An Efficient Non-interactive Multi-client Searchable Encryption with Support for Boolean Queries

Shi-Feng Sun[1], Joseph K. Liu[2](✉), Amin Sakzad[2], Ron Steinfeld[2], and Tsz Hon Yuen[3]

[1] Shanghai Jiao Tong University, Shanghai, China
crypto99@sjtu.edu.cn
[2] Faculty of Information Technology, Monash University, Melbourne, Australia
{joseph.liu,amin.sakzad,ron.steinfeld}@monash.edu
[3] Huawei, Singapore, Singapore
YUEN.TSZ.HON@huawei.com

Abstract. Motivated by the recent searchable symmetric encryption protocol of Cash et al., we propose a new multi-client searchable encryption protocol in this work. By tactfully leveraging the RSA-function, our protocol avoids the per-query interaction between the data owner and the client, thus reducing the communication overhead significantly and eliminating the need of the data owner to provide the online services to clients at all times. Furthermore, our protocol manages to protect the query privacy of clients to some extent, meaning that our protocol hides the exact queries from the data owner. In terms of the leakage to server, it is exactly the same as Cash et al., thus achieving the same security against the adversarial server. In addition, by employing attribute-based encryption technique, our protocol also realizes the fine-grained access control on the stored data. To be compatible with our RSA-based approach, we also present a deterministic and memory-efficient 'keyword to prime' hash function, which may be of independent interest.

Keywords: Cloud storage · Searchable encryption · Non-interaction · Multi-client · RSA function

1 Introduction

Cloud technology is now a major industry trend that offers great benefits to users. Cloud storage (or data outsourcing) provides an excellent way to extend the capability to store large volume of data, to prepare for the high velocity of data generation, and to easily process the high variety of data (the "3V" of Big Data). In other words, cloud storage is well designed for the big data era. Meanwhile, data outsourcing raises confidentiality and privacy concerns [2,9,20–24]. Simple encryption technology can protect data confidentiality easily. However, it is not possible to search within the encrypted domain. In order to

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part I, LNCS 9878, pp. 154–172, 2016.
DOI: 10.1007/978-3-319-45744-4_8

search for a particular keyword, user has to decrypt the data first, before starting the searching process. It is not practical especially when the volume of data is large. Searchable encryption (SE) [5,7,8,11,32] is a cryptographic primitive addressing encrypted search.

The architecture of SE can be classified into 4 types: single-writer/single-reader, single-writer/multi-reader, multi-writer/single-reader and multi-writer/multi-reader. The traditional single-writer/single-reader allows the data owner to first use a special encryption algorithm which produces an encrypted version of the database, including encrypted metadata, that is then stored on an external server. Later, data owner can interact with the server to carry out a search on the database and obtain the results (this is also called the symmetric setting as there is only one writer to the database, the owner, who uses symmetric encryption.) Single-writer/multi-reader SE allows an arbitrary group of parties other than the owner to submit search queries. The owner can control the search access by granting and revoking searching privileges to other users.

In the setting of searching on public-key-encrypted data, users who encrypt the data can be different from the owner of the decryption key. This creates the model for multi-writer/single-reader SE. A more generalized model further allows every user to write an encrypted document to the database as well as to search within the encrypted domain, including those ciphertexts produced by other users. This is the multi-reader/multi-writer setting.

In the rest of the paper, we focus on the single-writer/multi-reader setting. In this framework, whenever a reader (or client) wants to search over the database, she usually needs to perform a per-query interaction with the writer (or data owner) and asks the data owner to produce and send back the necessary trapdoor information to help her carry out the search, as shown in the representative work [16]. Thus, the data owner is required to be online all the time. However, the initial goal of the data owner is to outsource his storage and services to the cloud server, so removing the per-query interaction between the data owner and the client is a desired feature.

1.1 Our Contributions

In this work, we first present a deterministic and memory-efficient hash function, which maps keywords to primes. With this function, we then propose an efficient non-interactive multi-client searchable encryption in the single-writer/ multi - reader setting, with support for boolean queries. Our construction enjoys the following nice features:

1. Our construction is motivated by the searchable symmetric encryption (SSE) protocol of Cash et al. [7] (CASH). When compared to its multi-client version [16] (MULTI), we improve the communication overhead between the data owner and the client significantly. In fact, MULTI requires the client to interact with the data owner each time she wants to search on database. For each query, the data owner responds by generating a partial search token and

sending it back to the client. Then the client generates the full token and forwards it to the server to facilitate the searching process. In return, the server sends to the client an encrypted index (or document identifier), by decrypting which the client gets the plaintext document identifier. In our construction, we totally eliminate the interactive process, except at the beginning the client needs to obtain a search-authorized private key from the data owner for some permitted keywords. With the private key, the client can generate a search token for any boolean queries on those permitted keywords. In the return of the encrypted indices from the server, the client is also able to decrypt them without obtaining any assistance from the data owner.
2. We also note that there is a naive approach to turn MULTI into non-interactive setting. The data owner can pre-generate all possible search tokens for the client. The number of pre-generated tokens is of order $\mathcal{O}(M)$, where M is the number of possible queries the client is allowed to make. Our construction only requires the data owner to generate a search-authorized secret key to the client. The size of the secret key is of order $\mathcal{O}(1)$, which is actually just 3072 bits (with respect to 1024-bit RSA security) regardless of the number of permitted queries.
3. We deploy Attribute-Based Encryption (ABE) mechanism to allow the client to decrypt the encrypted indices given by the server without any assistance from the data owner. According to our framework, the data owner can also realize fine-grained access control on his data. In addition, the data owner in our protocol does not know which particular queries the client has generated or which documents the client has retrieved, provided that the data owner has authorized the client to search for a set of permitted keywords. In terms of information leakage to the server, we show that our construction is exactly the same as CASH, meaning the transcripts between the client and the server in real protocol can be properly simulated only with the same leakage profile as CASH. Regarding the expressiveness, our protocol is similar to CASH, which allows clients to perform arbitrary boolean queries efficiently.

1.2 Related Works

The first SE by Song et al. [32] is presented in the single-writer/single-reader setting. The first notion of security for SE was introduced by Goh [14]. Curtmola et al. [11] proposed the strong security notion of IND-CKA2. Kurosawa and Ohtaki [19] provided the IND-CKA2 security in the universal composability (UC) model. On the other hand, Boneh et al. [5] introduced the first public key encryption with keyword search, together with the security model in the multi-writer/single-reader architecture.

Kamara et al. [17,18] proposed dynamic SE schemes which allow efficient update of the database. Golle et al. [15] gave the first SE with conjunctive keyword searches, in the single-writer/single-reader setting, but its search time is linear in the number of keywords to search. Most recently, Cash et al. [7] proposed the first sublinear SE with support for boolean queries and efficiently implemented it in a large database [6].

In the single-writer/multi-reader architecture, Curtmola et al. [11] proposed a general construction based on broadcast encryption, which leads to a relatively inefficient implementation. The search time of the scheme by Raykova et al. [29] is linear in the number of documents. The scheme uses deterministic encryption and directly leaks the search pattern in addition to the access pattern. Recently, Jarecki et al. [16] extend the scheme by Cash et al. [7] to a single-writer/multi-reader setting, which preserves all nice features provided by the original scheme but requires a per-query interaction between the data owner and the client.

In the multi-writer/multi-reader setting, a number of schemes [1,3,12,33] were presented with a high level of security, but the search time is linear in the number of keywords per document. The scheme in [27] improved the search complexity by removing the need of TTP in previous schemes. In addition, a stronger model for access pattern privacy was proposed in [30]. However, all these schemes only support single keyword searches.

2 Preliminaries

In this section, we give a list of notations and terminologies (cf. Table 1) used through our work and a brief review of hardness assumptions and cryptographic primitives deployed in our construction.

2.1 Hardness Assumptions

Definition 1 (DDH problem). *Let $\mathbb{G}$ be a cyclic group of prime order p, the decisional Diffie-Hellman (DDH) problem is to distinguish the ensembles $\{(g, g^a, g^b, g^{ab})\}$ from $\{(g, g^a, g^b, g^z)\}$, where the elements $g \in \mathbb{G}$ and $a, b, z \in \mathbb{Z}_p$ are chosen uniformly at random. Formally, the advantage for any PPT distinguisher $\mathcal{D}$ is defined as:*

$$Adv_{\mathcal{D},\mathbb{G}}^{DDH}(\kappa) = |\Pr[\mathcal{D}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{D}(g, g^a, g^b, g^z) = 1]|.$$

We say that the DDH assumption holds if for any PPT distinguisher $\mathcal{D}$, its advantage $Adv_{\mathcal{D},\mathbb{G}}^{DDH}(\kappa)$ is negligible in κ.

Definition 2 (Strong RSA Problem [10]). *Let $n = pq$, where p and q are two κ-bit prime numbers such that $p = 2p' + 1$ and $q = 2q' + 1$ for some primes p', q'. Let g be a random element in $\mathbb{Z}_n^*$. We say that an efficient algorithm $\mathcal{A}$ solves the strong RSA problem if it receives as input the tuple (n, g) and outputs two elements (z, e) such that $z^e = g \bmod n$.*

2.2 Pseudorandom Functions

Let $F : \{0,1\}^\kappa \times \mathcal{X} \rightarrow \mathcal{Y}$ be a function defined from $\{0,1\}^\kappa \times \mathcal{X}$ to $\mathcal{Y}$. We say F is a pseudorandom function (PRF) if for all efficient adversaries $\mathcal{A}$, its advantage $Adv_{F,\mathcal{A}}^{\text{prf}}(\kappa) = |\Pr[\mathcal{A}^{F(K,\cdot)}(1^\kappa)] - \Pr[\mathcal{A}^{f(\cdot)}(1^\kappa)]| < negl(\kappa)$, where $K \xleftarrow{\$} \{0,1\}^\kappa$ and f is a random function from $\mathcal{X}$ to $\mathcal{Y}$.

Table 1. Notations and terminologies

Notation	Meaning
κ	A security parameter
id_i	The document identifier of the i-th document
W_{id_i}	A list of keywords contained in the i-th document
$\mathsf{DB} = (id_i, \mathrm{W}_{id_i})_{i=1}^{d}$	A database consisting of a list of document identifier and keyword-set pairs
$\mathsf{DB}[w] = \{id : w \in \mathrm{W}_{id}\}$	The set of identifiers of documents that contain keyword w
$\mathrm{W} = \bigcup_{i=1}^{d} \mathrm{W}_{id_i}$	The keyword set of the database
RDK	The retrieval decryption key array, used to retrieve the original documents
$\mathcal{U}$	The attribute universe of the system
$[T]$	The set of positive integers not larger than T, i.e., $\{1, 2, \ldots, T\}$
a$\|$b	The concatenation of a and b
$s \overset{\$}{\leftarrow} S$	The operation of uniformly sampling a random element s from a set S
sterm	The least frequent term among queried terms (or keywords) in a search query
xterm	Other queried terms in a search query (i.e., the queried terms excluding sterm)
PPT	The abbreviation of probabilistic polynomial time
$negl(\kappa)$	A negligible function in the security parameter κ

2.3 Non-interactive Multi-client Searchable Encryption

In our single-writer/multi-reader (we call it multi-client in the rest of this paper) setting, there are three parties: the data owner of the plaintext database, a service provider that stores the encrypted database, and the clients who want to perform search queries over the database. In more details, the data owner outsources his search service to a cloud server, and generates a search-authorized private key for each client in terms of her credentials. When a client performs a search query, she generates the search token by herself using her own private key and then forwards the token to the service provider. With the token, the server finally retrieves the encrypted identifier or documents for the client. Formally, the syntax of our non-interactive multi-client searchable encryption consists of the following algorithms:

- $\mathsf{EDBSetup}(1^{\kappa}, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$: the data owner takes κ, $\mathsf{DB}, \mathrm{RDK}$ and $\mathcal{U}$ as input and generates the system master key MK and public key PK, with which he processes the plaintext database DB and outsources the encrypted database EDB and XSet to the server.

- $\mathsf{ClientKGen}(\mathrm{MK}, S, \mathbf{w})$: for a client with attribute set S, the data owner takes MK, S and a set $\mathbf{w}$ of permitted keywords as input and generates a search-authorized private key sk for the client. Note that $\mathbf{w}$ is authorized by the data owner according to the client's credentials.
- $\mathsf{TokenGen}(sk, Q)$: the client uses her private key sk to produce the search token st for the query Q she wants to perform.
- $\mathsf{Search}(st, \mathsf{EDB}, \mathsf{XSet})$: with the search token st, the server carries out the search over the encrypted database EDB and XSet and returns the matching results R to the client.
- $\mathsf{Retrieve}(sk, R)$: the client uses her private key sk to decrypt the search result R (returned by the sever) and retrieves the original documents using the relevant document identifiers and decryption keys.

2.4 Security Definitions

In this section, we give security definitions of our searchable encryption. In the multi-client setting, we consider both securities with respect to (w.r.t.) the adversarial server and the clients. Similar to [7], we do not model the retrieval of encrypted documents in the security analysis and just focus on the storage and processing of the metadata.

First, let us consider the security w.r.t. the adversarial server, which can be extended straightforwardly from [7]. This security is parameterized by a leakage function $\mathcal{L}$, as described below, which captures information allowed to learn by an adversary from the interaction with a secure scheme. Loosely speaking, the security says that the server's view during an adaptive attack can be properly simulated given only the output of the leakage function $\mathcal{L}$. As in [7], the "adaptive" here means the server selects the database and queries. Moreover, it selects the authorized keywords for each client in our setting.

Let $\Pi = (\mathsf{EDBSetup}, \mathsf{ClientKGen}, \mathsf{TokenGen}, \mathsf{Search})$ be a searchable encryption scheme and $\mathcal{A}, \mathcal{S}$ be two efficient algorithms. The security is formally defined via a real experiment $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa)$ and an ideal experiment $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa)$ as follows:

$\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa)$: $\mathcal{A}(1^\kappa)$ chooses a database DB. Then the experiment runs the algorithm $(\mathrm{MK}, \mathrm{PK}, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathsf{EDBSetup}(1^\kappa, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$ and returns $(\mathrm{PK}, \mathsf{EDB}, \mathsf{XSet})$ to $\mathcal{A}$. After that, $\mathcal{A}$ selects a set $\mathbf{w}$ of authorized keywords for a client and then repeatedly chooses a search query q, where we assume the keywords associated with q are always within the authorized keyword set $\mathbf{w}$. To respond, the experiment runs the remaining algorithms in Π (including $\mathsf{ClientKGen}$, $\mathsf{TokenGen}$ and Search), and gives the transcript and client output to $\mathcal{A}$. Eventually, the experiment outputs the bit that $\mathcal{A}$ returns.

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa)$: The game initializes an empty list $\mathbf{q}$ and a counter $i = 0$. $\mathcal{A}(1^\kappa)$ chooses a DB. Then the experiment runs $(\mathrm{PK}, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{DB}))$ and gives $(\mathrm{PK}, \mathsf{EDB}, \mathsf{XSet})$ to $\mathcal{A}$. $\mathcal{A}$ then repeatedly chooses a search query q. To respond, the experiment records this query as $\mathbf{q}[i]$, increments i and gives the output of $\mathcal{S}(\mathcal{L}(\mathsf{DB}, \mathbf{q}))$ to $\mathcal{A}$, where $\mathbf{q}$ consists of all previous queries in addition to the latest query issued by $\mathcal{A}$. Eventually, the experiment outputs the bit that $\mathcal{A}$ returns.

Definition 3 (Security w.r.t. Server). *The scheme* Π *is called* $\mathcal{L}$*-semantically-secure against adaptive attacks if for all PPT adversaries* $\mathcal{A}$ *there exists an efficient simulator* $\mathcal{S}$ *such that* $|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa)]| \leq$ *negl*(κ).

Before going ahead, we first give the description of leakage function $\mathcal{L}$ used in our security analysis. We note that, for sake of simplicity, we only present the detailed security proof of our scheme for conjunctive queries, so we start by describing the leakage function for such a simple scenario. Actually, the scheme and security proof can be readily adapted to any search boolean queries, which will be further discussed later.

In the following, we represent a sequence of T conjunctive queries by $\mathbf{q} = (\mathbf{s}, \mathbf{x})$, where $\mathbf{s}[t]$ and $\mathbf{x}[t, \cdot]$ for $t \in [T]$ denote the sterm and xterms in the t-th query respectively, and each individual query is written as $\mathbf{q}[i] = (\mathbf{s}[i], \mathbf{x}[i, \cdot])$. With DB and $\mathbf{q}$ as input, the leakage function outputs the following leakage items:

- $N = \sum_{i=1}^{d} |\mathrm{W}_{id_i}|$ is the number of keyword-document pairs, which is the size of EDB and XSet.
- $\bar{\mathbf{s}} \in \mathbb{N}^T$ is the equality pattern of the sterms $\mathbf{s}$, indicating which queries have the same sterms. It is calculated as an array of integers, such that each integer represents one sterm. For instance, if we have $\mathbf{s}$ = (a, b, c, a, a), then $\bar{\mathbf{s}}$ = (1, 2, 3, 1, 1).
- $\mathrm{SP}[\sigma]$ is the size pattern of the queries, which is the number of matching results returned for each stag. Note that we index it by the values of $\bar{\mathbf{s}}$, i.e., $\sigma \in \bar{\mathbf{s}}$, instead of the query number t as in [7], so we have $\mathrm{SP}[\bar{\mathbf{s}}[t]] = |\mathsf{DB}[\mathbf{s}[t]]|$.
- $\mathrm{RP}[t, \alpha] = \mathsf{DB}[\mathbf{s}[t]] \cap \mathsf{DB}[\mathbf{x}[t, \alpha]]$, where $\mathbf{s}[t] \neq \mathbf{x}[t, \alpha]$, reveals the intersection of the sterm with any other xterm in the same query.
- $\mathrm{SRP}[t] = \mathsf{DB}[\mathbf{s}[t]]$ is the search result pattern corresponding to the stag of the t-th query.
- $\mathrm{IP}[t_1, t_2, \alpha, \beta] = \begin{cases} \mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[\mathbf{s}[t_2]], & \text{if } \mathbf{s}[t_1] \neq \mathbf{s}[t_2] \text{ and } \mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta] \\ \emptyset, & \text{otherwise} \end{cases}$
  is the conditional intersection pattern, which is a generalization of the IP structure in [7].
- $\mathrm{XT}[t] = |\mathbf{x}[t, \cdot]|$ is the number of xterms in the t-th query.

The leakage function for our protocol is similar to [7], but a number of components have been generalized and some additional components are introduced. The generalization of SP is straightforward. RP has changed a lot. Within a query, it is possible to test the results from the stag against any other keyword, since a full xtoken is sent to the server. RP captures this as the intersection between the sterm and xterms. IP is also generalized, where any of the sterms for each conjunctive query is considered instead of only one xterm per query. Of the additional pieces of leakage, XT is straightforward. However, there is also a component SRP which represents the results corresponding to any sterm. This component overstates the true leakage but is required by the design of the proof.

Actually, RP and IP also overstate the leakage they represent, because the server in real protocol never has access to the unencrypted indices.

Next, we continue to consider the security w.r.t. adversarial clients. In our setting, whenever a legitimate client registers to the system, the data owner assigns a set of keywords and generates the associated private key for the client according to her attributes or credentials. Thus, each client is only permitted to perform search queries on the authorized keywords in our system. Loosely speaking, the security requires that it be impossible to forge a valid search token for a query containing some non-authorized keywords, even for an adaptive client (who can select the authorized keywords by herself). That is, the malicious client is not allowed to gain information beyond what she is authorized for. Formally, the security is defined via the following game $\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{A},token}(\kappa)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:

Initialization: the challenger runs the setup algorithm $(\mathrm{MK}, \mathrm{PK}, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathsf{EDBSetup}(1^\kappa, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$ and returns the system public key PK to the adversary $\mathcal{A}$.

Client key extraction: when receiving a private key extraction request for keywords $\mathbf{w} = (w_1, \ldots, w_n)$, the challenger $\mathcal{C}$ runs the client key generation algorithm $sk \leftarrow \mathsf{ClientKGen}(\mathrm{MK}, S, \mathbf{w})$ and sends back sk to $\mathcal{A}$.

Output: Eventually, the adversary outputs a search token st for a new query containing some keyword $w' \notin \mathbf{w}$, and the challenger outputs 1 if st is valid.

Definition 4 (Security w.r.t. Client). *The search token in* Π *is said to be unforgeable against adaptive clients if for all PPT adversaries* $\mathcal{A}$ *its advantage* $\Pr[\mathbf{Exp}^{\mathrm{UF}}_{\mathcal{A},token}(\kappa) = 1] \leq negl(\kappa)$.

Note that in our syntax search tokens are produced by clients using their private keys, so if the generation of valid tokens is (almost) equivalent to that of the corresponding private key, then the security can be formulated in terms of forging a valid private key instead of a search token (i.e., the goal of the adversary in the game is to finally output a valid private key for some unauthorized keyword $w' \notin \mathbf{w}$). For the proof of our scheme, we will follow the latter equivalent way.

3 A Deterministic, Memory-Efficient Mapping from Keywords to Primes

Before presenting our multi-client SE protocol, we first give an efficient 'keyword to prime' hash function. In our work, we assume that the search index keywords have been mapped during the encrypted database setup to *prime* integers, in order to be compatible with our RSA-based token-derivation function, and that the token generation and search algorithms can re-compute the same corresponding primes for the keywords searched by the client. A straightforward approach to implement such a mapping would be to use a lookup table at the data owner and client, storing all keywords and their corresponding primes. While computationally efficient, this approach requires memory storage at the data owner proportional to the total number $|\mathrm{W}|$ of keywords in the database index, and memory storage at the client proportional to the number of keywords n to be searched for by this client, which may be prohibitive and would eliminate the advantage of the compact (constant length independent of n) client tokens of our protocol.

In this section, we show how to avoid the storage overheads of the lookup table approach, by constructing a *deterministic* and *memory-efficient* collision-resistant hash function for mapping keywords to their corresponding primes. In this construction, the memory requirements at the data owner and client are constant, indpendent of the number of keywords $|\mathrm{W}|$ in the index or the number of keywords n at the client.

Our construction of a 'keyword to prime' collision-resistant hash is a deterministic variant of the randomized 'strings to primes' hash function introduced by Gennaro, Halevi and Rabin [13].

Construction. The main idea is to use the randomized hash function introduced in [13] along with a primality test algorithm, derandomizing the result by using a pseudorandom function (PRF) and choosing the first prime in a psedurandom sequence of integers as the hash output. Our construction builds a collision-resistant 'keyword-to-prime' hash function family $\mathcal{H}$, where each function $h \in \mathcal{H}$ maps the keyword space W to the set $P_{2\kappa}$ of 2κ-bit prime integers. The construction uses the following ingredients:

- A collision-resistant hash family $\bar{\mathcal{H}}$, where each function $\bar{h} \in \bar{\mathcal{H}}$ maps W to the set of κ-bit strings $\{0,1\}^\kappa$.
- A PRF family $\mathcal{F}$, where each function $F_k \in \mathcal{F}$ maps $\{0,1\}^\kappa$ to $\{0,1\}^\kappa$.

We let Int denote the natural mapping from a binary string in $c \in \{0,1\}^\kappa$ to the integer $\mathsf{Int}(c)$ in $[0, 2^\kappa - 1]$ whose binary representation is c, and denote by Bin its inverse mapping from integers to binary strings. A hash function $h : \mathrm{W} \to P_{2\kappa}$ from our family $\mathcal{H}$ is specified by randomly picking a function $\bar{h}$ from the collision-resistant family $\bar{\mathcal{H}}$ and a pseudorandom function F_k from the PRF family $\mathcal{F}$. The algorithm for evaluating the function h on a given keyword $x \in \mathrm{W}$ using $(\bar{h}, F_k)$ to get a corresponding prime $w \in P_{2\kappa}$ is presented in Algorithm 1.

Lemma 1. (1) *The hash family $\mathcal{H}$ is collision-resistant if the hash family $\bar{\mathcal{H}}$ is collision-resistant.* (2) *Furthermore, if family $\mathcal{F}$ is a pseudorandom function family, and the density of primes in the intervals* $[2^\kappa \cdot \bar{h}(x), 2^\kappa \cdot \bar{h}(x) + 2^\kappa - 1]$ *is* $\geq 1/\ln(2^{2\kappa})$ *for each x (as heuristically expected from the Prime Number Theorem), then for each input $x \in \mathrm{W}$ and $m \geq 1$, the number of iterations of the while loop in Algorithm 1 is $\leq 1.4 \cdot m \cdot \kappa$, except with probability negligibly larger than* $\exp(-m)$.

We now estimate the practical cost of evaluating our hash function h. The memory storage costs are constant (independent of the size of the keyword set W), namely the cost of storing the two keys for the functions $\bar{h}$ and the

Algorithm 1. h: Hashing from Keywords to Primes

```
Input: keyword $x \in \mathrm{W}$, functions $\bar{h} : \mathrm{W} \rightarrow \{0,1\}^\kappa \in \bar{\mathcal{H}}$, $F_k : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa \in \mathcal{F}$
Output: prime integer $w \in P_{2\kappa}$
 1: foundprime ← False
 2: $r \leftarrow 0$.
 3: while foundprime = False do
 4:     let $w \leftarrow 2^\kappa \cdot \mathsf{Int}(\bar{h}(x)) + \mathsf{Int}(F_k(\mathsf{Bin}(r)))$   // random int. with MS bits equal to $\bar{h}(x)$
 5:     if $w$ is prime then
 6:         let foundprime ← True
 7:     end if
 8:     let $r \leftarrow r + 1 \bmod 2^\kappa$
 9: end while
10: return $w$.
```

PRF F_k. The main computation cost in Algorithm 1 is the cost of each primality check of the 2κ-bit integer w in the iterations of the while loop. According to Lemma 1 with $m = 3$, the number of such primality tests would be $L \leq 4.2 \cdot \kappa$, except with small probability ≈ 0.05. Let $T_{\text{exp}}(2\kappa)$ denote the time needed to compute a full exponentiation modulo a 2κ-bit modulus. Assuming that we implement these primality checks using a Miller-Rabin probabilistic primality test [25,28], the expected cost [31, Chap. 10] of these L tests (at a $2^{-\kappa}$ false positive probability) would be at most $\kappa/2$ exponentiations modulo a 2κ-bit integer for the last while loop iteration, plus an expected ≤ 2 exponentiations modulo a 2κ-bit modulus for all other $L - 1$ iterations (which give composites), giving a total expected time of $T_h \leq (\kappa/2 + 2 \cdot L) \cdot T_{\text{exp}}(2\kappa)$. Furthermore, using fast trial division by small primes up to (say) 101 before testing with Miller-Rabin, would reduce the number of dominant Miller-Rabin tests to $L_{\text{MR}} \approx (\prod_{\text{prime } p \leq 101} \frac{p-1}{p}) \cdot L \leq 0.11 \cdot L$. Thus, the overall expected time for evaluating our hash function would be

$$T_h \leq (\kappa/2 + 0.22 \cdot L) \cdot T_{\text{exp}}(2\kappa) \approx 1.5\kappa \cdot T_{\text{exp}}(2\kappa).$$

Thus, for a typical security parameter $\kappa = 100$, we estimate T_h to be equivalent to about $150 \cdot T_{\text{exp}}(200)$ (i.e. 150 exponentiations with a 200-bit modulus). To put this into context with the rest of our protocol, the latter requires during each token generation to perform an exponentiation modulo a $\lambda \approx 2048$-bit modulus (to make sure the RSA problem has a $\approx 2^{100}$ secrity level) for each keyword w. Since the time $T_{\text{exp}}(\kappa)$ for an exponentiation modulo a κ-bit modulus is, assuming classical arithmetic, at least quadratic in κ, we have $T_{\text{exp}}(\lambda)/T_{\text{exp}}(2\kappa) = T_{\text{exp}}(2048)/T_{\text{exp}}(200) \geq (2048/200)^2 \approx 104$, so the cost of evaluating our hash function for w is expected to be only $T_h \approx 150/104 \cdot T_{\text{exp}}(2048) \approx 1.44 \cdot T_{\text{exp}}(2048)$, i.e. equivalent to only 1.44 exponetiations with a 2048-bit modulus, thus adding only a reasonable overhead to the computation time of our protocol for typical security parameters (2.44 exponentiations instead of 1 exponentiation modulo 2048-bit per keyword).

4 Our Construction

In this section, we present our SE scheme which mainly consists of four algorithms $\Pi = (\mathsf{EDBSetup}, \mathsf{ClientKGen}, \mathsf{TokenGen}, \mathsf{Search})$. For completeness, we also give the description of a simple original document retrieval algorithm $\mathsf{Retrieve}$, by which the client finally retrieves the desired documents from the cloud server. In our construction, we deploy CP-ABE as a primitive, which has been an effective and scalable access control mechanism for encrypted data and generally consists of four algorithms $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$. For its formal syntax and semantic security, please refer to [4]. We always assume that the set $\mathrm{W} = \bigcup_{i=1}^{d} \mathrm{W}_{id_i}$ of keywords in $\mathsf{DB} = (id_i, \mathrm{W}_{id_i})_{i=1}^{d}$ consists of distinct primes, which are mapped from the real keywords by our 'keyword to prime' function given in Sect. 3, and that a specific policy $\mathbb{A}$ is implicitly specified for each document identifier id_i.

$\mathsf{EDBSetup}(1^\kappa, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$: takes as input a security parameter κ, a database $\mathsf{DB} = (id_i, \mathrm{W}_i)_{i=1}^{d}$, a retrieval decryption key array RDK and an attribute universe $\mathcal{U}$, it chooses big primes p, q, random keys K_I, K_Z, K_X for a PRF F_p and K_S for a PRF F. Then it outputs the system master key $\mathrm{MK} = (p, q, K_S, K_I, K_Z, K_X, g_1, g_2, g_3, msk)$ and the corresponding system public key $\mathrm{PK} = (n, g, mpk)$, where $(mpk, msk) \leftarrow \mathsf{ABE.Setup}(1^\kappa, \mathcal{U})$, $n = pq$, $g \xleftarrow{\$} \mathbb{G}$ and $g_i \xleftarrow{\$} \mathbb{Z}_n^*$ for $i \in [3]$. Then it generates the encrypted database EDB and XSet with the system keys as the following Algorithm 2.

Algorithm 2. EDB Setup Algorithm

```
Input: MK, PK, DB, RDK
Output: EDB, XSet
 1: function EDBGen(MK, PK, DB, RDK)
 2:     $\mathsf{EDB} \leftarrow \{\}$; $\mathsf{XSet} \leftarrow \emptyset$
 3:     for $w \in \mathrm{W}$ do
 4:         $c \leftarrow 1$; $\mathrm{stag}_w \leftarrow F(K_S, g_1^{1/w} \bmod n)$
 5:         for $id \in \mathsf{DB}[w]$ do
 6:             $\ell \leftarrow F(\mathrm{stag}_w, c)$; $e \leftarrow \mathsf{ABE.Enc}(mpk, id||k_{id}, \mathbb{A})$
 7:             $\mathrm{xind} \leftarrow F_p(K_I, id)$; $z \leftarrow F_p(K_Z, g_2^{1/w} \bmod n||c)$
 8:             $y \leftarrow \mathrm{xind} \cdot z^{-1}$; $\mathrm{xtag} \leftarrow g^{F_p(K_X, g_3^{1/w} \bmod n) \cdot \mathrm{xind}}$
 9:             $\mathsf{EDB}[\ell] = (e, y)$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathrm{xtag}\}$
10:             $c \leftarrow c + 1$
11:         end for
12:     end for
13:     return EDB, XSet
14: end function
```

$\mathsf{ClientKGen}(\mathrm{MK}, S, \mathbf{w})$: assuming that a legitimate client with attribute set S is permitted to perform searches over keywords $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, the data owner generates a corresponding private key $sk = (K_S, K_I, K_Z, K_X, sk_S, sk_{\mathbf{w}})$, where $sk_S \leftarrow \mathsf{ABE.KeyGen}(msk, S)$ and $sk_{\mathbf{w}} = (sk_{\mathbf{w}}^{(1)}, sk_{\mathbf{w}}^{(2)}, sk_{\mathbf{w}}^{(3)})$ is computed as

Algorithm 3. Token Generation Algorithm

```
Input: $sk, Q$
Output: $st$
 1: function TokenGen(sk, Q)
 2:     $st, \mathsf{xtoken} \leftarrow \{\}$; $\bar{\mathbf{s}} \leftarrow \emptyset$
 3:     $\bar{\mathbf{s}} \leftarrow \bar{\mathbf{s}} \cup \{w'_1\}$
 4:     $\mathbf{x} \leftarrow \bar{\mathbf{w}} \setminus \bar{\mathbf{s}}$
 5:     $\mathrm{stag} \leftarrow F\big(K_S, (sk^{(1)}_{\mathbf{w}})^{\prod_{w \in \mathbf{w} \setminus \{w'_1\}} w} \mod n\big) = F(K_S, g_1^{1/w'_1} \mod n)$
 6:     for $c = 1, 2, \ldots$ until the server stops do
 7:         for $i = 2, \ldots, m$ do
 8:             $\mathsf{xtoken}[c, i] \leftarrow g^{F_p(K_Z, (sk^{(2)}_{\mathbf{w}})^{\prod_{w \in \mathbf{w} \setminus \{w'_1\}} w} \mod n || c) \cdot F_p(K_X, (sk^{(3)}_{\mathbf{w}})^{\prod_{w \in \mathbf{w} \setminus \{w'_i\}} w} \mod n)}$
                $= g^{F_p(K_Z, g_2^{1/w'_1} \mod n || c) \cdot F_p(K_X, g_3^{1/w'_i} \mod n)}$
 9:         end for
10:     end for
11:     $st \leftarrow (\mathrm{stag}, \mathsf{xtoken})$
12:     return $st$
13: end function
```

$$sk^{(i)}_{\mathbf{w}} = \left(g_i^{1/\prod_{j=1}^{n} w_j} \mod n\right) \text{ for } i \in [3].$$

At last, the data owner sends back sk together with $\mathbf{w}$ to the client, where we implicitly assume that the keyword appearance frequency satisfies $|w_1| < |w_2| < \cdots < |w_n|$.

$\mathsf{TokenGen}(sk, Q)$: whenever the client wants to search a boolean query Q on keywords $\bar{\mathbf{w}} \subseteq \mathbf{w}$, she first chooses sterms $\bar{\mathbf{s}} \subseteq \bar{\mathbf{w}}$ according to the query Q. For simplicity, we take the conjunctive query, $Q = w'_1 \wedge w'_2 \wedge \cdots \wedge w'_m$, as an example and assume that w'_1 is the chosen sterm, then the search token st (including stags and xtoken) for this query is computed as in Algorithm 3.

$\mathsf{Search}(st, \mathsf{EDB}, \mathsf{XSet})$: takes the search token $st = (\mathrm{stag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \cdots)$ for a query Q and $(\mathsf{EDB}, \mathsf{XSet})$, the server returns the search result R as in Algorithm 4.

$\mathsf{Retrieve}(sk, \mathrm{R})$: the client with private key sk_S decrypts the encrypted indices (search result R) and gets the matching document identifiers and retrieval decryption keys:

- For each $e \in \mathrm{R}$, recover $(id||k_{id}) \leftarrow \mathsf{ABE.Dec}(sk_S, e)$ if the client's attributes in S satisfy the access policy $\mathbb{A}$ assigned by the data owner to document identifier id.
- Send id to the server, get the encrypted document $ct = \mathsf{Enc}(k_{id}, doc)$, and retrieve the document $doc = \mathsf{Dec}(k_{id}, ct)$ with the corresponding symmetric key k_{id}.

Note that our protocol is derived from CASH and the RSA function, its correctness is easy to verify, which follows from the correctness of CASH and the underlying ABE. In addition, it is easy to observe that the plaintext identifiers

Algorithm 4. Search Algorithm

```
Input: $st = (\mathrm{stag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \cdots), \mathsf{EDB}, \mathsf{XSet}$
Output: R
 1: function SEARCH($st$, EDB, XSet)
 2:     $\mathrm{R} \leftarrow \{\}$
 3:     for $\mathrm{stag} \in \mathrm{stags}$ do
 4:         $c \leftarrow 1; \ell \leftarrow F(\mathrm{stag}, c)$
 5:         while $\ell \in \mathsf{EDB}$ do
 6:             $(e, y) \leftarrow \mathrm{EDB}[\ell]$
 7:             if $\mathsf{xtoken}[c, i]^y \in \mathsf{XSet}$ for all $i$ then
 8:                 $R \leftarrow R \cup \{e\}$
 9:             end if
10:             $c \leftarrow c + 1; \ell \leftarrow F(\mathrm{stag}, c)$
11:         end while
12:     end for
13:     return R
14: end function
```

are leaked to the server during the second step of our Retrieve procedure, which in fact can be avoided by deploying e.g., blind storage in [26]. In this work, we are mainly concerned with search on encrypted indices, for more details about blind storage please refer to [26].

5 Security Analysis

In this section, we show the security of our protocol against the adaptive server and the client one after another. Similar to [7], we first give a proof of security against non-adaptive attacks w.r.t. server, and further discuss the proof of full security later. As to the security w.r.t. client, we use a slight variant of security definition where the goal of the adversarial client is to generate a new valid private key.

Theorem 1. *Our scheme Π is $\mathcal{L}$-semantically secure against non-adaptive attacks where $\mathcal{L}$ is the leakage function defined as before, assuming that the DDH assumption holds in $\mathbb{G}$, that F and F_p are secure PRFs and that* ABE *is a CPA secure attribute-based encryption.*

Theorem 2. *Our scheme Π is secure against malicious clients, i.e., search token in Π is unforgeable against adaptive attacks, assuming that the strong RSA assumption holds.*

Theorem 3. *Let $\mathcal{L}$ be the leakage function defined before, our scheme Π is $\mathcal{L}$-semantically secure against adaptive attacks, assuming that the DDH assumption holds in $\mathbb{G}$, that F and F_p are secure PRFs and that* ABE *is a CPA secure attribute-based encryption.*

We remark that for lack of space, we omit the detailed proofs here, which will be given in the full version.

6 Further Extension

For sake of simplicity, we only presented our protocol and its security analysis for the case of conjunctive queries. Similar to [7,16], our protocol can also be readily adapted to support such form of boolean queries "$w_1 \wedge \psi(w_2, \ldots, w_m)$", where ψ is a boolean formula over the keywords $(w_2, \ldots, w_m)$ and w_i belongs to the client's permitted keyword set $\mathbf{w}$. In this case, the client calculates the stag corresponding w_1 and the xtoken for the other keywords and forwards the search token $(\mathrm{stag}, \mathsf{xtoken})$ and the boolean formula ψ to the server. Then the server uses stag to retrieve the tuples (e, y) containing w_1. The only difference from the conjunctive case for the server is the way he determines which tuples match the sub-boolean query ψ. For the t-th tuple, instead of checking if $\mathsf{xtoken}[c, i]^y \in \mathsf{XSet}$ for all $2 \leq i \leq m$, the server will set a series of boolean variables $v_2, \ldots, v_m$ such that

$$v_i = \begin{cases} 1, & \mathsf{xtoken}[c, i]^y \in \mathsf{XSet} \\ 0, & \text{otherwise} \end{cases},$$

and evaluate the value of $\psi(v_2, \ldots, v_m)$. If it is true, meaning the tuple matches the query, the server returns the encrypted index e. Clearly, the search complexity for such boolean queries is still $\mathcal{O}(|\mathsf{DB}[w_1]|)$, the same as for conjunctive queries. For the same set of keywords, the leakage information to the server for boolean case is also the same as for the conjunctive case, except that the boolean formula ψ is exposed to the server too. Hence, the proof for this case can also be readily adapted. For the support of other boolean queries, please refer to the details of [7].

7 Security and Performance Comparison

In general, we focus on the privacy of data owner in (multi-client) searchable encryption settings. In some scenarios, however, the clients may not want the data owner to get the information about the search queries they made or hope that the data owner learns as little as possible about the queries performed by themselves.

To achieve the additional property mentioned above, Jarecki et al. [16] further augmented their multi-client SSE to the outsourced private information retrieval (OSPIR) setting. Same as the underlying protocol, the enhanced protocol OSPIR still requires the clients to interact with the data owner and to submit each boolean formula for each boolean query, although it enables to hide the exact queried values from data owner. Our initial goal is to avoid the interaction between the data owner and the clients, but we also succeed to protect the privacy of the clients to some extent. More precisely, the data owner in our multi-client SE only knows the queried values belong to the keyword set that is authorized by the data owner according to the client's credentials at the beginning, but he has no means to learn what kind of queries the client made. Moreover, he cannot learn the exact queried values of the search. Therefore, our multi-client SSE also enjoys some additional nice security features.

In contrast to previous works such as [7,16], we further enforce the security of documents by employing CP-ABE to encrypt the document identifiers and retrieval decryption keys, by which our protocol realizes the fine-grained access control on the documents at the same time. In this case, even though the client can retrieve many encrypted indices, she still cannot learn the matching document identifiers and retrieval keys if her attributes do not satisfy the access policy associated with the ciphertext (encrypted index). Regarding the leakage information learned by the server, it is easy to observe that our protocol is exactly the same as [7,16].

Both our protocol and MULTI [16] are based on the CASH [7], but they rely on different methods and have distinct features. Compared to MULTI, our protocol manages to avoid the interaction between the data owner and the client, except at the beginning the client gets a search-authorized private key for some permitted keyword set. Moreover, as discussed before, we achieve the fine-grained access control on the stored documents by leveraging the ABE technique. Identical to MULTI, our protocol also supports any boolean queries. All the functionality features are summarized in Table 2.

Table 2. Functionality analysis

Reference	Query-type	Multi-user	Interaction[a]	Access control
Cash et al. [7]	Boolean	No	-	No
Jarecki et al. [16]	Boolean	Yes	Yes	No
Our scheme	Boolean	Yes	No	Yes

[a]The interaction needed between the data owner and the clients whenever a client performs search queries.

In the above, we give a brief security and function analysis of our protocol and a comparison with the representative multi-client SSE in [16] (MULTI). Next, we continue to analyze the efficiency of our protocol. Due to the fact that both our protocol and the MULTI are under the framework of CASH, the communication overhead between the data owner and the server (mainly contributed by $(\mathsf{EDB}, \mathsf{XSet})$ during the setup phase) and that between the client and the server (mainly contributed by $(\mathrm{stag}, \mathsf{xtoken})$ during the search phase) are almost identical, except that in our protocol document identifiers are encrypted via ABE instead of symmetric encryption. Beside the storage overhead introduced by the ABE ciphertext, using ABE also brings some computational cost to the data owner in contrast to exploiting symmetric encryption. In addition, the data owner needs to compute one extra exponentiation (i.e., the RSA function) for each calculation of the PRF during the setup phase, totally introducing $(2\sum_{w \in \mathrm{W}} |\mathsf{DB}[w]| + |\mathrm{W}|)$ exponentiation operations for the whole database. Fortunately, the encrypted database $(\mathsf{EDB}, \mathsf{XSet})$ are outsourced to the server once and forever, hence in this part we focus on analyzing the communication overhead between the data owner and the client as well as their computational cost introduced by the frequent search queries.

For a conjunctive query, e.g., $Q = (w_1 \wedge w_2 \wedge \cdots \wedge w_m)$ performed by a client, we assume that the associated keywords belong to the client's authorized keyword set $\mathbf{w}$, i.e., $w_i \in \mathbf{w}$ for $i \in [m]$. To perform such a search, the client in [16] has to interact with the data owner each time and gets the corresponding trapdoor information and authentication information, where the data owner needs to calculate $(m-1)$ exponentiations and an authenticated encryption. In contrast, the client in our protocol only needs to get from the data owner some keyword-related (and attribute-related) secret information at the beginning, where the data owner needs to computes 3 exponentiations and generates an attribute-related secret key for each client, and then she can perform the following searches by herself at the cost of introducing $(m+1)$ additional exponentiations to the generation of xtoken. Note that following our approach the client needs not to intact with the data owner ever after receiving her secret key because she can use the keyword-related part to generate the search tokens by herself only if she performs a query complying to the authorized keyword set. Therefore, once the data owner in our protocol outsourced his data to the server, he needs not to be online all the time. Precisely, the communication (comm.) overhead and the computational (comp.) cost w.r.t. the data owner and the client during each query are summarized in Table 3. We remark that in the table we only focus on the main comm. overhead and comp. cost contributed by the queried keywords, and omit the less contributed part, e.g., $\mathsf{AuthEnc}$ in [16] and $\mathsf{ABE.KeyGen}$ (which is only computed once for each client) in our protocol.

Table 3. Communication overhead between client and data owner & their computational cost

Conjunctive query $Q = (w_1 \wedge w_2 \wedge \cdots \wedge w_m)$, where $w_i \in \mathbf{w}$			
Reference	Comm. overhead	Data owner's comp. cost	Clients' comp. cost
Cash et al. [7]	-	$\lvert\mathsf{DB}[w_1]\rvert(m-1)\cdot \exp$	-
Jarecki et al. [16]	$(m-1)\lvert\mathbb{G}\rvert$	$(m-1)\cdot \exp$	$\lvert\mathsf{DB}[w_1]\rvert(m-1)\cdot \exp$
Our scheme	$3\lvert\mathbb{Z}_n^*\rvert$	$3\cdot \exp$	$(\lvert\mathsf{DB}[w_1]\rvert(m-1)$ $+(m+1))\cdot \exp$

exp: the exponentiation operation on the group; $|\cdot|$: the size of a finite set or group, e.g., $|\mathbb{G}|$; $\mathbf{w}$: the authorized keyword set for a client.

It is easy to see from this table the communication complexity of our protocol for each conjunctive query is $\mathcal{O}(1)$, even taking into account of all the other part of the private key, e.g., the attribute-related key sk_S, and that of Jarecki et al. [16] is $\mathcal{O}(m)$. Moreover, when the client performs k conjunctive queries, which are assumed comply to her authorized keyword set $\mathbf{w}$, the complexity of our protocol remains the same but that of [16] is $\mathcal{O}(k \cdot m)$, which increases linearly with the number of legitimate queries.

8 Conclusions

In this paper we present a new efficient multi-client searchable encryption protocol based on the RSA function. Our protocol avoids the per-query interaction between the data owner and the client, which decreases their communication overhead significantly. Meanwhile, our protocol can protect the privacy of the client to some extent. Precisely, the data owner in our protocol only knows the permitted search keyword set of the client, but has no means to learn the exact type of search queries or documents. Moreover, by employing attribute-based encryption, our protocol realizes fine-grained access control on the stored data. Support for searchability and access control simultaneously is actually a desirable feature in the practical data sharing scenarios. However, our current protocol only allows one data owner to share his data with many clients. We leave as an open problem to construct a system with the same advantages of ours while also support multi-data owner setting.

Acknowledgement. This work is supported by National Natural Science Foundation of China (61472083) and Australian Research Council (ARC), ARC Discovery Grant DP150100285.

References

1. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: CCSW 2013, Berlin, Germany, 4 November, pp. 77–88 (2013)
2. Baek, J., Vu, Q.H., Liu, J.K., Huang, X., Xiang, Y.: A secure cloud computing based framework for big data information management of smart grid. IEEE Trans. Cloud Comput. **3**(2), 233–244 (2015)
3. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. LNCS, vol. 4991, pp. 71–85. Springer, Heidelberg (2008)
4. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE S&P 2007, Oakland, California, USA, 20–23 May 2007, pp. 321–334 (2007)
5. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
6. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Dynamic searchable encryption in very-large databases: data structures and implementation. In: NDSS 2014, San Diego, California, USA, 23–26 February 2014
7. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013)
8. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 577–594. Springer, Heidelberg (2010)

9. Chu, C., Zhu, W.T., Han, J., Liu, J.K., Xu, J., Zhou, J.: Security concerns in popular cloud storage services. IEEE Pervasive Comput. **12**(4), 50–57 (2013)
10. Cramer, R., Shoup, V.: Signature schemes based on the strong RSA assumption. In: ACM CCS 1999, Singapore, 1–4 November 1999, pp. 46–51 (1999)
11. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM CCS 2006, Alexandria, VA, USA, 30 October–3 November 2006, pp. 79–88 (2006)
12. Dong, C., Russello, G., Dulay, N.: Shared and searchable encrypted data for untrusted servers. In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 127–143. Springer, Heidelberg (2008)
13. Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 123–139. Springer, Heidelberg (1999)
14. Goh, E.: Secure indexes. IACR Cryptology ePrint Archive, 2003:216 (2003)
15. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
16. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: ACM CCS 2013, pp. 875–888. ACM (2013)
17. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: FC 2013, Okinawa, Japan, 1–5 April 2013, pp. 258–274 (2013)
18. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM CCS 2012, Raleigh, NC, USA, 16–18 October 2012, pp. 965–976 (2012)
19. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: FC 2012, Kralendijk, Bonaire, 27 Februray–2 March 2012, pp. 285–298 (2012)
20. Liang, K., Au, M.H., Liu, J.K., Susilo, W., Wong, D.S., Yang, G., Phuong, T.V.X., Xie, Q.: A dfa-based functional proxy re-encryption scheme for secure public cloud data sharing. IEEE Trans. Inf. Forensics Secur. **9**(10), 1667–1680 (2014)
21. Liang, K., Liu, J.K., Wong, D.S., Susilo, W.: An efficient cloud-based revocable identity-based proxy re-encryption scheme for public clouds data sharing. In: Kutyłowski, M., Vaidya, J. (eds.) ICAIS 2014, Part I. LNCS, vol. 8712, pp. 257–272. Springer, Heidelberg (2014)
22. Liang, K., Susilo, W., Liu, J.K.: Privacy-preserving ciphertext multi-sharing control for big data storage. IEEE Trans. Inf. Forensics Secur. **10**(8), 1578–1589 (2015)
23. Liu, J., Huang, X., Liu, J.K.: Secure sharing of personal health records in cloud computing: ciphertext-policy attribute-based signcryption. Future Gener. Comp. Syst. **52**, 67–76 (2015)
24. Liu, J.K., Liang, K., Susilo, W., Liu, J., Xiang, Y.: Two-factor data security protection mechanism for cloud storage system. IEEE Trans. Comput. **65**(6), 1992–2004 (2016)
25. Miller, G.L.: Riemann's hypothesis and tests for primality. J. Comput. Syst. Sci. **13**(3), 300–317 (1976)
26. Naveed, M., Prabhakaran, M., Gunter, C.A.: Dynamic searchable encryption via blind storage. In: IEEE SP 2014, Berkeley, CA, USA, 18–21 May 2014, pp. 639–654 (2014)
27. Popa, R.A., Zeldovich, N.: Multi-key searchable encryption. IACR Cryptology ePrint Archive 2013:508 (2013)
28. Rabin, M.O.: Probabilistic algorithm for testing primality. J. Number Theor. **12**(1), 128–138 (1980)

29. Raykova, M., Vo, B., Bellovin, S.M., Malkin, T.: Secure anonymous database search. In: CCSW 2009, Chicago, IL, USA, 13 November 2009, pp. 115–126 (2009)
30. Van Rompay, C., Molva, R., Önen, M.: Multi-user searchable encryption in the cloud. In: López, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 299–316. Springer, Heidelberg (2015)
31. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, New York (2008). Also available on the Internet
32. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P 2000, Berkeley, California, USA, 14–17 May 2000, pp. 44–55 (2000)
33. Yang, Y., Lu, H., Weng, J.: Multi-user private keyword search for cloud computing. In: CloudCom 2011, Athens, Greece, 29 November–1 December 2011, pp. 264–271 (2011)