files, a data user may wish to verify the correctness of each encrypted data file (thus, he can remove incorrect results and retain the correct ones as the ultima query results) or wants to check how many or which qualified data files are not returned on earth if the cloud server intentionally omits some query results. These information can be regarded as a hard evidence to punish the cloud server. This is challenging to achieve the fine-grained verifications since the query and verification are enforced in the encrypted environment. In [27], we proposed a secure and fine-grained query results verification scheme by constructing the *verification object* for encrypted outsourced data files. When a query ends, the query results set along with the corresponding verification object are returned together, by which the query user can accurately verify: 1) the correctness of each encrypted data file in the results set; 2) how many qualified data files are not returned and 3) which qualified data files are not returned. Furthermore, our proposed verification scheme is lightweight and loose-coupling to concrete secure query schemes and can be very easily equipped into any secure query scheme for cloud computing.

However, some necessary extensions and important works need to be further supplied to perfect our original scheme such as detailed performance evaluation and formal security definition and proof. More importantly, in the dishonest cloud environment, the scheme suffers from the following two important security problems:

1) Just as possibly tampering or deleting query results, the dishonest cloud server may also tamper or forge verification objects themselves to make the data user impossible to perform verification operation. Specially, once the cloud server knows that the query results verification scheme is provided in the secure search system, he may return inveracious verification object to escape responsibilities of misbehavior.

2) When a data user wants to obtain the desired verification object, some important information will be revealed such as which verification objects are being or have been requested before frequently, etc. These information may leak query user's privacy and expose some useful contents about data files. More importantly, these exposed information may become temptations of misbehavior for the cloud server. We will detailedly describe this part content in Section 7.

1.2 Our Contributions

In this paper, we extend and reinforce our work in [27] to make it more applicable in the cloud environment and more secure to against dishonest cloud server. The main contributions of this paper are summarized as follows:

1) We formally propose the verifiable secure search system model and threat model and design a fine-grained query results verification scheme for secure keyword search over encrypted cloud data.
2) We propose a short signature technique based on certificateless public-key cryptography to guarantee the authenticity of the verification objects themselves.
3) We design a novel verification object request technique based on Paillier Encryption, where the cloud server knows nothing about what the data user is requesting for and which verification objects are returned to the user.
4) We provide the formal security definition and proof and conduct extensive performance experiments to evaluate the accuracy and efficiency of our proposed scheme.

The rest of this paper is organized as follows. We reviews the related work in Section 2. Section 3 illustrates background and presents the preliminary techniques. We propose the query results verification scheme in Section 4 and the a discussion of the scheme is shown in Section 5. We describe the signature and authentication of verification object in Section 6. In Section 7, a secure verification object request mechanism is proposed. We analyze the security and evaluate performances of our proposed scheme in Sections 8 and 9. In Section 10, we conclude the paper.

2 Related Work

2.1 Secure Search in Cloud Computing

Essentially, the secure search is thus a technique that allows an authorized data user to search over the data owner's encrypted data by submitting encrypted query keywords in a privacy-preserving manner and is an effective extension of traditional searchable encryption to adapt for the cloud computing environment. Motivated by the effective information retrieve on encrypted outsourced cloud data, Wang *et al.* first proposed a keyword-based secure search scheme [13] and later the secure keyword search issues in cloud computing have been adequately researched [14], [15], [16], [17], [18], [19], [20], [21], which aim to continually improve search efficiency, reduce communication and computation cost, and enrich the category of search function with better security and privacy protection. A common basic assumption of all these schemes is that the cloud is considered to be an "honest-but-curious" entity as well as always keeps robust and secure software/hardware environments. As a result, under the ideal assumption, the correct and complete query results always be unexceptionally returned from the cloud server when a query ends every time.

2.2 Verifiable Secure Search in Cloud Computing

In practical applications, the cloud server may return erroneous or false search results once he behaves dishonestly for illegal profits or due to possible software/hardware failure of the cloud server. Because of the possible data corruption under a dishonest setting, serval research works have been proposed to allow the data user to enforce query results verification in the secure search fields for cloud computing. In [23], Wang *et al.* applied hash chain technique to implement the completeness verification of query results by embedding the encrypted verification information into their proposed secure searchable index. In [24], Sun *et al.* used encrypted index tree structure to implement secure query results verification functionality. In this scheme, when the query ends, the cloud server returns query results along with a minimum encrypted index tree, then the data user searches this minimum index tree using the same search algorithm as the cloud server did to finish result verification. Zheng *et al.* [25] constructed a verifiable secure query scheme over encrypted cloud data based on attribute-based encryption technique (ABE) [28] in the public-key setting. Sun *et al.* [26] referred to the Merkle hash tree and applied Pairing operations to implement the correctness and completeness verification of query results for keyword search over large dynamic encrypted cloud data. However, these secure verification schemes cannot achieve our proposed fine-grained verification goals. Furthermore, these verification mechanisms are generally tightly coupled to corresponding secure query schemes and have not universality.

3 Background

To clarify our proposed problems, in this section, we present our system model, threat model, and several preliminaries used to implement our scheme.

3.1 System Model

The system model of the secure search over encrypted cloud data usually includes three entities: data owners, data users, and the cloud server, which describes the following scenario: data owners encrypt their private data and upload them to cloud server for enjoying the abundant benefits brought by the cloud computing as well as guaranteeing data security. Meanwhile, the secure searchable indexes are also constructed to support effective keyword search over encrypted outsourced data. An authorized data user obtains interested data files from the cloud server by submitting query trapdoors (encrypted query keywords) to the cloud server, who performs search over secure indexes according to trapdoors and sends the query results to the data user.

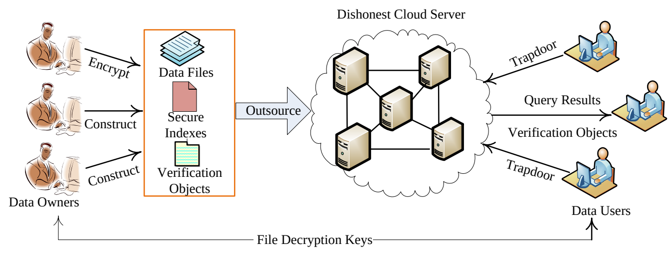


Fig. 1. A system model of verifiable secure search over encrypted cloud data

The above application scenario is based on an ideal assumption that the cloud server is considered as an honest entity and always honestly returns all qualified query results. In this paper, we consider a more challenging model, where the query results would be maliciously deleted or tampered by the dishonest cloud server. When the query results face the risks that are deleted or tampered, a well-functioning secure query system should provide a mechanism that allows the data user to verify the correctness and completeness of query results. To achieve the results verification goal, we propose to construct secure verification objects for data files that are outsourced to the cloud with encrypted data and secure indexes together. The query results along with corresponding data verification object are returned to the data user when a query ends. The improved system model of verifiable secure search over encrypted cloud data is illustrated in Fig. 1.

3.2 Threat Model

In this paper, compared with the previous works, an important distinction about the threat model is that the cloud is considered to be an untrusted entity. More specifically, first of all, the cloud server tries to gain some valuable information from encrypted data files, secure indexes, and verification objects (e.g., a misbehaving cloud administrator aims at obtaining these information for possible monetary profits). Then, the cloud server would intentionally return false search results for saving computation resource or communication cost. Further, if the cloud server knows a query results verification mechanism is embedded, he may tamper or forge verification objects to escape responsibilities of misbehavior.

Similar to the previous works, both data owners and authorized data users are considered to be trusted in our threat model.

3.3 Preliminaries

3.3.1 *Bloom Filter*

A Bloom Filter [29] is a space-efficient probabilistic data structure which is used to test whether an element is a member of a set. An empty Bloom filter is a bit array of m bits, where all bits are set to 0 initially. Given a set $S = \{a_1, a_2, ..., a_n\}$ of n elements, in order to insert an element $a \in S$ into a Bloom filter, it needs to use l independent hash functions $h_1, ..., h_l$ with the same output range $[0, m-1]$ to hash a to get l different positions in the Bloom filter, and sets all these positions to 1. To determine whether an element b is in S or not, it checks whether all the positions corresponding to $h_i(b)$ equal to 1, $1 \le i \le l$. If all are 1, then $a \in S$ or resulting in a false positive due to hash collision. If any one of these positions is 0, then $b \notin S$. We can control false positive rate by adjusting Bloom Filter parameters. If n elements are inserted into an m-bit Bloom filter which uses l independent hash functions, the false positive rate is $fp = (1-(1-\frac{1}{m})^{ln})^l$, where $\frac{1}{m} \to 0, fp \approx (1-e^{-\frac{ln}{m}})^l$. Therefore, given m and n, when $l = \frac{m}{n}\ln 2$, the minimum false positive rate is 2^{-l}. As a matter of fact, the standard Bloom Filter does not support element deletion operations because each bit of the Bloom Filter cannot record the number of hash collisions when inserting different elements. To allow effective element deletion, in [30], Fan *et al.* designed Counting Bloom Filter by using fixed size counters to represent an element instead of single bits. When an element is inserted, the corresponding counters are incremented by 1 and the corresponding counters are decreased by 1 when an element is deleted.

3.3.2 Pseudo Random Function

A pseudo-random function [31] prf: $\{0,1\}^* \times \{0,1\}^\tau \to \{0,1\}^s$ is a computationally efficient function, which maps an arbitrary length string $x \in \{0,1\}^*$ to a random s-bit string y under a given key $\lambda \in \{0,1\}^\tau$ such that y looks like being randomly chosen from the range space $\{0,1\}^s$. It satisfies the following properties:

- Computability: Given $x \in \{0,1\}^*$ and $\lambda \in \{0,1\}^\tau$, there is a polynomial time algorithm to compute $prf(\lambda, x)$.
- Collision Resistance: Give two distinct numbers $x, y \in \{0,1\}^*$ and $\lambda \in \{0,1\}^\tau$, it is computationally infeasible to satisfy $prf(\lambda, x) = prf(\lambda, y)$.
- One-wayness: Give the value $prf(\lambda, x)$, it is computationally infeasible to calculate x and λ.

3.3.3 Bilinear Map

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic multiplicative groups with the same large prime order q. A bilinear map [32], $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, satisfies the following properties:

- Computable: For any $Q, Z \in \mathbb{G}_1$, there is a polynomial time algorithm to compute $e(Q, Z) \in \mathbb{G}_2$.
- Bilinear: For all $x, y \in \mathbb{Z}_q^*$ and $Q, Z \in \mathbb{G}_1$, the equality $e(Q^x, Z^y) = e(Q, Z)^{xy}$ holds.
- Non-degenerate: If g, h are generators of $\mathbb{G}_1$, then $e(g, h)$ is a generator of $\mathbb{G}_2$.

3.3.4 Paillier Encryption

Paillier encryption [33] is a public-key encryption scheme with the remarkable additive homomorphic property and normally consists of Gen, Enc, and Dec

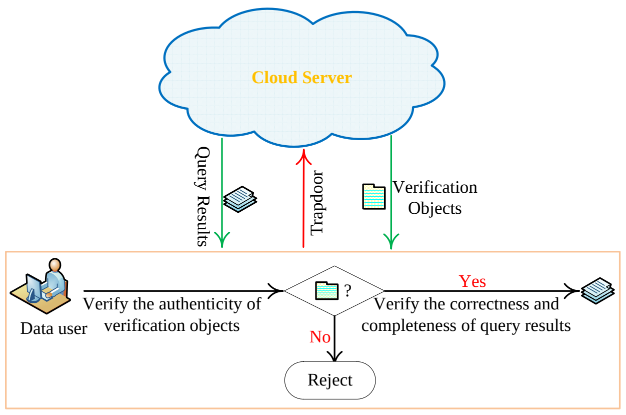

Cloud Server
Query Results
Trapdoor
Verification Objects
Data user
Verify the authenticity of verification objects
?
Yes
No
Verify the correctness and completeness of query results
Reject

Fig. 2. The process of query results verification

three polynomial-time algorithms. We briefly introduce these algorithms as follows:

- $Gen(1^n)$: The probabilistic polynomial-time algorithm takes the secure parameter n as input and outputs $(N, p, q, \psi(N))$ where $N = pq$, p and q are n-bit primes, and $\psi(N) = (p-1)(q-1)$. The public key is $pk = N$ and the private key is $sk = < N, \psi(N) >$.
- $Enc(pk, m)$: The Enc is a probabilistic polynomial-time algorithm, which takes the public key pk and a message m as input and outputs the ciphertext of m
$$c = [(1+N)^m \cdot r^N \mod N^2]$$
where r is a randomly chosen number from Z_N^*.
- $Dec(sk, c)$: The Dec is a deterministic polynomial-time algorithm, which takes the private key sk and the ciphertext c of the message m as input and outputs m
$$m = \left[\frac{[c^{\psi(N)} \mod N^2] - 1}{N} \cdot \psi(N)^{-1} \mod N\right]$$

4 Query Results Verification Scheme

4.1 Scheme Overview and Problem Definition

Fig. 2 shows an overview of the query results verification process. In brief, when a query ends, both query results and corresponding verification objects are returned to the data user by the cloud server. Upon receiving these data, the data user first checks the authenticity of verification objects and then continued to verify query results according to the verification objects if verification objects pass the test; otherwise, the data user rejects this query. The notations used in this paper are shown in Table 1.

In what follows, we further state our proposed problem. Given F, the data owner first forms the files collection $\{F_w\}_{w \in W}$ and uses any semantically secure encryption scheme such as AES to encrypt $\{F_w\}_{w \in W}$ to get the ciphertext files collection $\{C_w\}_{w \in W}$. Given a query trapdoor of keyword w, the cloud server returns R_w to the data user. Theoretically, the set R_w should be equal to C_w, however, which may be incomplete

TABLE 1
Notations used in this paper

Notations	Description
F	The plaintext set of data files
C	The ciphertext set of data files
W	The keywords dictionary
w	Any keyword in W
F_w	The set of data files containing the keyword w
C_w	The corresponding cipertext set of F_w
R_w	The set of query results containing the keyword w
VO_w	The verification object of C_w
prf_k	A pseudo-random function with the key k
$\mathcal{H}$	A hash function family with l hash functions h_1, h_2 $..., h_l$ with the same range $[0, m-1]$
$\|\alpha\|$	If α is a string, $\|\alpha\|$ denotes the bit length of α; If α is a set, $\|\alpha\|$ denotes the cardinality of α

(i.e., $|R_w| < |C_w|$) or contain some incorrect data due to the possible misbehaviours of the dishonest cloud server. To allow an authorized data user to verify the query results R_w, our main idea is that the data owner constructs a secure verification object VO_w for each C_w in $\{C_w\}_{w\in W}$, by which the data user can efficiently verify the correctness and completeness of the returned query result set R_w. Let $\{VO_w\}_{w\in W}$ denote the corresponding verification objects collection of $\{C_w\}_{w\in W}$.

4.2 The Verification Object Construction

To maximize reduce storage and communication cost and achieve privacy guarantee of the verification objects, in this paper, we will utilize Counting Bloom Filters and the pseudo-random function prf_k to construct our verification objects, on which the authorized data user can efficiently perform query results verification. Next, we elaborate on the construction process of verification objects as follows.

Given a ciphertext set C_w of F_w, the data owner first generates a Counting Bloom Filter VO_w with m counters, in which each counter is set to be 0 initially. Then, for each encrypted data file $c \in C_w$, he uses the pseudo-random function prf_k under the key k to calculate the secret value $prf_k(c)$. Further, he continues to uses l hash functions $h_1, ..., h_l$ of the hash function family $\mathcal{H}$ to hash the $prf_k(c)$ to get $h_1(prf_k(c) \in [0, m-1], ..., h_l(prf_k(c)) \in [0, m-1]$. Lastly, the data owner inserts these hash values into the Counting Bloom Filter VO_w by performing operations that the corresponding counter $VO_w[h_i(prf(c))], 1 \leq i \leq l$ is increased by 1. Our basic idea is to let the VO_w represent the verification object of C_w.

However, C_w's verification object VO_w reveals the number of the data files contained in C_w and the cloud server can easily obtain the statistical information that how many data files are in C_w by calculating $\sum_{i=0}^{m-1} VO_w[i])/l$. To avoid revealing the size of the set C_w, an effective method is to all VO_ws in $\{VO_w\}_{w\in W}$ have the same value $(\sum_{i=0}^{m-1} VO_w[i])$ by padding different number of random elements for different verification object VO_w. However, directly padding random elements into VO_w by hashing these random elements to the range $[0, m-1]$ will disable the effective query results verification since the VO_w contains some invalid data (i.e., random elements).

To address this problem, we propose to let the data owner generate a pad region only used for random elements pad. Specifically, given a set C_w, the data owner first generates a Bloom Filter VO_w with n counters and inserts all data files in C_w into the first m counters using prf_k and $\mathcal{H}$. Then, let $|C_w|_{max}$ denote the maximum number of data files containing some keyword $w \in W$, i.e., $|C_w|_{max} = max\{|C_{w_i}|, i = 1, ..., |W|\}$, the data owner generates $l \times |C_w|_{max} - \sum_{i=0}^{m-1} VO_w[i]$ random strings $\{R_1, R_2, ...\}$ and uses a pad function $\mathcal{P}$ with the range $[m, n-1]$ to compute $\mathcal{P}(R)$, the corresponding position $VO_w[\mathcal{P}(R)]$ is increased by 1. The pad function $\mathcal{P}$ can be defined as:

$$\mathcal{P}(x) = m + (prf_k(x) \bmod (n-m)), x \in \{0,1\}^*$$

After padding, all VO_ws in $\{VO_w\}_{w\in W}$ satisfy:

$$\sum_{j=0}^{n-1}(VO_w[j]) = l \times |C_w|_{max}$$

Fig. 3 shows an example of our verification object. The whole process of constructing verification objects is shown in Algorithm 1.

Algorithm 1 Constructing Verification Objects

```
Input:
    The ciphertext files collection $\mathbb{C} = \{C_w\}_{w\in W}$
Output:
    The verification objects collection $\{VO_w\}_{w\in W}$
 1: Generate an empty set $\mathbb{VO}$={};
 2: for each $C_w \in \mathbb{C}$ do
 3:     Generate a Counting Bloom Filter $VO_w$ with $n$ counters;
 4:     for each $c \in C_w$ do
 5:         Calculate $v_c = prf_k(c)$;
 6:         Calculate $h_1(v_c), ..., h_l(v_c)$ using hash function family $\mathcal{H}$;
 7:         $VO_w[h_1(v_c)], ..., VO_w[h_l(v_c)]$ are increased by 1 in $VO_w$;
 8:     end for
 9:     Generate $l \times (|C_w|_{max} - |C_w|)$ random strings $R_1$, $R_2, ...$
10:     Calculate $\mathcal{P}(R_1), \mathcal{P}(R_2), ...$
11:     $VO_w[\mathcal{P}(R_1)], VO_w[\mathcal{P}(R_2)], ...$ are increased by 1 in $VO_w$;
12:     Add the $VO_w$ into $\mathbb{VO}$
13: end for
14: return $\mathbb{VO} = \{VO_w\}_{w\in W}$
```

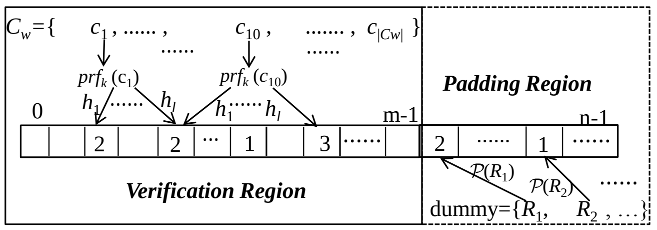


Fig. 3. An example of the verification object

4.3 Verifying the Correctness and Completeness of Query Results

When a query ends, the query results set and corresponding verification object are together returned to the query user, who verifies the correctness and completeness of query results based on the verification object. Our proposed query results verification scheme not only allows the query user to easily verify the correctness of each encrypted data file in the query results set, but also enables the data user to efficiently perform completeness verification before decrypting query results. More importantly, for an incomplete query results set, our verification objects can definitely tell the data user that the cloud server has omitted how many qualified data files for a query, which is a significant advantage compared with the previous related works.

We give an example to illustrate how to perform query results verifications. Assume that an authorized data user submits an encrypted query keyword w to the cloud server, the cloud server performs query operations and returns back the query results set R_w along with the corresponding verification object VO_w. The data user only needs two steps to finish the correctness verification for each data file c in R_w. First, the data user uses the shared key k and the hash functions family $\mathcal{H}$ to calculate $h_1(prf_k(c)), ..., h_l(prf_k(c))$. Second, he checks all counters correspond to $VO_w[h_1(prf_k(c))], ..., VO_w[h_l(prf_k(c))]$. If these counters are all greater than 0, then c is a correct query result; otherwise, c is regarded as an incorrectness query result as long as one of them is equal to 0 and should be removed from R_w. The correctness verification process is shown in Algorithm 2.

In terms of the query results completeness verification, our well constructed verification object is able to allow the data user to quickly find out the number of data files that satisfy the query yet are not returned by the cloud server if the cloud server does intentionally omit some data files. The verification process is described in detail as follows. Given R_w and VO_w, the data user first invokes the **Algorithm 2** to do the correctness verification. Then, If $|R_w| \neq 0$, for each correct data file c, he calculates $h_1(prf_k(c)), ..., h_l(prf_k(c))$ using the shared key k and the corresponding counters $VO_w[h_1(prf_k(c))], ..., VO_w[h_l(prf_k(c))]$ in VO_w are decreased by 1. Finally, the data user calculates the

Algorithm 2 Correctness Verification

```
Input:
    The query results set $R_w$ and corresponding verification object $VO_w$ for some query keyword $w$.
Output:
    $R_w$
1: for each $c$ in $R_w$ do
2:     Calculate $v_c = prf_k(c)$;
3:     Calculate $h_1(v_c), ..., h_l(v_c)$ using hash function family $\mathcal{H}$;
4:     Check all counters $h_1(v_c), ..., h_l(v_c)$ in $VO_w$. If one of them is equal to 0, then remove $c$ from $R_w$;
5: end for
6: return $R_w$.
```

following value:

$$Rem = \left(\sum_{j=0}^{m-1} VO_w[j] \right) / l$$

If $Rem = 0$, then the cloud server has returned back all data files that satisfy the query; otherwise, the data user can confirm that the cloud server has omitted Rem qualified data files in the query. The completeness verification is shown in Algorithm 3.

Algorithm 3 Completeness Verification

```
Input:
    The query results set $R_w$ that has past correctness verification, and corresponding verification object $VO_w$ for some query keyword $w$.
Output:
    The number of data files that have not been returned by the cloud server $Rem$
1: if $|R_w| \neq 0$ then
2:     for each $c$ in $R_w$ do
3:         Calculate $v_c = prf_k(c)$;
4:         Calculate $h_1(v_c), ..., h_l(v_c)$ using hash function family $\mathcal{H}$;
5:         The corresponding counters $h_1(v_c), ..., h_l(v_c)$ in $VO_w$ are decreased by 1;
6:     end for
7: end if
8: Calculate $Rem = \frac{\sum_{j=0}^{m-1}(VO_w[j])}{l}$
9: return $Rem$.
```

5 Discussion

5.1 About False Positive

It is worth noticing that our proposed scheme may allow an incorrect query result to pass the correctness verification due to the fact that Bloom Filter may yield false positives with a certain probability because of hash collisions. Fortunately, we can adjust the Bloom Filter parameters to minimize the false positive rate. More specifically, in this paper, given the bit length m of each verification object VO_w and the number of data files in C_w, i.e., $|C_w|$, we set the number of hash functions l to be $\frac{m}{|C_w|} \times \ln 2$ to minimize the false positive rate to be:

$$1-\left(1-\frac{1}{m}^{l|C_w|}\right)^l \approx (1-e^{-l|C_w|/m})^l = 2^{-l} \approx 0.6185^{m/|C_w|}$$

However, the set size $|C_{w_i}|$ is different for different $C_{w_i}, w_i \in W, 1 \leq i \leq |W|$, to choose the same hash functions family $\mathcal{H}$ with the same number of hash functions l for all verification object in $\{VO_w\}_{w \in W}$, we set

$$l = \frac{m}{|C_w|_{max}} \times \ln 2$$

where $|C_w|_{max}$ is the maximum number of data files containing a certain keyword w. Thus, for any set $C_{w'}$ that satisfies $|C'_w| < |C_w)|_{max}$, the false positive rate incurred by the corresponding $VO_{w'}$ is less than $0.6185^{m/|C_w|_{max}}$.

For example, to guarantee that the false positive rate induced by Bloom Filter is less than 0.01, we can set the number of hash functions to be $l = \log_{\frac{1}{2}} 0.01 = 7$ and the length of Bloom Filter to be $m = |C_w|_{max} \log_{0.6185} 0.01$.

5.2 Size of Verification Objects

In addition, the bit length of each counter in VO should be discussed adequately, because the huge VO will bring heavy storage cost and communication cost, which makes the verification object impractical. Generally, the 4 bits for each counter are sufficient for Counting Bloom Filters. We give a simple derivation and more detailed analysis please refer to [30]. Theoretically, the probability that the i-th counter has been increased by j times, when inserting the set $C_w, 1 \leq i \leq m, w \in W$ using l hash functions, can be denoted:

$$Pr(c(i) = j) = \binom{|C_w|l}{j}(\frac{1}{m})^j\left(1-\frac{1}{m}\right)^{|C_w|l-j}$$

According to the Stirling's approximation, the probability that any counter is greater or equal j is:

$$Pr(c(i) \geq j) \leq \binom{|C_w)|l}{j}(\frac{1}{m^j}) \leq \left(\frac{e|C_w|l}{jm}\right)^j$$

To minimize the false positive rate, we set $l = \frac{m}{|C_w|} \times \ln 2$ and further simplify the above inequality:

$$pr(c(i) \geq j) \leq m\left(\frac{e \ln 2}{j}\right)^j$$

Hence, if we set the bit length of each counter to be 4, obviously, when $j = 16$, the counter happens overflow and the probability can be calculated as $pr(c(i) \geq 16) \leq 1.37 \times 10^{-15} \times m$. Obviously, the value is infinitesimal and the probability of happening overflow can be ignored.

5.3 An Enhanced Completeness Verification Construction

For completeness verification, a significant advantage of our constructed verification object is that the query user can quickly count the number of data files omitted by the cloud server for an incomplete query result set. As a more strict completeness verification requirement (i.e., which qualified data files are not returned in a query), an enhanced verification object based on the Counting Bloom Filter was proposed in our work [27], which further allows the data user to definitely obtain the file identifier of each data file that satisfies the query yet is omitted by the cloud server, by reasonably designing the identifiers of data files and secretly preserving them in the corresponding verification object. Here, we are not intend to elaborate on the verification construction to avoid unnecessary repetition. Interested readers please refer to [27] to obtain details about this.

6 Signature and Authentication of Verification Object

Under the dishonest threat model, the cloud server may tamper or forge verification objects to escape responsibilities of misbehaviour if it knows a query results verification mechanism is involved in the secure search scheme. Therefore, authenticating the verification object itself is the first indispensable step for the data user when he receives query results and the corresponding verification object from the cloud server. To achieve this goal as well as prevent the cloud server from sending forged verification object to a data user, digital signature over verification objects is a natural and good choose. That is to say, on one hand, the data owner computes a signature on each verification object using his private key after constructing verification objects; on the other hand, the data user verify the authenticity of verification object using the data owner's public key upon receiving the specified verification object in a query.

However, traditional digital signature techniques such as DSA require certificates to guarantee the authenticity of public key by Certification Authority, which is generally considered to be costly to use due to expensive certificate library management and maintenance problems. To eliminate the certificates, certificateless signature schemes [34], [35], [36], [37], [38] have been proposed based on the certificateless cryptography [39], [40]. In this paper, we present an efficient short signature scheme based on [34] and [36] to significantly reduce the storage and communication cost of signatures of verification objects. Compared to above existing schemes, the signature length of our scheme is only a half or one-third of these schemes. In certificateless cryptography, there needs a trusted Key Generation Center (KGC) to help the user to generate public/private key pair, while KGC cannot obtain user's public/private key.

Assume that there exists a trusted KGC in our system, o is a data owner and u denotes an authorized data user, our proposed certificateless signature for verification object is composed of the following seven steps.

- Setup: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic multiplicative groups of prime order q, which are equipped with an efficiently computable bilinear map e. Let g be a generator of $\mathbb{G}_1$. On input a sufficiently large secure parameter l, KGC generates system master key $mk \in \mathbb{Z}_q^*$ and system public key $S_{pub} = g^{mk}$ as well as two cryptographic one-way hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 : \{0,1\}^* \times \mathbb{G}_1 \rightarrow \mathbb{Z}_q^*$. KGC opens the system public parameter $params = \{\mathbb{G}_1, \mathbb{G}_2, e, q, g, S_{pub}, H_1, H_2\}$ and keeps mk as secret.
- Extract-partial-private-key: o sends his/her identity $ID \in \{0,1\}^*$ to KGC. The KGC computes $Q_{ID} = H_1(ID)$, $d_{ID} = g^{\frac{1}{mk+Q_{ID}}}$ and sends d_{ID} to o via secure communication channels. KGC opens a public value $PV_{ID} = S_{pub} \cdot g^{Q_{ID}}$ to data users.
- Set-secret-value: o randomly picks up a value $k_{ID} \in \mathbb{Z}_q^*$ as own secret value.
- Set-private-key: o first computes $H_1^*(ID)$ according to his own identity ID and then verifies whether the following equation holds or not.

$$\begin{aligned} e(S_{pub}{\cdot}g^{H_1^*(ID)}, d_{ID}) &= e(g^{mk} \cdot g^{H_1^*(ID)}, g^{\frac{1}{mk+Q_{ID}}}) \\ &= e(g^{mk+H_1^*(ID)}, g^{\frac{1}{mk+H_1(ID)}}) \\ &= e(g,g)^{mk+H_1^*(ID)/mk+H_1(ID)} \\ &= e(g,g) \end{aligned}$$

  If the above equation holds (i.e., $H_1^*(ID) = H_1(ID)$ holds), then o confirms that his identity ID is not tampered and o further determines the complete private key as $sk_{ID} =< d_{ID}, k_{ID} >$ according to the partial key d_{ID} and the secret value k_{ID}.
- Set-public-key: o computes $H_1(ID)$ and uses the secret value k_{ID} to generate his public key:

$$\begin{aligned} pk_{ID} &= [S_{pub} \cdot g^{H_1(ID)}]^{k_{ID}} \\ &= [S_{pub} \cdot g^{Q_{ID}}]^{k_{ID}} \\ &= (PV_{ID})^{k_{ID}} \end{aligned}$$

- Sign: Given a verification object VO_w, o signs the VO_w using his private key as follows:

$$\sigma = (d_{ID})^{\frac{1}{k_{ID}+H_2(VO_w, pk_{ID})}}$$

Upon receiving a query result set R_w and its verification object VO_w, the authorized data user u first verifies the authenticity of VO_w by invoking the verification algorithm **Ver**($params, VO_w, ID, pk_{ID}, \sigma$), which decides whether the following equation holds or not:

$$e(\sigma, pk_{ID} \cdot (PV_{ID})^{H_2(\mathrm{VO}(w), pk_{ID})}) = e(g,g)$$

If the above equation holds, then u continues to verify the correctness of R_w according to the verification object VO_w; otherwise, u refuses this query. If the returned VO_w is not forged or tampered with by the cloud server, the above equation holds because (for ease writing, let $\lambda = H_2(VO_w, pk_{ID})$):

$$\begin{aligned} & e(\sigma, pk_{ID}{\cdot}(PV_{ID})^{H_2(VO_w, pk_{ID})}) \\ &\quad = e((d_{ID})^{\frac{1}{k_{ID}+\lambda}}, pk_{ID} \cdot (PV_{ID})^{\lambda}) \\ &\quad = e(g^{\frac{1}{mk+Q_{ID}} \cdot \frac{1}{k_{ID}+\lambda}}, (PV_{ID})^{k_{ID}+\lambda}) \\ &\quad = e(g^{\frac{1}{(mk+Q_{ID})(k_{ID}+\lambda)}}, [S_{pub} \cdot g^{Q_{ID}}]^{k_{ID}+\lambda}) \\ &\quad = e(g^{\frac{1}{(mk+Q_{ID})(k_{ID}+\lambda)}}, g^{(mk+Q_{ID})(k_{ID}+\lambda)}) \\ &\quad = e(g,g) \end{aligned}$$

TABLE 2
Comparison of Different Signature Schemes

	[34]	[35]	[36]	[37]	[38]	Our
public key length (bits)	320	320	160	160	320	160
signature length (bits)	320	320	320	480	320	160

A comparison is presented in Table 2, where the group with 160-bit order is used to instantiate $\mathbb{G}_1$ and $\mathbb{G}_2$.

7 Securely Requesting Verification Object

Another important question is that how to obtain the correct verification object from the set $\{VO_w\}_{w \in W}$ without leaking any useful information to the cloud server. A simple way is to organize the set $\{VO_w\}_{w \in W}$ as $\{(w, VO_w)\}_{w \in W}$, thus the cloud server can very easily return back the correct verification object according to the submitted keyword w by the data user. However, the straightforward method compromises the security and privacy of whole secure query system from three aspects at least. First, the set $\{(w, VO_w)\}_{w \in W}$ contains plaintext information of all keywords in W. Second, the query keywords of the data user are leaked (i.e., query privacy) when he submits a keyword w of interest to request the corresponding VO_w; Third, the cloud knows the associations between the keywords and verification objects that may allow the cloud server to obtain some useful information about data files (e.g., which data files contain a given query keyword w).

To remedy these drawbacks, in [27], we proposed to encrypt each keyword by a pseudo random function $\widetilde{prf}_k$ and store the verification object set $\{(\widetilde{prf}_k(w), VO_w)\}_{w \in W}$ in a lookup table T. When the data user wishes to obtain VO_w after performing a query using keyword w, he encrypts w as $\widetilde{prf}_k(w)$ under the shared key k and summits the ciphertext to the cloud server. The cloud server returns back the verification object by scanning T according to $\widetilde{prf}_k(w)$ without knowing any underlying plaintext about w. However, we find the construction still leaks some important information including: (1) the submitted verification object request information, for example, whether data users often adopt the same keyword to request verification objects, (2) which verification object is being requested and returned in the current request task, and (3) which verification objects are frequently or rarely requested. These exposed information may become temptations of behaving dishonestly for the dishonest cloud server. For example, if the cloud server knows some verification objects are returned rarely, he may maliciously delete these data for saving storage space.

Therefore, our goal is to prevent the cloud server from obtaining these information by using the Paillier encryption scheme to design thus a secure verification object request mechanism.

Now, we describe our proposed verification object request scheme in detail. First of all, the authorized data user generates the public/private pair $(pk = N, sk)$ for the Paillier encryption system, where N denotes a large composite number. For a certain keyword $w_i \in W$ (assume $|W| = d$) that the data user wants to request the corresponding verification object VO_{w_i} from the cloud server, the data user sets a flag f_i for w_i to be 1 to denote VO_{w_i} to be the desired verification object. For other all keywords $w_j \in W(1 \leq j \leq d, j \neq i)$, he sets the corresponding flag $f_j = 0(1 \leq j \leq d, j \neq i)$. Then, the data user encrypts each flag $f_k(k = 1, 2, ..., d)$ using the Paillier encryption under the public key pk and a randomly-chosen $r_k \in Z_N^*$ as follows:

$$c_k = [(1 + N)^{f_k} \cdot r_k^N \mod N^2]$$

Further, the data user sends the ciphertext set $\{c_k | 1 \leq k \leq d\}$ to the cloud sever. Upon receiving the set, the cloud server computes the following ciphertext for each c_k according to the corresponding verification object VO_{w_k} in $\{VO_w\}_{w \in W}$ stored in the cloud server

$$\begin{aligned} c_k^* = c_k^{VO_{w_k}} &= [(1 + N)^{f_k} \cdot r_k^N \bmod N^2]^{VO_{w_k}} \\ &= [(1 + N)^{f_k \cdot VO_{w_k}} \cdot r_k^{N \cdot VO_{w_k}} \bmod N^2] \\ &= [(1 + N)^{f_k \cdot VO_{w_k}} \cdot (r_k^{VO_{w_k}})^N \bmod N^2] \end{aligned}$$

and further computes:

$$\begin{aligned} c^* &= \left[\prod_{k=1}^{d} c_k^* \mod N^2\right] = \left[\prod_{k=1}^{d} c_k^{VO_{w_k}} \bmod N^2\right] \\ &= \left[(1 + N)^{\sum_{k=1}^{d} f_k \cdot VO_{w_k}} \cdot \left(\prod_{k=1}^{d} r_k^{VO_w}\right)^N \mod N^2\right] \\ &= \left[(1 + N)^{1 \cdot VO_{w_i}} \cdot \left(\prod_{k=1}^{d} r_k^{VO_w}\right)^N \mod N^2\right] \\ &= \left[(1 + N)^{VO_{w_i}} \cdot \left(\prod_{k=1}^{d} r_k^{VO_w}\right)^N \mod N^2\right] \end{aligned}$$

After finishing the above calculation, the cloud server sends c^* to the data user. Obviously, c^* is an effective Paillier encryption of the message VO_{w_i} under the public key $pk = N$ and the random number $\prod_{k=1}^{d}(r_k^{VO_w})$. Therefore, the data user can successfully decrypt c^* with the owned private key sk to gain the objective verification object VO_{w_i} locally. During the whole process, the cloud server knows nothing about which verification object is requested and which verification object is actually returned every time due to the undeterministic Paillier encryption.

In addition, to verify the authenticity of returned verification object itself, the corresponding signature should also be returned together. A common way is to attach the signature to its verification object by $\sigma || VO$, where $||$ denotes a string concatenation notation. Thus, the verification object set can be denoted as $\{(\sigma || VO)_w\}_{w \in W}$. Algorithm 4 shows the algorithm of securely obtaining a verification object and we use E_{pk} and D_{sk} to denote the Paillier encryption and decryption respectively, for simplicity.

Algorithm 4 Securely Obtaining Verification Object

```
Input:
    The keyword set $W, |W| = d$ and corresponding verification object set $\{(\sigma||VO)_w\}_{w \in W}$, the submitted keyword $w_i$.
Output:
    The objective verification object $(\sigma||VO)_{w_i}$
1:  Generate the public/private pair $(pk, sk)$
2:  for $k \in [1, d]$ do
3:      if $w_i = w_k$ then
4:          Set $f_k = 1$ and encrypt $f_k$ as $E_{pk}(f_k)$
5:      else
6:          Set $f_k = 0$ and encrypt $f_k$ as $E_{pk}(f_k)$
7:      end if
8:  end for
9:  for each $k \in [1, d]$ do
10:     Calculate $c_k = E_{pk}(f_k)^{(\sigma||VO)_{w_k}}$
11: end for
12: Calculate $c^* = \prod_{k=1}^{d} c_k$
13: Decrypt $c^*$ as $VO_{w_i} = D_{sk}(c^*)$
14: return $(\sigma||VO)_{w_i}$.
```

8 Security Analysis

8.1 Security of Verification Object

Similar to the secure index semantic security [5], the security of the verification object aims to capture the notion that the verification object reveals nothing about contents of data files. A more formal and rigorous security definition is the verification object indistinguishability, which is synoptically described as that, given two verification objects $VO_w, VO_{w'}$ of set $C_w, C_{w'}$ for two different keywords w, w', no polynomial-time adversary $\mathcal{A}$ can determine which verification object is for which data file set with probability that is non-negligible greater than 1/2. We use formulation of verification object indistinguishability to prove the semantic security of our scheme and formally use the following game to define the formulation.

The verification object indistinguishability game ($\widehat{Game}$)**:** Let f denote the verification object construction algorithm described in **Algorithm 1**. There is a challenger $\mathcal{B}$ and a probabilistic polynomial time adversary $\mathcal{A}$ in this game.

1) $\mathcal{A}$ asks $\mathcal{B}$ for the output of f for his submitted challenging set C_w for different keyword w many times.
2) $\mathcal{A}$ sends two different set C_{w_0} and C_{w_1} to $\mathcal{B}$, which are not challenged in setp 1. $\mathcal{A}$ continues to ask $\mathcal{B}$ for the output of f for C_w. The only restriction is that C_w is not C_{w_0} or C_{w_1}.
3) After receiving C_{w_0} and C_{w_1}, $\mathcal{B}$ chooses a bit $b \in \{0,1\}$ with probability 1/2 and invokes $f(C_{w_b})$ to output the verification object VO_{w_b} for C_{w_b}.
4) VO_{w_b} is sent to $\mathcal{A}$, $\mathcal{A}$ outputs his guess b' of b. If $b = b'$, the game input 1, and 0 otherwise. We say $\mathcal{A}$ wins the game and succeeds if the input is 1.

Intuitively, if the verification object is indistinguishable, the probability of $\mathcal{A}$ wins the game is at most negligibly greater that 1/2, since it is easy to succeed with probability 1/2 by taking a random guess to input b'. Before proving the security of the verification objects, we first give the following two definitions.

Definition 1. *The advantage of the probabilistic polynomial time $\mathcal{A}$ in winning the above game is defined as:*

$$Adv_{\mathcal{A}}^{f} = |Pr[b = b'] - \frac{1}{2}|$$

If $Adv_{\mathcal{A}}^{f}$ is negligible, we say that our constructed verification object is semantically secure and achieves indistinguishability.

Definition 2. *Given a computationally efficient and keyed function F: $\{0,1\}^* \times \{0,1\}^{\tau} \rightarrow \{0,1\}^s$, for any probabilistic polynomial time distinguisher $\mathcal{D}$ and an arbitrary length string $\boldsymbol{x} \in \{0,1\}^*$, the advantage that $\mathcal{D}$ distinguishes $F_k(\boldsymbol{x})$ from a random string $\boldsymbol{r}$ of length s is defined as:*

$$Adv_{\mathcal{D}}^{F} = |Pr[\mathcal{D}(r) = 1] - Pr[\mathcal{D}(F_k(x)) = 1]|$$

where $k \in \{0,1\}^{\tau}$ and $\boldsymbol{r}$ is chosen at random uniformly from $\{0,1\}^s$ (the output of a random function). If F is a pseudo-random function, then the advantage $Adv_{\mathcal{A}}^{F}$ is negligible under the randomly chosen key k from $\{0,1\}^{\tau}$.

Definition 2 means that no polynomial time algorithm can distinguish the output of a pseudo-random function from the output of a real random function [31]. According to the construction of verification objects, it is easy to see that achieving the indistinguishability of verification objects needs to guarantee that the positions in the Bloom Filter of the inserted elements are indistinguishable, which can further reduce to guarantee the indistinguishability of inserted elements (for a set C_w, the inserted elements include data files and padding elements). In **Algorithm 1**, we use the pseudo-random function guarantee the indistinguishability of inserted elements. We give the formal security proof as follows.

Theorem 1. *If prf is a pseudo-random function, then our constructed verification object is semantically secure and achieves indistinguishability in the random oracle model.*

Proof. Suppose the adversary $\mathcal{A}$ has a non-negligible advantage ϵ ($\epsilon < 1$) to win $\widehat{Game}$, we can use $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$ who can distinguish the output of the pseudo-random function from the output of a real random function with a non-negligible advantage.

According to the **Algorithm 1** denoted as f_0, we construct another algorithm **Algorithm 1*** denoted as f_1. The only difference between them is that **Algorithm 1*** uses a random function $rrf : \{0,1\}^* \rightarrow \{0,1\}^s$ to replace the pseudo-random function prf_k used in f_0. Essentially, prf_k and rrf are modeled the random oracles that $\mathcal{D}$ has access to. $\mathcal{D}$ accepts the algorithm $f_x, x \in \{0,1\}$ and its goal is to determine whether $x = 0$ or whether $x = 1$. To do this, $\mathcal{D}$ emulates the game $\widehat{Game}$ for $\mathcal{A}$, and observes whether $\mathcal{A}$ succeeds or not. If $\mathcal{A}$ succeeds then $\mathcal{D}$ determines $x = 0$; otherwise, $x = 1$.

$\mathcal{D}$ is given the algorithm f_x and $\mathcal{A}$ chooses two file sets C_{w_1} and C_{w_2}. $\mathcal{D}$ randomly chooses a bit $b \in \{0,1\}$ with probability $\frac{1}{2}$ and invokes $f_x(C_{w_b})$ to output VO_{w_b} and then sends VO_{w_b} to $\mathcal{A}$. $\mathcal{A}$ outputs a bit b' and $\mathcal{D}$ outputs a guess x' for x. Since $\mathcal{A}$ has a non-negligible advantage ϵ to succeed in the game $\widehat{Game}$ (i.e., the output b' satisfies $b' = b$), correspondingly, $\mathcal{D}$ also has advantage ϵ to determine the guess $x' = x = 0$. That is, $\mathcal{D}$ can distinguish the output VO_{w_b} of $f_0(C_{w_b})$ (using the pseudo-random function prf_k under the key k) from $f_1(C_{w_b})$ (using the real random function rrf) with the non-negligible advantage ϵ. Since VO_{w_b} contains $|C_w|_{max}$ (let $|C_w|_{max} = \alpha$) elements after padding , for each element c, assume that the advantage that $\mathcal{D}$ distinguishes $prf_k(c)$ from $rrf(c)$ is ϵ', we have:

$$\underbrace{\epsilon' \cdot \epsilon' ... \cdot \epsilon'}_{\alpha} = \epsilon \Rightarrow \epsilon' = \sqrt[\alpha]{\epsilon}$$

Since ϵ is non-negligible, the advantage $\sqrt[\alpha]{\epsilon}$ is also non-negligible. Therefore, if $\mathcal{A}$ wins the game $\widehat{Game}$ with a non-negligible advantage ϵ, then $\mathcal{D}$ distinguishes the output of the pseudo-random function prf_k from the output of the real random function rrf with the non-negligible advantage $\sqrt[\alpha]{\epsilon}$, which contradicts the **Definition 2**. □

8.2 Unforgeability of Verification Object Signature

To guarantee the authenticity of the verification objects themselves, a short signature scheme is proposed based on [34] and [36]. We follow the threat model and security definition of certificateless signature scheme by [36] and omit the unforgeability proof of our scheme due to space limitations. Please refer to [36] for detail security proofs.

8.3 Security of Verification Object Request

We design a secure verification object requesting technique by adopting Paillier encryption. During the whole verification object request process, the cloud server knows nothing about which verification object is requested and which verification object is actually returned since the Paillier encryption is a probabilistic public-key encryption scheme.

9 Experimental Evaluation

We conduct experiments to evaluate the performance of our scheme from four aspects: verification object construction and query results verification, verification object signature and authentication, verification information request generation, and verification accuracy.

9.1 Experiment Setup

To evaluate the verification object construction time and query results verification time, we generate 5 text file sets $F_{hardware}$ and $|F_{hardware}| = 200$, $F_{machine}$ and $|F_{mathine}| = 400$, $F_{subject}$ and $|F_{subject}| = 600$, $F_{protocal}$ and $|F_{protocal}| = 800$, $F_{network}$ and $|F_{network}| = 1000$, respectively. For example, $F_{hardware}$ denotes a set of text files containing the keyword *hardware* with the cardinality 200. All these text files are randomly picked up from the real data set RFC (Request For Comments Database) [41]. We encrypt the 5 file sets using AES to get their corresponding ciphertext set $C_{hardware}, C_{machine}, C_{subject}, C_{protocal}, C_{network}$. Obviously, $|C_{network}|_{max} = |C_{network}| = 1000$.

Recall that each verification object is composed of a Counting Bloom Filter and a random elements pad region, if we set the number of hash functions to be $l = \log_{\frac{1}{2}} 0.01 = 7$ and the number of counters in Counting Bloom Filter to be $m = 1000 \log_{0.6185} 0.01 = 1000 \times 9.585 = 9585$, then the false positive is less than 0.01. We expand the Counting Bloom Filter from $m = 9585$ to $n = 12085$, the last 2500 counters are regarded as pad region. Thus, the size of each verification object is about 6KB ($12085 \times 4 = 48340$ bits). In addition, we use HMAC-MD5 with a 128 bits key to instantiate the pseudo random hash function $prf_k()$.

We implement our verification object signature and authentication scheme based on Java library of the Pairing-Based Cryptography Library (JPBC) [42] and choose Type A elliptic curve group with 160-bit prime order, which can achieve 1024-bit discrete log security. To implement secure verification object request, we use Pailier Encryption for request information encryption, the secret key is set to be 512 bits.

In our experiments, we use Java language to implement all programs. The client side is an Inter Core i5-6200U 2.3GHz computer with 4GB RAM running windows 7. The cloud environment is simulated by using the Dell blade M610 running Linux Centos5.8 OS, which has 4 processor cores and supports 8 parallel threads.

9.2 Performance of Verification Object Construction and Query Results Verification

Fig. 4(a) shows the time cost of verification object generation for different data files set with different number of data files, we can observe that the size of different data files set has little influence on the time cost of verification object generation. For example, the time cost of generating $VO_{hardware}$ (94ms) and the time cost of

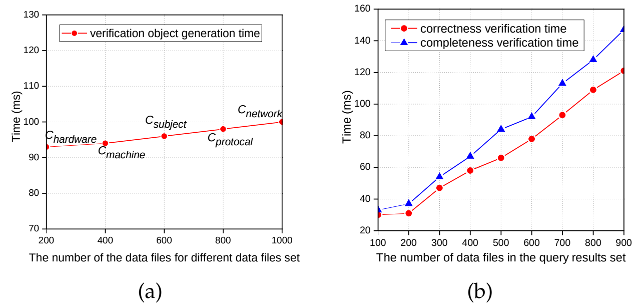


Fig. 4. (a) The time cost of verification object generation with different set of data files containing a certain keyword. (b) The time cost of correctness and completeness verification with different number of data files in a query results set.

generating $VO_{network}$ (99ms) is almost equal, though the size of $C_{network}$ is five times as large as that of $C_{hardware}$. The reason is that, in our scheme, generating a verification object is mainly determined by HMAC-MD5 operations, which are involved in both data file insertions and pads. For each data files set, the less the number of data files in the set is, the more random elements is needed to pad for constructing its verification object. Thus the total number of HAMC-MD5 operations will keep the same for each set of data files containing a certain keyword. In addition, the execution time is almost zero of 7 hash functions in $\mathcal{H}$ when hashing a small string of length 128 bits. In experiments, the random pad elements are also picked up from RFC randomly for each verification object.

Fig. 4(b) shows the time cost of query results correctness and completeness verification. In experiment, due to without considering the secure query scheme, we artificially formulate 9 different query results sets $R_{network}$s with increasing number of data files and step length 100 and use the constructed verification object $VO_{network}$ to verify their correctness and completeness. We can observe that the time cost of correctness and completeness verification is linearly increasing with the increase of the number of data files in the query results set and the completeness verification needs to consume a little more time than correctness verification due to deletion operations of elements in Bloom Filter.

9.3 Performance of Verification Object Signature and Authentication

In this section, we estimate the time cost of the verification object authentication. The time cost of all operations used for running the experiments based on the JPBC library and our software/hardware setting is shown in Table 3.

Table 4 shows the time cost of signature and authentication for verification object. We can see that a data owner needs to consume about 53ms to perform a signature on each verification object and a data user needs about

TABLE 3
Time Cost of Operation

Notations	Descriptions	Time Cost (ms)
$E_{\mathbb{G}_1}$	exponentiation operation in $\mathbb{G}_1$	≈ 42
$M_{\mathbb{G}_1}$	multiplication operation in $\mathbb{G}_1$	< 1
$A_{\mathbb{Z}_q}$	addition operation in $\mathbb{Z}_q$	< 1
H_2	hash function $\{0,1\}^* \times \mathbb{G}_1 \to \mathbb{Z}_q^*$	≈ 11
P	pairing operation	≈ 46

TABLE 4
Time Cost for Each Verification Object

	Cost	time (ms)
Signature	$H_2 + A_{\mathbb{Z}_q} + E_{\mathbb{G}_1}$	53
Authentication	$H_2 + E_{\mathbb{G}_1} + M_{\mathbb{G}_1} + 2P$	145

145ms to finish the verification object authentication.

9.4 Performance of Verification Object Request

Recall that a data user securely obtains the desired verification object by three steps, during the whole process, the Paillier encryption is used. First, the data user encrypts request information according to the requested keyword and keywords dictionary W. Second, the cloud server generates encrypted desired verification object according to submitted request information and the outsourced verification objects set $\{(\sigma||VO)_w\}_{w\in W}$. Third, the data user decrypts the encrypted desired verification object returned by the cloud server. Table 5 shows the time cost of all encryption/decryption operations used for implementing our scheme, where E, D denotes a Paillier encryption and a Paillier decryption, respectively, and x denotes a verification object.

Fig. 5(a) shows the time cost of verification object request information generation for the data user, which is linearly increasing with the increase of the number of keywords in W. In this process, $|W|-1$ operations of $E(0)$ and one $E(1)$ operation are involved, when varying the size of the dictionary from 10 to 100 with step length 10, the time cost changes from about 70ms to about 700ms, correspondingly. Fig. 5(b) shows the time cost of decrypting a desired verification object for data user. The process only includes one $D(x)$ operation, which generally consumes 15ms in our experiments. We can observe that the size of keyword dictionary W has no influence on time cost of the desired verification object decryption.

Fig. 6 shows the time cost of generating encrypted desired verification object for the cloud server, which is linearly increasing with the increase of the dictionary size (Fig. 6(a)) and is not affected by the number of data files contained in the verification object when fixing the dictionary size (Fig. 6(b)). In the encryption process, $|W|-1$ operations of $E(0)^x$, one $E(1)^x$ operation, and $|W|-1$ operations of $E(x)\cdot E(x)$ are involved in the cloud server side. When setting the number of threads to be 1 and varying the size of the dictionary from 10 to 100 with step length 10, the time cost changes from about 122ms to about 1.25s. We also observe that the time cost can be reduced remarkably by increasing the number of threads when the programs are run in a parallel computing environment.

TABLE 5
Time Cost of Operation

Notations	Descriptions	Cost (ms)
$E(1), E(0)$	Paillier encryption of plaintext 1 and 0	≈ 7
$E(1)^x, E(0)^x$	exponentiation operation of ciphertext	≈ 12
$E(x)\cdot E(x)$	multiplication operation of ciphertext	< 1
$D(x)$	Paillier decryption	≈ 16

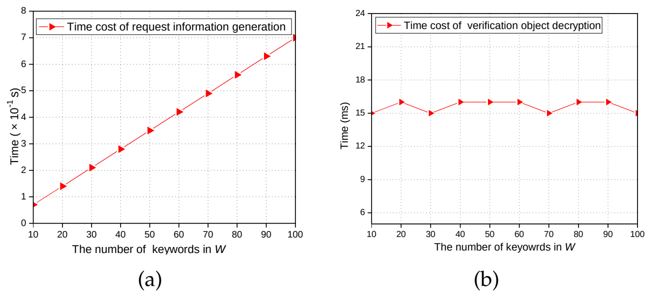


Fig. 5. (a) The time cost of verification object request information generation with different number of keywords in the keyword dictionary W. (b) The time cost of the desired verification object decryption with different number of keywords in the keyword dictionary W.

9.5 Verification Accuracy

Essentially, the constructed verification object is the Bloom Filter, which can incur the false positive. This means that our scheme may cause incorrect query results to pass the correctness verifications. To numerically evaluate the verification accuracy, we first define several verification *Events* in Table 6. Given a returned query results set R_w of the query keyword w and corresponding verification object VO_w, for each result $c \in R_w$, if a verification *Event* occurs, then the *Event* is set as 1; obviously,

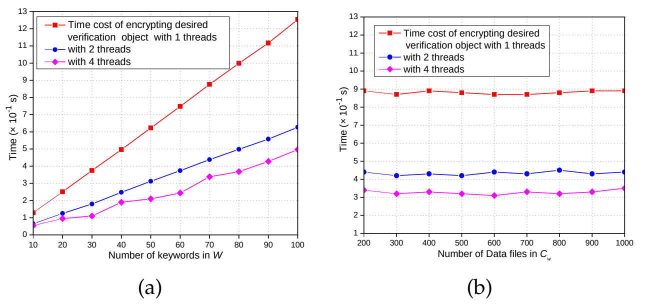
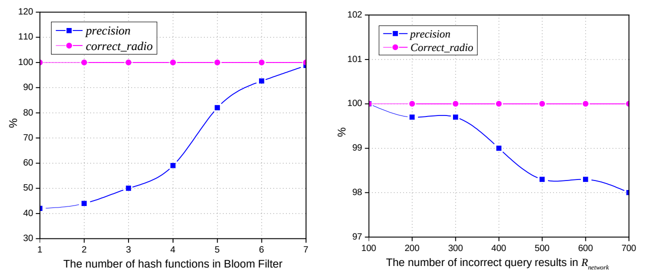


Fig. 6. The time cost of generating encrypted desired verification object for cloud server: (a) for different number of keywords in the keyword dictionary W, and (b) for different number of data files in C_w with the fixed dictionary, $|W| = 70$.

TABLE 6
Events of Result Verification

Event	Description
tp (True Positive)	The correct query result pass verification.
fp (False Positive)	The incorrect query result pass verification.
fn (False Negative)	The correct result does not pass verification.



(a) (b)

Fig. 7. Evaluation of verification accuracy: (a) for different number of hash functions in Bloom Filter, $R_{network}$ contains 300 correct query results and 500 incorrect query results, and (b) for different number of incorrect query results in $R_{network}$ with the fixed hash functions ($l = 7$), $R_{network}$ contains 300 correct query reuslts.

for each result, these events are independent each other. We define the two metrics $correct\_radio = \frac{\sum tp}{\sum tp + \sum fn}$ and $precision = \frac{\sum tp}{\sum tp + \sum fp}$ to evaluate the verification accuracy, where $\sum$ denotes the total number of occurrences of an event when using VO_w to verify R_w. We artificially formulate the query results set $R_{network}$ based on the data files set $C_{network}$ as the experiment data set. Correspondingly, the verification object is $VO_{network}$. Fig. 7 shows the verification accuracy of our scheme.

Fig. 7(a) and Fig. 7(b) show the $correct\_radio$ always keeps 100%. This is because that Bloom Filter does not introduce false negatives which means that our scheme can guarantee all correct results in $R_{network}$ to be able to correctly pass verifications ($\sum fn = 0$ and $\sum tp = 300$). On the other hand, from the Fig. 7(a), we can see that, when the number of the hash functions of Bloom Filter is set as 1, our scheme causes about 420 incorrect results to pass verifications ($\sum fp = 420$) and thus leads to a very low verification $precision = \frac{\sum tp}{\sum tp + \sum fp} = \frac{300}{300+420} = 41.7\%$; $precision$ gradually increases with the increasing number of hash functions and approximatively achieves $\frac{300}{300+4} \approx 98.7\%$ when the number of hash functions is set to the optimal value $l = 7$. Fig. 7(b) demonstrates that though the $precision$ has the trend of decrease with the increases of the number of incorrect query results in $R_{network}$, it is not less than 98% when the number of hash function is set to the optimal value $l = 7$ even $R_{network}$ contains the large number of incorrect results. For the very few incorrect query results that pass verifications, the data user can delete them after decrypting.

10 Conclusion

In this paper, we propose a secure, easily integrated, and fine-grained query results verification scheme for secure search over encrypted cloud data. Different from previous works, our scheme can verify the correctness of each encrypted query result or further accurately find out how many or which qualified data files are returned by the dishonest cloud server. A short signature technique is designed to guarantee the authenticity of verification object itself. Moreover, we design a secure verification object request technique, by which the cloud server knows nothing about which verification object is requested by the data user and actually returned by the cloud server. Performance and accuracy experiments demonstrate the validity and efficiency of our proposed scheme.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have helped to improve the quality of the manuscript. The work is supported by the National Natural Science Foundation of China (61472131, 61300218, and 61472132); The Natural Science Foundation of Hunan Province (2017JJ2292); Science and Technology Key Projects of Hunan Province (2015TP1004, 2016JC2012); Open Research Fund of Hunan Provincial Key Laboratory of Network Investigational Technology (2016WLZC011).

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," http://dx.doi.org/10.602/NIST.SP.800-145.
[2] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.
[3] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Springer RLCPS*, January 2010.
[4] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Sympsoiumon Security and Privacy*, vol. 8, 2000, pp. 44–55.
[5] E.-J.Goh, "Secure indexes," IACR ePrint Cryptography Archive, http://eprint.iacr.org/2003/216, Tech. Rep., 2003.
[6] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public-key encryption with keyword search," in *EUROCRYPR*, 2004, pp. 506–522.
[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved deinitions and efficient constructions," in *ACM CCS*, vol. 19, 2006, pp. 79–88.
[8] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Springer CRYPTO*, 2007.
[9] K. Kurosawa and Y. Ohtaki, "Uc-secure searchable symmetric encryption," *Lecture Notes in Computer Science*, vol. 7397, pp. 258–274, 2012.
[10] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2266–2277, 2013.
[11] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2013, pp. 258–274.
[12] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *IEEE S&P*, May 2014, pp. 639–654.
[13] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *IEEE ICDCS*, 2010, pp. 253–262.

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

[15] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *ACM ASIACCS*, 2013.

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

[17] W. Zhang, S.Xiao, Y. Lin, J. Wu, and S. Zhou, "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1566–1577, May 2016.

[18] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed System*, vol. 27, no. 2, pp. 340–352, 2015.

[19] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Transactions on Communications*, vol. E98-B, no. 1, pp. 190–200, 2015.

[20] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, 2014.

[21] H. Yin, Z. Qin, L. Ou, and K. Li, "A query privacy-enhanced and secure search scheme over encrypted data in cloud computing," *Journal of Computer and System Sciences*, http://dx.doi.org/10.1016/j.jcss.2016.12.003.

[22] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 43–56, 2014.

[23] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[24] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, and Y. T. Hou, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 3025–3035, 2014.

[25] Q. Zheng, S. Xu, and G. Ateniese, "Vabks: Verifiable attribute-based keyword search over outsourced encrypted data," in *IEEE INFOCOM*, May 2014, pp. 522–530.

[26] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *IEEE INFOCOM*, April 2015, pp. 2110–2118.

[27] H. Yin, Z. Qin, L. Ou, Q. Liu, Y. Hu, and H. Rong, "A secure and fine-grained query results verification scheme for private search over encrypted cloud data," in *IEEE ICA3PP*, 2015, pp. 667–681.

[28] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encryption data," in *ACM CCS*, 2006, pp. 89–98.

[29] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 12, no. 7, pp. 422–426, 1970.

[30] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide area web cache sharing protocal," in *ACM SIGCOMM*, 1998, pp. 254–265.

[31] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*. Lecture Notes, 2001.

[32] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Springer CRYPTO*, ser. LNCS 2139, J. Kilian, Ed., 2001, pp. 213–229.

[33] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Springer EUROCRYPT*, 1999, pp. 223–238.

[34] X. Li, K. Chen, and L. Sun, "Certificateless signature and proxy signature schemes from bilinear pairings," *Lithuanian Mathematical Journal*, vol. 45, no. 1, pp. 76–83, 2005.

[35] M. Gorantla and A. Saxena, "An efficient certificateless signature scheme," in *Computational Intelligence and Security, CIS 2005*, 2005, pp. 110–116.

[36] W. Yap, S. Heng, and B. Goi, "An efficient certificateless signature scheme," in *Emerging Directions in Embedded and Ubiquitous Computing, EUC 2006 Workshops*, 2006, pp. 322–331.

[37] H. Xiong and Z. Qin, "Revocable and scalable certificateless remote authentication protocol with anonymity for wireless body area networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1442–1455, 2015.

[38] J. Liu, Z. Zhang, X. Chen, and K. S. Kwak, "Certificateless remote anonymous authentication schemes for wirelessbody area networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 332–342, 2014.

[39] S. Al-Riyami and K. Paterson, "Certificateless public key cryptography," in *Springer ASIACRYPT*, 2003, pp. 452–473.

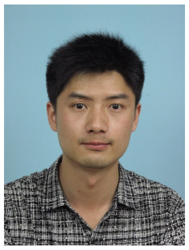
[40] H. Xiong, "Cost-effective scalable and anonymous certificateless remote authentication protocol," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 12, pp. 2327–2339, 2014.

[41] "Rfc, request for comments database," http://www.ietf.org/rfc.html.

[42] http://gas.dia.unisa.it/projects/jpbc/index.html.

Hui Yin received the BS degree from Hunan Normal University and the MS degree from Central South University, both in computer science, in 2002 and 2008, respectively. He is currently pursuing the PhD degree in the College of Computer Science and Electronic Engineering at Hunan University, China. His main interests are information security, privacy protection, cloud computing, and big data processing.

Zheng Qin received the PhD degree in computer software and theory from Chongqin University, China, in 2001. He is a professor of computer science and technology in Hunan University, China. He is a member of China Computer Federation (CCF) and ACM respectively. His main interests are computer network and information security, cloud computing, big data processing and software engineering.



Jixin Zhang received the BS degree from Hubei Polytechnie University in Mathematics, and the MS degree from Wuhan University of Technology in computer science and technology, in 2007 and 2014, respectively. Currently, he is pursuing the PhD degree in the College of Computer Science and Electronic Engineering at Hunan University, China. His research focuses on malware detection, privacy protection and big data.

Lu Ou received the BS degree from Science and Technology of Changsha University in computer science and the MS degree from Hunan University in software engineering, in 2005 and 2012, respectively. Currently, she is pursuing the PhD degree in the College of Computer Science and Electronic Engineering at Hunan University, China. Her research focuses on network security, privacy protection and big data.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 470 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.