Verifiable Searchable Symmetric Encryption from Indistinguishability Obfuscation

Rong Cheng
School of Electronic and Communication Engineering, Shenzhen Polytechnic

Jingbo Yan
Dept. of CSE, University at Buffalo, SUNY

Chaowen Guan
Dept. of CSE, University at Buffalo, SUNY

Fangguo Zhang*
School of Information Science and Technology, Sun Yat-sen University
isszhfg@mail.sysu.edu.cn

Kui Ren
Dept. of CSE, University at Buffalo, SUNY

ABSTRACT

Searchable symmetric encryption(SSE) allows a client to encrypt his data in such a manner that the data can be efficiently searched. SSE has practical application in cloud storage, where a client outsources his encrypted data to a cloud server while maintaining the searchable ability over his data. Most of the current SSE schemes assume that the cloud server is honest-but-curious. However, the cloud may actively cheat on the search process to keep its cost low. In this paper, we focus on the malicious cloud model and propose a new verifiable searchable symmetric encryption scheme. Our scheme is built on the secure indistinguishability obfuscation ($i\mathcal{O}$) and can be considered as the first step to apply $i\mathcal{O}$ in the SSE field. Moreover, our scheme can be easily extended to multiple functionalities, such as conjunctive and boolean queries. Furthermore, it can be extended to realize a publicly verifiable SSE. Thorough analysis shows that our scheme is secure and achieves a better performance.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection

General Terms

Security, Privacy

Keywords

Searchable symmetric encryption, indistinguishability obfuscation, cloud storage, malicious server

*Corresponding author

ASIA CCS'15, April 14–17, 2015, Singapore..

http://dx.doi.org/10.1145/2714576.2714623.

1. INTRODUCTION

With the developments of cloud computing, more and more individuals or IT enterprises are willing to outsource their data to the cloud server. As people's awareness of privacy concern is increasing, sensitive data has to be encrypted before being outsourced to the cloud. Confidentiality of the data is guaranteed by the security of symmetric or public encryption schemes. Unfortunately, this process inevitably raises the problem that the cloud cannot do any effective computations on the encrypted data, specially, making it difficult to implement search on the stored data. Thus how to enable an effective searchable functionality on the encrypted data becomes one of the most significant issues and has attracted much research attention [1].

Fully homomorphic encryption [2] or oblivious RAMs [3] are very powerful tools and can be adopted to realize an ideally secure method to search on the encrypted data. However, they are far too inefficient for practical use. Searchable encryption was first proposed by Song *et al.* in [4]. Generally, they can be divided into two categories, searchable symmetric encryption (SSE) and public key encryption with keyword search (PEKS). However, PEKS is still not efficient due to the complex operation of public key encryption, especially for a large-scale database. Therefore, we will focus on how to construct a secure and efficient SSE.

Since the initial SSE approach by Song *et al.* [4], there have been a lot of researches emerging to enhance the SSE. In functionality, a lot of works[5, 6, 7, 8, 9] are proposed to fulfill conjunctive search, dynamic search, ranked search, fuzzy search and boolean queries, *et al*. Moreover, in the security models of most existing SSE constructions, servers are assumed to be honest-but-curious. They will honestly execute the search protocol and only attempt to infer the sensitive information about the clients. However, in real world, this is not always the case. If a server does not honestly execute the search operation, we call it a **malicious server**. In [10], Chai *et al.* considered a semi-honest-but-curious cloud servers, which execute only a fraction of search operations honestly and return a fraction of correct search outcome. In [11], Kurosawa *et al.* first studied the security model where the server is malicious. They proposed a verifiable searchable symmetric encryption scheme, in which

they use the message authenticate code (MAC) to protect the reliability of the searching results. Their scheme is proved secure in the universally composable security framework, *i.e.* UC-secure. However their verifiable SSE requires a search time linear in the number of data files.

To our delight, we find a very powerful cryptographic primitive named indistinguishability obfuscation ($i\mathcal{O}$), which can be used to build verifiable SSE scheme. Barak *et al.* [12] proposed the concept *Indistinguishability Obfuscation* which requires that obfuscations of any two equal-size programs that compute the same function are computationally indistinguishable.

The $i\mathcal{O}$ becomes so important because recently there is a breakthrough result of Garg *et al.* [13] that puts forward the first candidate construction for an efficient $i\mathcal{O}$ for general boolean circuits. Many surprising applications in cryptographic field were proposed such as multi-party key exchange [14], deniable encryption [15], short signature with fast signing [16], full domain Hash [17], *etc.* This fascinating achievement provides the possibility for us to adopt $i\mathcal{O}$ in our approach.

Our Work. In this paper we pioneer the idea of adopting indistinguishability obfuscation ($i\mathcal{O}$) to design a new SSE scheme. Our $i\mathcal{O}$-based SSE scheme can have various searching functionalities embedded into the obfuscation circuit so that it can meet flexible searching requirements. Moreover, it can be extended to realize public verifiability on the results returned by the malicious server.

The remainder of this paper is organized as follows. In section 2, we briefly introduce the tools used in our construction. Then we give the system model and security definition in section 3. In section 4, we describe the detailed construction of our SSE scheme and its extensions to multiple search functionalities and public verifiability. Then we give out the analysis for our construction in section 5. Lastly we give the conclusion in section 6.

2. PRELIMINARIES

In this part we will simply explain the cryptographic tools used in our construction: indistinguishability obfuscation and constrained pseudorandom functions.

Indistinguishability Obfuscation. The definition of indistinguishability obfuscation ($i\mathcal{O}$) was first proposed by Barak *et al.* [12]. Informally, the requirement for the indistinguishability obfuscation is that for two programs with the same functionality, after the obfuscation, they will be computationally indistinguishable with each other. In Crypto 2013, the breakthrough result of Garg *et al.*[13] proposed the first efficient indistinguishability obfuscation for general polynomial circuits. All the polynomial circuits can be securely obfuscated. The detailed definition of indistinguishability obfuscation can be referred to [13].

Constrained Pseudorandom Functions. Constrained PRF is a pseudorandom function (PRF) which is only defined on a subset of the usual input space. There are many constructions for simple types of constrained PRFs[18, 19, 20]. In our construction, we use a punctured PRF which is a constrained PRF that can be constrained on the complement of any polynomial size set $\mathcal{G} \subseteq \mathcal{X}$. Following Boneh and Waters [18], we define punctured pseudorandom functions as a PRF with the following added functionality: PRF.Puncture(G) for subset $\mathcal{G} \subseteq \mathcal{X}$ outputs an efficient program for the function:

$$PRF^{\bar{G}}(x) = \begin{cases} PRF(x), & if \ \ x \notin G; \\ \perp, & if \ \ x \in G. \end{cases}$$

3. SECURITY MODEL

The definition of our $i\mathcal{O}$-based SSE is as follows.

Definition 1. ($i\mathcal{O}$-based Searchable Symmetric Encryption). A $i\mathcal{O}$-based SSE scheme is a tuple of six polynomial-time algorithms $\Pi = (Gen, Enc, Token, Search, Verify, Dec)$

- $K \leftarrow Gen(1^k)$: take the security parameter k as input and the client outputs the secret keys K.
- $(i\mathcal{O}_{\mathcal{F}}, \mathcal{C}) \leftarrow Enc(K, \mathcal{W}, \mathcal{F})$: take the secret key K, data file collection $\mathcal{F}$ and keywords set $\mathcal{W}$ as inputs, client outputs the searching circuit $i\mathcal{O}_{\mathcal{F}}$ and ciphertext of $\mathcal{F}$;
- $t_w \leftarrow Token(K, w)$: take as inputs the secret key K and a keyword w, client generates the corresponding search token t_w.
- $(\mathcal{C}_w, \mathcal{S}_w, Mac) \leftarrow Search(\mathcal{C}, i\mathcal{O}_{\mathcal{F}}, t_w)$: take the searching circuit $i\mathcal{O}_{\mathcal{F}}$, the search token t_w and a sequence of encrypted files $\mathcal{C}$ as inputs, cloud server outputs the searched identifiers $\mathcal{S}_w$, searched files $\mathcal{C}_w$ and a verification message Mac.
- $Bool \leftarrow Verify(\mathcal{S}_w, \mathcal{C}_w, Mac, K)$: take the searched identifiers $\mathcal{S}_w$, searched files $\mathcal{C}_w$, MAC value Mac and secret key K as inputs, client outputs $Bool$, if accept, $Bool = 1$; otherwise, $Bool = 0$.
- $\mathcal{Q}_w \leftarrow Dec(K, \mathcal{C}_w, Bool)$: take the secret key K, feedback encrypted file $\mathcal{C}_w$, and $Bool$ value as inputs, abort if $Bool = 0$, else client decrypts to get the file $\mathcal{Q}_w$.

Correctness. We require that cloud server can output the correct searching result with the token and the obfuscated searching circuit.

Our privacy definition adopts the real/ideal simulation paradigm which is popular in the recent SSE works [21, 6, 22]. We require that the adversary cannot distinguish the real game $Real_A^{\Pi}(k)$ and a simulation game $Idea_{A,S}^{\Pi}(k)$(L is the leakage function).

$Real_A^{\Pi}(k)$: *Challenger runs* $K \leftarrow Gen(1^k)$ *to get secret keys* K. *Adversary* $A(1^k)$ *chooses data files* $\mathcal{F}$. *Then challenger runs* $(i\mathcal{O}_{\mathcal{F}}, \mathcal{C}) \leftarrow Enc(K, \mathcal{W}, \mathcal{F})$ *and and sends* $i\mathcal{O}_{\mathcal{F}}$ *and* $\mathcal{C}$ *to* A. *Then adversary makes polynomial number of queries by picking different keywords* w_i, *and the challenger responds the respective search token* $t_{w_i} \leftarrow Token(K, w_i)$. *Finally,* A *returns a bit* b *that is output by the experiment.*

$Idea_{A,S}^{\Pi}(k)$: *Adversary* $A(1^k)$ *chooses data files* $\mathcal{F}$. *Then simulator* S *generates* $(i\mathcal{O}_{\mathcal{F}}, \mathcal{C}) \leftarrow S(L(\mathcal{F}))$ *and sends* $i\mathcal{O}_{\mathcal{F}}$ *and* $\mathcal{C}$ *to* A. *Then adversary makes polynomial number of queries by picking different keywords* w_i. *To respond, simulator generators respective search tokens* $t_{w_i} \leftarrow S(L(\mathcal{F}))$. *Finally,* A *returns a bit* b *that is output by the experiment.*

Verifiability. In our security model, a malicious server may not honestly execute the search operation and return the invalid searching results. The invalid searching results include two cases: the malicious server forges the searching results or deletes some of them. The verifiability requires that the malicious server cannot generate an invalid response which results that the *Verify* algorithm outputs 1.

4. SSE BASED ON IO

In this section, we start with the construction for the basic single keyword search. Then we show how to easily extend it to a conjunctive and boolean keyword. Furthermore, we present how to use $i\mathcal{O}$ to realize the publicly verifiable property for our SSE scheme.

Before presenting our construction, we analyze two trivial SSE constructions using $i\mathcal{O}$: (I) the obfuscated search program takes as inputs one search token and an encryption of the entire database; (II) the obfuscated search program takes as inputs one keyword token and an encryption of one file. For more explicit comparison, we use an concrete example where a client encrypts 100 files, each of which has size 20MB ($20 \cdot 2^{23}$ bits), and then uploads those corresponding 100 ciphertexts to the server, to which the client will make keyword search request in the future. The number of valid keywords is 1000. Let n be the number of files, λ be the size of a file and m be the number of valid keywords.

In construction (I), the search program first decrypts the entire database, and then checks which files have the matching keyword and outputs a corresponding ciphertext of those files. Thus, the circuit that needs to be obfuscated has size $O(poly(n\lambda)) = O(poly(100 \cdot 20 \cdot 2^{23})$, where $poly(x)$ represents a polynomial in terms of x, due to the fact that the cost for the entire database's decryption dominates over searching's. This will lead to a significantly larger obfuscation of search program. In construction (II), the search program decrypts the input file, and then checks whether this file has the matching keyword and outputs "yes/no" correspondingly. The circuit being obfuscated has size $O(poly(\lambda)) = O(poly(20 \cdot 2^{23}))$. Note that it is still a huge-size circuit in $i\mathcal{O}$'s applications. Also observe that, to search for the files associated with one keyword token provided by user, the server needs to execute this obfuscated search program 100 times. Plus, this approach of searching over the files separately could leak additional information, like search pattern.

While in our construction, we first generate an index table and then apply $i\mathcal{O}$. Thus, the circuit that will be obfuscated would has size $O(poly(mn)) = O(poly(1000 \cdot 100))$. Clearly, $m \ll \lambda$ (it is very common, in real life applications, that the number of valid keywords is much smaller than the file's size), which means that $O(poly(mn)) \ll O(poly(n\lambda))$. Additionally, $O(poly(mn)) \ll O(poly(\lambda))$ in this case. Thus, the size of this circuit is significantly smaller than those in the above two trivial constructions, which results in a significantly smaller obfuscated search program. Also note that for each keyword search request, the server just needs to run this search program once, and this obfuscated search program is computed only once and can be repeatedly used in the future. As we can see, although $i\mathcal{O}$ is a powerful primitive, it is not trivial to exploit it.

4.1 The Basic SSE Scheme

There are two parties involved in our construction: client and cloud server, and the whole process can be divided into two phases: setup and retrieval. Let k be the security parameter that will be used in $Gen(\cdot)$. Let $\mathcal{E}$ be a IND-CPA secure symmetric encryption scheme. Let $H(\cdot,\cdot)$ be a collision resistant hash function and $PRF(\cdot,\cdot)$ be a punctured pseudo-random function.

SETUP PHASE

In the setup phase, the client will upload the encrypted data files along with the customized $i\mathcal{O}$ circuit to the cloud server. This phase includes two algorithms, *i.e.* Gen, Enc. The details of two algorithms is shown as follows.

- Gen : The client initiates the scheme by calling $Gen(1^k)$ and generates three random secret keys $k_1, k_2, k_3 \xleftarrow{R} \{0,1\}^k$. k_1 is used to encrypt his data files, k_2 is used to generate searching tokens and k_3 is used to generate verification messages.
- Enc : In this algorithm, the client has two main missions to accomplish. He needs to generate the $i\mathcal{O}$ circute that the server can excute, and encrypt his data files. We denote his data files by $DB = \{(\mathrm{id}(F_i), F_i)\}$, $i = 1 \cdots n$, here each file F_i has an location identifier $\mathrm{id}(F_i)$. All the keywords contained in the files forms the set $\mathcal{W} = (w_1, w_2, ..., w_m)$.
  1. First the client encrypts his database DB using his secret key k_1, that is
  $$C_i = \mathcal{E}.Enc(k_1, F_i) \| h_{mac}.$$
  $$h_{mac} = H(k_1, \mathrm{id}(F_i) \| \mathcal{E}.Enc(k_1, F_i))$$
  Then the encrypted database is
  $$EDB = \{(\mathrm{id}(F_i), C_i)\}, i \in \{1 \cdots n\}.$$
  2. The client generates the look-up table $\mathcal{T}$ for his database. The look-up table is generated according to the relationship between the keywords and data files containing the keywords. $\mathcal{T}$ contains t items $\mathcal{T}_{w_i} = (t_{w_i}, \mathcal{S}_{w_i})$, $i = 1 \cdots m$, where t_{w_i} is the secret token for certain keyword w_i and $\mathcal{S}_{w_i}$ denotes the identifiers' set for files including w_i. Token t_{w_i} is calculated by a hash function H with the key k_2, $t_{w_i} = H(w_j, k_2)$.
  3. The client generates a searching circuit $\mathcal{P}_{search}$ according the look-up table generated in step 2. He can customize $\mathcal{P}_{search}$ in accordance with the required functionality. The procedure of the seach circuit P_{search} for single keyword is demonstrated in Algorithm 1. Then he securely obfuscate it to get the indistinguishability obfuscation circuit $i\mathcal{O}(\mathcal{P}_{search})$.
  4. Data owner sends the encrypted database EDB along with the indistinguishability obfuscation circuit $i\mathcal{O}(P_{search})$ to the cloud server.

Algorithm 1 Search Circuit $\mathcal{P}_{search}$

- *Constants*: Secret key k_3. Look-up table $\mathcal{T}$.
- *Inputs*: searching token t_w.
- *Outputs*:
  1. For $j = 1 \cdots t$, verify that $t_{w_j} \stackrel{?}{=} t_w$. If yes, set $S = S_j$; if no keyword matches, abort.
  2. Compute verifying hash value
  $$Mac = PRF(S \| t_w, k_3).$$
  3. Output the the searching results (S, Mac).

RETRIEVAL PHASE

In the retrieval phase, the client wants to search data files containing certain keyword. He generates the searching token corresponding to the keyword and sends the token to the cloud server. With the searching token, cloud server can

utilize the obfuscation circuit to search for the result and generate the verification message. Then server sends back the result to the client, the client can verify the correctness of the searching results and decrypt the correct results to get the files. This phase has 4 algorithms *Token, Search, Verify* and *Dec*, which are described as follows.

- $Token$: Input a keyword $w \in \mathcal{W}$, the client computes the searching token by $t_w = H(w, k_2)$ and sends it to server;
- $Search$: Cloud server puts the token t_w into the indistinguishability obfuscation circuit $i\mathcal{O}(\mathcal{P}_{search})$ and obtains the searching result $(\mathcal{S}_w, Mac)$. According to identifier set S_w, it finds the corresponding items in EDB. At last, cloud server responds to client's search query by $(\mathcal{C}_w, \mathcal{S}_w, Mac)$.
- $Verify$: The client first verifies the hash value by
$$PRF(\mathcal{S}_w \| t_w, k_3) \stackrel{?}{=} Mac$$
Then if the result passes this verification, the client still needs to make sure the identifier set $\mathcal{S}_w$ is correctly mapped to the ciphertext $\mathcal{C}_w$, he verifies it by:
$$H(k_1, \mathcal{S}_w^i \| \mathcal{E}.Enc(k_1, F_i)) \stackrel{?}{=} \mathcal{C}_w^i.h_{mac}$$
If both of the verifications are passed, the Bool value will be set 1, otherwise it is 0.
- Dec : If the Bool value is 1, the client believes the results are correct and decrypts all files in $\mathcal{C}_w$ by secret key k_1. That is:
$$\mathcal{Q}_w = \mathcal{E}.Dec(k_1, \mathcal{C}_w).$$
If the Bool value is 0, the client will output $fail$.

Correctness. The functionality property of indistinguishability obfuscation gurantees that the search protocol will return the correct results. Besides, if the cloud server is not honest, the security of the verification can detect the cheating behavior.

4.2 Extensions to Other Functionalities

Our SSE scheme can be easily extended to schemes which support multiple functionalities, *e.g.* conjunctive queries and boolean queries. If we treat the $i\mathcal{O}$ circuit as a blackbox, it is easy to see that other parts of the basic framework is independent with the search functionalities. The client will only need to slightly modify the construction of the search circuit $\mathcal{P}_{search}$ and its corresponding $i\mathcal{O}$ circuit. In $\mathcal{P}_{search}$ of original SSE scheme which supports single keyword search, the searched identifier set $\mathcal{S}_w$ can be directly gotten after a one-round mapping over the look-up table. In order to support for conjunctive and boolean queries, we need to add a step to operate conjunctive or boolean function on the look-up table $\mathcal{T}$. The algorithms of conjunction keyword search and boolean queries are shown in Algorithm 2 and 3.

Moreover, our proposed SSE scheme can be further extended to achieve the public verifiability, *i.e.* the ability to securely delegate the verification to a third party. A public verification circuit $\mathcal{P}_{pv}$ can be designed with the verifying key embedded in. The client can generate the indistinguishability obfuscation of verification circuit $i\mathcal{O}(\mathcal{P}_{pv})$. Anyone with the indistinguishability obfuscation $i\mathcal{O}(\mathcal{P}_{pv})$ can verify the server's searching results. The verification circuit $\mathcal{P}_{pv}$ is shown in Algorithm 4.

Algorithm 2 Search Circuit $\mathcal{P}_{CSearch}$

- *Constants*: Secret key k_3. Look-up table $\mathcal{T}$.
- *Inputs*: searching token $t_{w_x}, t_{w_y}, t_{w_z}$.
- *Outputs*:
  1. For $j = 1 \cdots t$, verify that $t_{w_j} \stackrel{?}{=} t_{w_x}$. If yes, set $S_x = S_j$; if no keyword matches, abort. Similarly, obtain S_y and S_z.
  2. Compute searched index set $S = S_x \wedge S_y \wedge S_z$.
  3. Compute verifying hash value
  $$Mac = PRF(S \| t_{w_x} \| t_{w_y} \| t_{w_z}, k_3).$$
  4. Output the the searching results (S, Mac).

Algorithm 3 Search Circuit $\mathcal{P}_{BSearch}$

- *Constants*: Secret key k_3. Look-up table $\mathcal{T}$. Boolean Function BF.
- *Inputs*: searching token $S_t = \{t_{w_i}\}$, $i \in [1, t]$.
- *Outputs*:
  1. For $j = 1 \cdots t$, verify that $t_{w_j} \stackrel{?}{=} t_{w_1}$. If yes, set $S_1 = S_j$; if no keyword matches, abort. Similarly, obtain $S_2, S_3, \cdots, S_t$.
  2. Compute $S = BF(S_1, S_2, \cdots, S_t)$.
  3. Compute verifying hash value
  $$Mac = PRF(S \| t_{w_1} \cdots \| t_{w_t}, k_3).$$
  4. Output the the searching results (S, Mac).

Algorithm 4 Public Verification Circuit $\mathcal{P}_{PV}$

- *Constants*: Secret key (k_1, k_3).
- *Inputs*: Searching token t_w, Searching result $(\mathcal{S}_w, Mac)$, encrypted file set $\mathcal{C}_w$.
- *Outputs*:
  1. Verify that $PRF(S_w \| t_w, k_3) \stackrel{?}{=} Mac$.
  2. Verify that $H(k_1, \mathcal{S}_w^i \| \mathcal{E}.Enc(k_1, F_i)) \stackrel{?}{=} \mathcal{C}_w^i.h_{mac}$.
  3. If both accept, output 1; else output 0.

5. ANALYSIS OF OUR SSE CONSTRUCTION

We evaluate the proposed scheme by analyzing its security and performance in this section. First, we will show how the $i\mathcal{O}$-based SSE meets our security guarantee defined in section 2. Then we will give the performance analysis and comparisons with existing verifiable SSE schemes.

5.1 Security Analysis

We define the leakage function L as follows: on input the index table $\mathcal{T}$ and the set of files $\mathcal{F}$, it outputs the number of files n, number of keywords m and the size of each file; on the input search token t_w, it outputs the identifier set $\mathcal{S}_w$ of the files containing keyword w. Then we have:

THEOREM 1. *If PRF is a secure punctured PRF, and $i\mathcal{O}$ a secure indistinguishability obfuscator, then SSE construction in section 4.1 is L-secure.*

Proof. We prove the security through a sequence of indistinguishable games.

Game 0 This game is same to the game $Real_A^{\Pi}(k)$.

- Challenger runs $K \leftarrow Gen(1^k)$ to get secret keys $K = (k_1, k_2, k_3)$.
- Adversary $A(1^k)$ chooses data files $\mathcal{F}$, then challenger runs $(i\mathcal{O}(\mathcal{P}_{search}), \mathcal{C}) \leftarrow Enc(K, \mathcal{W}, \mathcal{F})$ and sends $i\mathcal{O}(\mathcal{P}_{search})$ and EDB to A.
- Adversary A makes polynomial number of queries by picking different keywords w_i. Challenger responds with $t_{w_i} = H(w_i, k_2)$
- A returns a bit b that is output by the experiment.

Game 1 Let q_{QH} be the upper bound on the number of registered honest queries. Before the game begins, choose q_{QH} random values $t_i^* \in \mathcal{Y}$ ($H : \mathcal{X} \times \mathcal{K} \rightarrow \mathcal{Y}$). We will use these t_i^* values as the t_{w_i} values to answer honest token queries. Besides this, other steps are the same with Game 0. As the famous leftover hash lemma [23] shows that universal hash functions are good randomness extractors, the tokens are indistinguishable from those in Game 0.

Game 2 Notice that with overwhelming probability, none of the t_i^* for token queries in Game 1 have a valid keyword preimage under hash function H. Therefore, with overwhelming probability, there is no input to $\mathcal{P}_{search}$ that will match any searched index set S. We construct a circuit C that takes these tokens as input and accepts input S if and only if the corresponding token of S is not contained in $T^* = (t_1^*, t_2^*, \cdots, t_{q_{QH}}^*)$, then construct the constrained function PRF^C following the definition in section 3. Also, we build a new encrypted database $EDB' = \{(\mathrm{id}(F_i), C_i')\}$, $i = 1 \cdots n$ and a new look-up table $\mathcal{T}'$ from the leakage function L. Last, replace PRF with PRF^C, $\mathcal{T}$ with $\mathcal{T}'$ in the program of $\mathcal{P}_{search}$, and then generate the program $\mathcal{P}'_{search}$ described in Algorithm 5. So we construct Game 2 as follows:

- Adversary $A(1^k)$ chooses data files $\mathcal{F}$. Challenger generates new encrypted database EDB' according to the leakage function. Then it generates program $\mathcal{P}'_{search}$ as above and outputs indistinguishability obfuscator $i\mathcal{O}(\mathcal{P}'_{search})$. At last, challenger sends $i\mathcal{O}(\mathcal{P}'_{search})$ and EDB' to A.
- Adversary A makes at most q_{QH} queries by picking different keywords w_i. Challenger responds with $T^* = (t_1^*, t_2^*, \cdots, t_{q_{QH}}^*)$.
- A returns a bit b that is output by the experiment.

Algorithm 5 Search Circuit $\mathcal{P}'_{search}$

- *Constants*: PRF^C, Look-up table $\mathcal{T}'$.
- *Inputs*: searching token t_w.
- *Outputs*:
  1. For $j = 1 \cdots t$, verify that $t_{w_j} \stackrel{?}{=} t_w$. If yes, set $S = S_j$; if no item in $\mathcal{T}'$ matches, abort.
  2. Compute $Mac = PRF^C(S \| t_w, k_3)$.
  3. Output the the searching results (S, Mac).

Observe that, with overwhelming probability, $\mathcal{P}'_{search}$ and $\mathcal{P}_{search}$ have the same functionality. Thus from the security of indistinguishability obfuscator, $i\mathcal{O}(\mathcal{P}'_{search})$ and $i\mathcal{O}(\mathcal{P}_{search})$ are computationally indistinguishable. Therefore, Game 2 is indistinguishable from Game 1.

From the above sequence of games, we have that Game 0 is computationally indistinguishable from Game 2. While, Game 0 is same to the game $Real_A^{\Pi}(k)$ and Game 2 is the same to game $Idea_{A,S}^{\Pi}(k)$ described in definition 1. Thus we proved that the $i\mathcal{O}$-based SSE scheme is L-secure.

THEOREM 2. *If MAC scheme is unforgeable, and $i\mathcal{O}$ a secure indistinguishability obfuscator, then SSE construction in section 4.1 satisfies verifiability.*

Proof. If the malicious server forges an invalid response which makes the $Verify$ algorithm output 1, then we can utilize the malicious server to forge a valid MAC value. From the SSE construction in section 4.1, the cloud server inputs the searching token to the obfuscator $i\mathcal{O}(P_{search})$ and obtains the outputs (S, Mac). The verification value Mac can be verified by using the client's secret key k_3. So if a malicious cloud server does not honestly search by the obfuscator and return the wrong results, then the MAC verification will not succeed. Because as long as the cloud server does not know the secret key k_3, it cannot forge a valid pair $\{S, Mac\}$ to deceive the client. Similarly, the server cannot construct a valid mapping for identifier and non-corresponding encrypted files without key k_1. By this method, the client can detect the cloud server's malicious actions. For more details about the security proof of the MAC authentication scheme, readers are referred to [11].

5.2 Performance and Functionality

In this section, we will give a performance analysis of our basic theoretical $i\mathcal{O}$-based scheme and compare them with existing works.

For the client, like other prior works, it needs a preprocess in the Enc phase. It will encrypt all the data files first and generate an obfuscated searching circuit. Although the pre-computation time is not ideally small, we want to emphasize that it is a one-time effort which can be amortized to plenty of following queries. Compared to client's pre-computation, we would rather to focus on the computation and communication overheads in the $Search$ phase, which can affect user experience more due to the high frequency.

Table 1 shows a comparison between our $i\mathcal{O}$-based SSE and other related works. We use n to denote the number of total files stored in the server, m, the number of all keywords and r, the number of the retrieving files for a certain keyword. As shown in the table, Kamara *et al.* [22] provides the fastest search time so far because it benefits from a parallel search (p means the number of parallel processors) method. However, it does not support verifiability. Kurosawa *et al.* [11] has proposed the first verifiable SSE scheme against malicious server. The computation and communication overhead of their search phase is proportional to n. Their following work [24] adds a dynamic property but still does not improve the efficiency.

6. CONCLUSIONS

In this paper, we proposed a secure searchable symmetric encryption scheme based on indistinguishability obfuscation as an initial attempt. Our scheme considers a threat model with a malicious cloud server and provides verifiability against the server. We first present a basic scheme supporting single keyword query, and then extend it to multiple search functionalities such as conjunctive and boolean queries. Further more, our scheme can be extended to support public verifiability. The security of our SSE scheme is guaranteed

	Computation (Search)	Communication (Search)	Verifiability	Against Malicious Server	Against Malicious Client
KO12[11]	$\mathcal{O}(n)$	$\mathcal{O}(r)$	YES	YES	NO
KP13[22]	$\mathcal{O}((rlogn)/p)$	$\mathcal{O}(1)$	NO	NO	NO
KO13[24]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	YES	YES	NO
CG12[10]	$\mathcal{O}(m)$	$\mathcal{O}(r)$	YES	NO	NO
Our Scheme	$\mathcal{O}(m) + \mathcal{O}(r)$	$\mathcal{O}(r)$	YES	YES	YES

Table 1: Comparison of Verifiable SSE schemes. n denotes the number of total files, m denotes the number of all keywords, r denotes the number of the retrieving files for a certain keyword.

by the indistinguishability obfuscation and the MAC authentication concept. We also employ the constrained PRF in our security analysis, which proves that our scheme is secure with some acceptable information leakage. The computation process of the cloud server in the search phase is isomorphic with search in the plaintext domain and the communication overhead is linear with the number of retrieval files.

7. ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (No. 61379154 and U1135001), Specialized Research Fund for the Doctoral Program of Higher Education under Grant 20120171110027, and US National Science Foundation under grant CNS-1262277.

8. REFERENCES

[1] K. Ren, C. Wang, and Q. Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, 2012.

[2] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC 2009 Proceedings*, pages 169–178, 2009.

[3] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[4] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *Symposium on Research in Security and Privacy (SSP)*, pages 44–55, 2000.

[5] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *ACNS 2004 Proceedings*, pages 31–45. Springer-Verlag, 2004.

[6] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM CCS 2012 Proceedings*, pages 965–976, 2012.

[7] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO 2013 Proceedings*, pages 353–373. Springer-Verlag, 2013.

[8] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy keyword search over encrypted data in cloud computing. In *Proceedings of the 29th INFOCOM*, pages 441–445. IEEE, 2010.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM 2011 Proceedings*. IEEE, 2011.

[10] Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *ICC 2012 Proceedings*, pages 917–922, 2012.

[11] K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *FC 2012 Proceedings*, pages 285–298. Springer-Verlag, 2012.

[12] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO 2001 Proceedings*, pages 1–18. Springer-Verlag, 2001.

[13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS 2013 Proceedings*, pages 40–49, 2013.

[14] D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO 2014 Proceedings*, pages 480–499. Springer-Verlag, 2014.

[15] A. Sahai and B. Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *STOC 2014 Proceedings*, pages 475–484, 2014.

[16] K. Ramchen and B. Waters. Fully secure and fast signing from obfuscation. In *ACM CCS 2014*.

[17] S. Hohenberger, A. Sahai, and B. Waters. Replacing a random oracle: full domain hash from indistinguishability obfuscation. In *EUROCRYPT 2014 Proceedings*, pages 201–220. Springer-Verlag, 2014.

[18] D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *AsiaCrypt 2013 Proceedings*, pages 1–23. Springer-Verlag, 2013.

[19] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC 2014 Proceedings*, pages 501–519. Springer-Verlag, 2014.

[20] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 2013 Proceedings*, pages 669–684, 2013.

[21] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS 2006 Proceedings*, pages 79–88, 2006.

[22] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *FC 2013 Proceedings*, pages 258–274. Springer-Verlag, 2013.

[23] J. Hastad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28, 1999.

[24] K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *CNS Proceedings*, pages 309–328, 2013.