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], Tsz Hon Yuen[3]

[1]Shanghai Jiao Tong University, China
E-mail: crypto99@sjtu.edu.cn
[2]Faculty of Information Technology, Monash University, Australia
E-mail: {joseph.liu,amin.sakzad,ron.steinfeld}@monash.edu
[3] Huawei, Singapore
E-mail: 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 volumne 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. Simple encryption technology can protect data confidentiality easily. However, it is not possible to search within the encrypted domain. In order to search for a particular keyword, user has to decrypt the data first, before starting the searching process. It is not practical especailly when the volumne of data is large. Searchable encryption (SE) [27,5,10,8,7] 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 (and send it to the server) 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 [17]. 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 [17] (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 can get the document identifier. In our construction, we totally eliminate the interactive process, except at the beginning the client needs to obtain a search-authorized secret key from the data owner for some permitted keywords. After that, the client can generate a search token from this secret key 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 that 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 the client to perform arbitrary boolean queries efficiently.

1.2 Related Works

The first searchable encryption by Song et al. [27] is presented in the single-writer/single-reader setting. The first notion of security for searchable encryption was introduced by Goh [13]. Curtmola et al. [10] proposed the strong security notion of IND-CKA2. Kurosawa and Ohtaki [20] 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. [19,18] proposed dynamic searchable encryption schemes which allow efficient update of the database. Golle et al. [14] gave the first searchable encryption with conjunctive keyword searches, in the single-writer/single-reader setting. The search time is linear in the number of keywords to search. Most recently, Cash et al. [7] proposed the first sublinear searchable encryption with support for boolean queries and efficiently implemented it in a large database [6].

In the single-writer/multi-reader architecture, Curtmola et al. [10] proposed a general construction, which uses broadcast encryption on top of a single-reader scheme. The search time of the scheme by Raykova et al. [24] is linear in the number of documents. The scheme uses deterministic encryption and directly leaks the search pattern in addition to the access pattern. Jarecki et al. [17] extend the scheme by Cash et al. [7] to a single-writer/multi-reader setting while preserving all nice features provided by the original scheme.

In the multi-writer/multi-reader setting, a number of schemes [3,11,29,1] achieved a high level of security. The search time is linear in the number of keywords per document. The scheme in [22] improved the search complexity by removing the need of TTP in previous schemes. A stronger model for access pattern privacy was proposed in [25]. All these schemes only support single keyword search.

2 Preliminaries

In the section, we give a list of notations and terminologies used through our work and a brief review of hardness assumptions and cryptographic primitives deployed in our construction.

2.1 Notations

Table 1. Notations used in our work

Notation	Meaning
1^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, which is used to retrieve the original documents
$\mathcal{U}$	the attribute universe of the system
$[T]$	the set of positive integers less than T, i.e., $\{1, 2, \ldots, T\}$
$s \xleftarrow{\$} S$	the operation of uniformly sampling a random element s from S
sterm	the least frequent term among the queried terms/keywords in a search query
xterm	other queried terms in a search query (i.e., the queried terms excluding the sterm)

2.2 Hardness Assumptions

The security of our construction relies on the hardness of the DDH problem and the strong RSA problem [9], which are formally defined as follows.

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 probabilistic polynomial time (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). *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 algoritm $\mathcal{S}$ solves the strong RSA problem if it receives as input the tuple (n, g) and outputs two elements (z, e) such that $z^e = g \bmod n$.*

2.3 Pseudorandom Functions

Let $F : \{0,1\}^\kappa \times \mathcal{X} \to \mathcal{Y}$ 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 $Adv_{F,\mathcal{A}}^{\mathrm{prf}}(\kappa)$ defined as,

$$Adv_{F,\mathcal{A}}^{\mathrm{prf}}(\kappa) = |\Pr[\mathcal{A}^{F(K,\cdot)}(1^\kappa)] - \Pr[\mathcal{A}^{f(\cdot)}(1^\kappa)]|,$$

is negligible in κ, where $K \xleftarrow{\$} \{0,1\}^\kappa$ and f is a random function from $\mathcal{X}$ to $\mathcal{Y}$.

2.4 Attribute-based Encryption (ABE)

Attribute-based Encryption (ABE) [15,4,28,16,2] can be broadly categorized into key policy ABE (KP-ABE) and ciphertext policy ABE (CP-ABE). KP-ABE allows data to be encrypted with a set of *attributes*, and each decryption key is associated with an *access policy* (defined in terms of attributes); while CP-ABE is complementary – data are encrypted and tagged with the pre-determined access policy, and a decryption key is associated with the set of *attributes*. In either type, a ciphertext can be decrypted using the corresponding decryption key only if the attributes satisfy the access policy. ABE has been shown to be an effective and scalable access control mechanism for encrypted data. In our construction, we deploy CP-ABE as a primitive.

In general, a CP-ABE consists of the following algorithms ABE=($\mathsf{ABE.Setup}$, $\mathsf{ABE.KeyGen}$, $\mathsf{ABE.Enc}$, $\mathsf{ABE.Dec}$):

- $\mathsf{ABE.Setup}(\kappa)$: this algorithm takes no input other than the security parameter κ and outputs the public parameters mpk and a master secret key msk.
- $\mathsf{ABE.KeyGen}(mpk, msk, S)$: this algorithm takes as input a set of attributes S, the master key msk and the public parameters mpk and outputs a decryption key sk.
- $\mathsf{ABE.Enc}(mpk, m, \mathbb{A})$: this algorithm takes as input a message m, an access structure $\mathbb{A}$ and the public parameters mpk. It outputs the ciphertext ct such that only a user possessing a set of attributes that satisfies the access structure will be able to decrypt the message. The ciphertext is assumed to implicitly contains $\mathbb{A}$.
- $\mathsf{ABE.Dec}(sk, ct)$: this algorithm takes a ciphertext ct which contains an access structure $\mathbb{A}$ and a private key sk which is associated with a set of attributes S, and recovers the message m if $S \in \mathbb{A}$.

For the standard semantic security of CP-ABE, please refer to [4,28].

2.5 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 multi-client searchable encryption consists of the following algorithms:

- $\mathsf{EDBSetup}(1^\kappa, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$: the data owner takes 1^κ, $\mathsf{DB}, \mathrm{RDK}$ and $\mathcal{U}$ as input and generates the system master key MK and public key PK, with which it 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 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 performs 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 identifier and decryption key.

The goal of the data owner is to outsource his storage and service to the cloud server while leaking as little as possible information about the queries and plaintext data to the server, and preventing the clients from performing any search query over unpermitted keywords, which is formalized in the following subsection.

2.6 Security Definitions

In this section, we give security definitions of 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. an 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 $(\mathrm{MK}, \mathrm{PK}, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathsf{EDBSetup}(1^\kappa, \mathrm{DB}, \mathrm{RDK}, \mathcal{U})$ 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 database 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* $|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa)]| \leq negl(\kappa)$.

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, 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:

- $N = \sum_{i=1}^{d} |W_{id_i}|$ is the number of keyword-document pairs. This 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 $\mathbf{s}[t] \neq \mathbf{x}[t, \alpha]$. It 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 results 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].
- $\mathrm{XT}[t] = |\mathbf{x}[t]|$: the number of xterms in the t-th query.

The leakage function for our protocol is similar to that in [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 that they represent, because the server in the actual 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 his attributes/credentials. Thus, each client is only permitted to proceed search queries for 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 (that can select the authorized keywords by himself). That is, the malicious client is not allowed to gain information beyond what he 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 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 according to the generation of 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 un-authorized 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 this 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 user. 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 [12].

Construction. The main idea is to use the randomized hash function introduced in [12] 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 2κ-bit strings $\{0,1\}^{2\kappa}$.
- 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 picking a random function $\bar{h}$ from the collision-resistant family $\bar{\mathcal{H}}$ and a random 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.

Algorithm 1 h: Hashing from keywords to primes

```
Input: keyword $x \in \mathrm{W}$, functions $\bar{h} : \mathrm{W} \to \{0,1\}^\kappa \in \bar{\mathcal{H}}$, $F_k : \{0,1\}^\kappa \to \{0,1\}^\kappa \in \mathcal{F}$
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 $w$.
```

Security and Efficiency. The following statement summarizes the security and complexity of evaluating our hash family construction.

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 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* $\exp(-m)$.

Proof. The proof of (1) is immediate from the collision-resistance of $\bar{\mathcal{H}}$ and the fact that $\bar{h}(x)$ forms the κ MS bits of w. To prove (2), fix $x \in \mathrm{W}$ and let p_x denote the probability that a uniformly random integer in the interval $[2^\kappa \cdot \bar{h}(x), 2^\kappa \cdot \bar{h}(x) + 2^\kappa - 1]$ is prime, and let L denote the number of iterations of the while loop in algorithm 1 on input x. Assume for the moment that F_k is a perfect random function, so that $\mathsf{Int}(F_k(\mathsf{Bin}(r)))$ is an independent uniformly random integer in $[0, 2^\kappa - 1]$ for each $r \in [0, 2^\kappa - 1]$. Under this assumption, we have for each $n \geq 1$ that L is geometrically distributed as $\Pr[L = n] = (1 - p_x)^{n-1} \cdot p_x$, and it follows that $\Pr[L > n] = \sum_{t=n+1}^{\infty} (1 - p_x)^{t-1} \cdot p_x = (1 - p_x)^n$. Hence $L \leq m/p_x$, except with probability $p_{err} = (1 - p_x)^{m/p_x} \leq \exp(-m)$, where we have used the fact that $1 - p_x \leq \exp(-p_x)$ for $p_x \geq 0$. If we now replace the perfectly random F_k with the pseudorandom F_k, the probability p_{err} can exceed $\exp(-m)$ by at most a negligible amount (in the security parameter κ), or else we obtain an efficient distinguisher against the pseudorandomness of $\mathcal{F}$. To complete the proof of (2), we use our assumption on the density of primes in the interval $[2^\kappa \cdot \bar{h}(x), 2^\kappa \cdot \bar{h}(x) + 2^\kappa - 1]$, which implies that $p_x \geq \frac{1}{1.4 \cdot \kappa}$ for all x. $\square$

We note that the heuristic assumption in Lemma 1 part (2) about the distribution of primes is consistent for large κ with what one expects from the Prime Number Theorem, and is only needed for our efficiency estimates for the hash function evaluation, not for its collision-resistance security (if one wants to, this heuristic assumption can also be avoided to give a provable similar complexity, using a more complex hash function algorithm, as in [12]).

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 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 $T_{\exp}(2\kappa)$ 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 [21,23], the expected cost [26][Ch. 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 $T_h \leq (\kappa/2 + 2 \cdot L) \cdot T_{\exp}(2\kappa)$. 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 $L_{\mathrm{MR}} \approx (\prod_{\text{prime } p \leq 101} \frac{p-1}{p}) \cdot L \leq 0.11 \cdot L$. Thus, the overall expected time for evaluating our hash function would be

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

Thus, for a typical security parameter $\kappa = 100$, we estimate T_h to be equivalent to about $150 \cdot T_{\exp}(200)$ (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 $T_{\exp}(\kappa)$ for an exponentiation modulo a κ-bit modulus is, assuming classical arithmetic, at least quadratic in κ, we have $T_{\exp}(\lambda)/T_{\exp}(2\kappa) = T_{\exp}(2048)/T_{\exp}(200) \geq (2048/200)^2 \approx 104$, so the cost of evaluating our hash function for w is expected to be only $T_h \approx 150/104 \cdot T_{\exp}(2048) \approx 1.44 \cdot T_{\exp}(2048)$, 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 searchable encryption scheme which mainly consists of four algorithms $\Pi = (\mathsf{EDBSetup}, \mathsf{ClientKGen}, \mathsf{TokenGen}, \mathsf{Search})$. For completeness, we also give the description of document retrieval algorithm $\mathsf{Retrieve}$, by which the client finally retrieves the desired document from the cloud server. In the construction, 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 Section 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
Output: EDB, XSet
 1: function EDBGen(MK, PK, DB)
 2:     $\mathsf{EDB} \leftarrow \{\}$; $\mathsf{XSet} \leftarrow \emptyset$
 3:     for $w \in \mathrm{W}$ do
 4:         $c \leftarrow 1$; $\mathrm{stag}_w \leftarrow F(K_S, g_1^{1/w} \mod n)$
 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:             $\mathrm{xind} \leftarrow F_p(K_I, id)$; $z \leftarrow F_p(K_Z, g_2^{1/w} \mod n || c)$
 8:             $y \leftarrow \mathrm{xind} \cdot z^{-1}$; $\mathrm{xtag} \leftarrow g^{F_p(K_X,\ g_3^{1/w} \mod n) \cdot \mathrm{xind}}$
 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
```

$\mathsf{ClientKGen}(MK, S, \mathbf{w})$: 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 $\mathcal{D}$ 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

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

At last, $\mathcal{D}$ sends back sk together with $\mathbf{w}$, implicitly assuming that the keyword appearance frequency satisfies $|w_1| < |w_2| < \cdots < |w_n|$.

$\mathsf{TokenGen}(sk, Q)$: whenever the client $\mathcal{C}$ wants to search a boolean query Q with keywords $\bar{\mathbf{w}} \subseteq \mathbf{w}$, he 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 xtokens) for this query is computed as Algorithm 3.

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

$\mathsf{Search}(st, \mathsf{EDB}, \mathsf{XSet})$: 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 Algorithm 4.

Algorithm 4 Search Algorithm

```
Input: $st = (\mathrm{stag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \cdots), \mathsf{EDB}$
Output: R
 1: function Search($st, \mathsf{EDB}$)
 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
```

$\mathsf{Retrieve}(sk, \mathrm{R})$: the client with private key sk_S decrypts the encrypted index (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 identified by 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.

5 Security Analysis

In this section, we show the security of our scheme against the adaptive server and client respectively. 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.*

Proof. The proof is conducted via a sequence of Games. In all games, the adversary supplies a database DB and all search queries $\mathbf{q}$ at the beginning of the game. The first game Game_0 is designed to have the same distribution as $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa)$ (assuming no false positives), and the last game Game_8 is designed to be easily simulated by an efficient simulator $\mathcal{S}$. By showing that the distributions of all games are (computationally) indistinguishable to each other, we can see that the simulator $\mathcal{S}$ meets the requirements of the security definition w.r.t. server, thus completing the proof of the theorem. Our proof differs from that of [7] by not including a game equivalent to their G_6. This is because we use a dictionary instead of their TSet data structure, which we handle in earlier games. Their TSet is a specification of a hash table with an API designed for their purposes. We use a standard dictionary instead for familiarity to readers and to save specifying additional data structures and leakages. Moreover, for sake of simplicity we model a slightly different version of the protocol where the client always sends T_c (some publicly known upper bound) xtoken arrays for each query, instead of having the server interactively tell the client when to stop. However the proof could be easily generalized to the latter case.

Game$_0$: this Game is slightly modified from the real Game with some minor changes to make the analysis easier, the details of which is shown in Algorithm 5. With $(\mathsf{DB}, \mathbf{w}, \mathbf{s}, \mathbf{x})$ as input, the Game starts to simulate the encrypted database $(\mathsf{EDB}, \mathsf{XSet})$ by running Initialize, which is identical to $\mathsf{EDBSetup}$ in Algorithm 2 except that the calculation of XSet is separated as a single function, XSetSetup, to assist in the presentation of changes in the following Games. Then Initialize generates the transcript using the TransGen function, as defined in 5. Specifically, Initialize first computes the private key $\mathsf{sk}_{\mathbf{w}}$ corresponding to the authorized keywords $\mathbf{w}$. To calculate the transcript array $\mathbf{t}$, for $t \in [T]$, it then lets $\mathbf{t}[t]$ be the output of TransGen$(\mathsf{EDB}, \mathsf{XSet}, \mathsf{sk}_{\mathbf{w}}, K_S, K_X, K_Z, \mathbf{s}[t], \mathbf{x}[t, \cdot])$, which generates a transcript as in the real Game except that it computes ResInds in a different way: it directly looks up the result corresponding to the query instead of decrypting the returned ciphertexts Res (concretely, it calculates $\mathsf{DB}(\mathbf{s}_t)$ and then filters the values that are also in $\mathsf{DB}(\mathbf{x}_t)$, where $\mathbf{s}_t = \mathbf{s}[t]$, $\mathbf{x}_t = \mathbf{x}[t, \cdot]$ and $\mathsf{DB}(\mathbf{x}_t)$ denotes the intersection of all $\mathsf{DB}[\mathbf{x}[t, i]$ for $i = 2, 3, \ldots, n)$.

In addition, we also made the following minor bookkeeping change: the order in which the document identifiers are used for each keyword w are recorded in an array WPerms$[w]$. The order is chosen as a random permutation, which matches the real game.

Assuming no false positives happening, the distribution of the game is exactly the same as that of $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa)$. So we have,

$$\Pr[\mathrm{Game}_0 = 1] \leq \Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\kappa) = 1] + \mathrm{negl}(\kappa).$$

Game$_1$: this game is identical to the last one, except that the boxed codes are also included, the details of which are also given in Algorithm 5. More precisely, we record in this game the stag values after they are first computed and look them up on subsequent uses, rather than recomputing them: for each $t \in [T]$ it lets $\mathsf{query\_stag} \leftarrow \mathrm{stags}[\mathbf{s}[t]]$. According to the calculation of stags in the $\mathsf{TokenGen}$ algorithm, it is easy to observe that the distributions of these two games are identical. Thus, we have

$$\Pr[\mathrm{Game}_1 = 1] = \Pr[\mathrm{Game}_0 = 1].$$

Game$_2$: this game is the same as previous, except that we replaces the PRFs F and F_p with random functions, the details of which are shown in Algorithm 6. Note that since $F(K_S, \cdot)$ is only evaluated on the same input once, its evaluations can be replaced with random selections from the appropriate range. Regarding

$F_p(K_X, \cdot)$, $F_p(K_I, \cdot)$ and $F_p(K_Z, \cdot)$, they are replaced by f_X, f_I and f_Z respectively. By a standard hybrid argument, it is easy to see that there exist efficient adversaries $\mathcal{B}_{1,1}$ and $\mathcal{B}_{1,2}$ such that

$$\Pr[\text{Game}_1 = 1] - \Pr[\text{Game}_0 = 1] \leq \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}_{1,1}}(\kappa) + 3\mathbf{Adv}^{\text{prf}}_{F_p,\mathcal{B}_{1,2}}(\kappa).$$

Game$_3$: this game is almost the same as Game$_2$, except that it also includes the boxed code, as shown in Algorithm 6. This means that the encryption of document identifiers is always replaced with an encryption of the constant string 0^κ. Since the encryption is operated for polynomial, say $ploy(\kappa)$, times, by a standard hybrid argument we can see that there exists an efficient adversary $\mathcal{B}_2$ such that

$$\Pr[\text{Game}_2 = 1] - \Pr[\text{Game}_1 = 1] \leq ploy(\kappa) \cdot \mathbf{Adv}^{\text{IND-CPA}}_{\Sigma,\mathcal{B}_2}(\kappa).$$

Note that the reduction to the IND-CPA security of the underlying ABE scheme Σ is possible, because the ciphertexts need not to be decrypted. We omit the tedious details here.

Game$_4$: the only difference of this game from the previous is that the XSet and xtoken are generated in an alternative but equivalent way, which is shown in Algorithm 7. Loosely speaking, all possible values $\text{XSET\_ELEM}(w, \text{id}) = g^{f_X(g_3^{1/w} \mod n)\cdot f_I(\text{id})}$ for each identifier id and keyword $w \in W$ are pre-computed and stored in an array H. Moreover, some xtoken values in transcripts, which do not correspond to possible matches, are generated and stored in another array Y.

In this game, the function XSETSETUP is altered to use the values from the H array to produce the XSet elements: for a given $w \in W$ and $id \in \mathsf{DB}[w]$, it adds the value $H[id, w]$ to XSet. Recall that this value is set to be $g^{f_X(g_3^{1/w} \mod n)\cdot f_I(id)}$ during the INITIALIZE, which is the same as that in Game$_3$. In addition, it is easy to observe that each invocation of TRANSGEN here will return the same output as the previous game only if xtoken values are the same in both games. Hence, we only focus on the generation of xtoken array in the following.

In Game$_3$, xtoken is computed as in the real game. Specifically, the xtoken value $\mathsf{xtoken}[\alpha, c]$ for each xterm $\mathbf{x}_t[\alpha]$ and $c \in [T_c]$ is set to be $g^{f_Z\left((sk^{(2)}_{\mathbf{w}})^{\prod_{w \in \mathbf{w}\setminus\{\mathbf{s}_t\}} w} \mod n||c\right)\cdot f_X\left((sk^{(3)}_{\mathbf{w}})^{\prod_{w\in\mathbf{w}\setminus\mathbf{x}_t[\alpha]\}} w} \mod n\right)}$, which is equal to $g^{f_Z(g_2^{1/\mathbf{s}_t} \mod n||c)\cdot f_X(g_3^{1/\mathbf{x}_t[\alpha]} \mod n)}$. In this game, xtoken is generated in the following way. First, TRANSGEN looks up $(\text{id}_1, \ldots, \text{id}_{T_s}) \leftarrow \mathsf{DB}[\mathbf{s}_t]$ and $\sigma \leftarrow \text{WPerms}[\mathbf{s}_t]$. For each $\mathbf{x}_t[\alpha]$ and $c \in [T_c]$ it then retrieves $\mathsf{EDB}[l] = (e, y)$ using $\mathsf{query\_stag}$, where $y = f_I(\text{id}_{\sigma[c]}) \cdot (f_Z(g_2^{1/\mathbf{s}_t} \mod n||c))^{-1}$, and sets the value $\mathsf{xtoken}[\alpha, c]$ as:

$$\mathsf{xtoken}[\alpha, c] = \begin{cases} H[\text{id}_{\sigma[c]}, \mathbf{x}_t[\alpha]]^{1/y}, & c \in [T_s], \\ Y[\mathbf{s}_t, \mathbf{x}_t[\alpha], c], & c \in [T_c] \setminus [T_s]. \end{cases}$$

By a simple verification, we can see that $\mathsf{xtoken}[\alpha, c] = g^{f_Z(g_2^{1/\mathbf{s}_t} \mod n||c)\cdot f_X(g_3^{1/\mathbf{x}_t[\alpha]} \mod n)}$ for each $\mathbf{x}_t[\alpha]$ and $c \in [T_c]$, which indicates that the xtoken values in both games are exactly the same. Hence, we get

$$\Pr[\text{Game}_4 = 1] = \Pr[\text{Game}_3 = 1].$$

Game$_5$: this game is almost identical to Game$_4$, except that the single boxed code in Algorithm 7 is also included: the values y are now drawn randomly from $\mathbb{Z}_p^*$. Due to the modifications made in Game$_4$ and the fact that the mapping $w \to (g_2^{1/w} \mod n)$ is injective, the value of $f_Z(g_2^{1/w} \mod n||c)$ is used only once during INITIALIZE and it is uniform and independent of the rest of randomness in the game. Moreover, since $y = \text{xind} \cdot z^{-1}$ depends on the random value of f_Z, it is also uniformly and independently distributed. Thus, replacing y with random values does not affect the distribution of the game. Therefore, we have

$$\Pr[\text{Game}_5 = 1] = \Pr[\text{Game}_4 = 1].$$

Game$_6$: this game is exactly like the previous game, except that it also includes the doubly boxed code in Algorithm 7. That is, all the values of H and Y arrays are selected at random from $\mathbb{G}$. Under the DDH assumption, there exists an efficient algorithm $\mathcal{B}_6$ such that

$$\Pr[\text{Game}_6 = 1] - \Pr[\text{Game}_5 = 1] \leq \mathbf{Adv}^{\text{DDH}}_{\mathbb{G},\mathcal{B}_6}(\kappa).$$

To show the indistinguishability between these two games, a simple reduction can be conducted similarly as in [7]. Briefly speaking, the values of the X array in Game$_5$ are the g^a values, and the X values are raised to the power of xind when computing H and to the power of $f_Z(g_2^{1/w} \mod n||c)$ when computing Y, where xind and $f_Z(g_2^{1/w} \mod n||c)$ act as the b values of the DDH tuple. Thus, H and Y in Game$_5$ have values of the form

g^{ab}, while in Game$_6$ they are replaced with random values. Differentiating between them can be easily reduced to the DDH problem, we omit the details here.

In the last two games **Game**$_7$ and **Game**$_8$, we change the way the H array is accessed for enabling the final simulator to work with its given leakage. The details are shown in Algorithm 8. Briefly speaking, the access to H is reduced to the cases where H values will be used for multiple times. For other cases, the access to H will be replaced with a random choice. Since in the later cases, there would be only one chance for the game to access H, the random section will not affect the distribution of the game. Note that although Game$_7$ continues to use the TransGen function of Algorithm 7, we removed for simplicity the irrelevant code from the previous games, such as the selection of random functions.

Game$_7$: this game is almost identical to the above, except that we modify XSetSetup to only include the members of H that could actually be used or accessed for multiple times, as shown in Algorithm 8. Recall that after the array H is generated, it is only used in two subroutines: the function XSetSetup and TransGen. Clearly, the function XSetSetup will never repeat an access to H, so for an index (id, w) of H it only needs to check if this position will be accessed by TransGen. However, TransGen only accesses H for such positions that satisfy $\mathrm{id} \in \mathsf{DB}[\mathbf{s}_t]$ and $w = \mathbf{x}_t[\alpha]$ for some t and α. Apparently, this condition exactly captures the elements of H used for multiple times. Regarding the other positions, the corresponding elements cannot be distinguished from random choices. Thus, the modification does not change the distribution of the game. Hence, we get that

$$\Pr[\text{Game}_7 = 1] = \Pr[\text{Game}_6 = 1].$$

Game$_8$: the final game is exactly like the previous one, except that we change the way TransGen accesses H, so that it only computes the xtokens with the members of H that are accessed for multiple times, which is also shown in Algorithm 8. To test a possible reusage of a member (e.g., indexed by (id, w)) of H, we must check if either XSetSetup has access to this index, or if TransGen will access it again. In Game$_7$, XSetSetup was modified in such a way that it only accesses the member $H[\mathrm{id}, w]$ if $\in \mathsf{DB}[\mathbf{s}_t] \cap \mathsf{DB}[\mathbf{x}_t[\alpha]]$ and $w = \mathbf{x}_t[\alpha]$ for some t and α, which is captured by the first "if" statement in the TransGen of Game$_8$. However, it is also possible that TransGen uses the same member twice. We note that this happens only if it is called for two different queries because one execution of the subroutine only touches unique elements of H. For clearity, the current query number t is additionally passed in as an argument. More precisely, for an element indexed by (id, w) to be accessed twice, it must hold that $\mathrm{id} \in \mathsf{DB}[\mathbf{s}_t] \cap \mathsf{DB}[\mathbf{s}_{t'}]$ and $w = \mathbf{x}_t[\alpha] \in \mathbf{x}_{t'}$ for some $t' \neq t$. The condition for such a repeated access is exactly captured by the second "if" statement in TransGen of this game. If neither of these conditions apply, the xtoken is randomly selected from $\mathbb{G}$. Since all repeating elements of H in the previous are still used here, we have

$$\Pr[\text{Game}_8 = 1] = \Pr[\text{Game}_7 = 1].$$

Simulator: In the following, we give a simulator $\mathcal{S}$ that takes as input the leakage $\mathcal{L}(\mathsf{DB}, \mathbf{s}, \mathbf{x}) = (N, \bar{\mathbf{s}}, \mathrm{SP}, \mathrm{RP}, \mathrm{SRP}, \mathrm{IP}, \mathrm{XT})$ and outputs a simulated $(\mathsf{EDB}, \mathsf{XSet})$ and transaction array $\mathbf{t}$. By showing that the simulator produces the same distribution as Game$_8$ and combining the relations between the games, we will show that the simulator satisfies the requirements in the theorem. At the beginning, the simulator first computes a restricted equality pattern of $\mathbf{x}$ as below, denoted by $\hat{\mathbf{x}}$. Then it proceeds through algorithms 9 to 11 to produce its final output.

The restricted equality pattern $\hat{\mathbf{x}}$ can be computed because it is possible for the server to infer that certain xterms are equal based on potential XSet elements $\mathrm{XSet\_elem}(w, \mathrm{id})$. This leakage coming from these equalities is made precise in the IP structure. If there exist id, t_1 and t_2 such that $\mathrm{id} \in \mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[\mathbf{s}[t_2]]$, then it is possible to infer if two xterms $\mathbf{x}[t_1, \alpha]$ and $\mathbf{x}[t_2, \beta]$ are equal because there will be repeating values $\mathrm{XSet\_elem}(\mathbf{x}[t_1, \alpha], \mathrm{id})$ and $\mathrm{XSet\_elem}(\mathbf{x}[t_2, \beta], \mathrm{id})$. This can be formulated equivalently in terms of the leakage IP by defining a $T \times A$ table $\hat{\mathbf{x}}[t, \alpha]$ such that $\hat{\mathbf{x}}[t_1, \alpha] = \hat{\mathbf{x}}[t_2, \beta]$ iff $\mathrm{IP}[t_1, t_2, \alpha, \beta] \neq \emptyset$. The table $\hat{\mathbf{x}}$ describes which xterms are "known" to be equal by the adversarial server. In particular, we have that

$$\hat{\mathbf{x}}[t_1, \alpha] = \hat{\mathbf{x}}[t_2, \beta] \implies \mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]$$

and

$$(\mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]) \wedge (\mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[\mathbf{s}[t_2]] \neq \emptyset) \implies \hat{\mathbf{x}}[t_1, \alpha] = \hat{\mathbf{x}}[t_2, \beta].$$

In the simulator, $\hat{\mathbf{x}}$ is an array of integers and can be computed by incrementally numbering off the $\hat{\mathbf{x}}[t_1, \alpha]$ values but setting $\hat{\mathbf{x}}[t_2, \beta]$ to the value of $\hat{\mathbf{x}}[t_1, \alpha]$ when $(t_1, \alpha) < (t_2, \beta)$ and $\mathrm{IP}[t_1, t_2, \alpha, \beta] \neq \emptyset$.

The simulated EDB is produced by Algorithm 9. The main difference between Game$_8$ and the simulator code is that Game$_8$ fills out the entries of EDB for every $w \in W$ while the simulator only does it for $i \in \bar{\mathbf{s}}$. The entries of $\bar{\mathbf{s}}$ are integers that correspond to the sterms in the queries, so we may have $|\bar{\mathbf{s}}| < |W|$. The EDB must have N entries, so the simulator adds additional random entries until the correct size is reached. In both the simulator and the game, the dictionary keys are indistinguishable and the values are encryptions of 0^{κ} under the same keys, so the distribution of EDB is indistinguishable between these two cases.

The simulator XSet is produced by algorithm 10. We claim that it is distributed identically in Game$_8$ jointly consistent with the EDB and xtokens. First, it can be seen that both algorithms add N randomly chosen group elements to the XSet. In Game$_8$ this is done by looping through every keyword $w \in W$ and id $\in \mathsf{DB}[w]$, totally giving $\sum_{w \in W} \mathsf{DB}[w]$ additions. In the simulator, this is done by keeping track of each addition with a counter j and then adding additional elements until N elements have been added. What remains to be shown is that the distribution of the elements is consistent with the EDB and the xtokens.

To show this, we will consider how the xtokens are calculated. The simulated client-server transcript $\mathbf{t}$, including the xtokens, is produced by Algorithm 11. Clearly the y and σ values are distributed identically, both being uniformly random. In Game$_8$, the permutations σ are reused when an stag repeats. The reuse pattern in the simulator is the same, based on repeating values in $\bar{\mathbf{s}}$.

Next, we show that both Game$_8$ and the simulator perform equivalent access to H when calculating the xtokens. In Game$_8$, $H[\mathrm{id}, w]$ is accessed if either (1) id matches a conjunction within the query that uses w (i.e. $\exists \alpha$ s.t. id $\in \mathsf{DB}[\mathbf{s}_t] \wedge \mathbf{x}_t[\alpha] = w$) or (2) the value will be reused in a later query. The same reads are made by the simulator by only reading for identifiers in R, which is calculated as the identifiers that match (1) using RP and those that match (2) using IP.

Finally, we show that when H is used for multiple times at the same positions, the reusage is the same in both Game$_8$ and the simulator. That is, if two reads $(\mathrm{id}_1, \mathbf{x}[t, \alpha])$ and $(\mathrm{id}_2, \mathbf{x}[t, \alpha])$ are equal in Game$_8$, then the equivalent reads $(\mathrm{id}_1, \hat{\mathbf{x}}[t, \alpha])$ and $(\mathrm{id}_2, \hat{\mathbf{x}}[t, \alpha])$ are also equal in the simulator. Formally,

$$(\mathrm{id}_1, \mathbf{x}[t, \alpha]) = (\mathrm{id}_2, \mathbf{x}[t, \alpha]) \iff (\mathrm{id}_1, \hat{\mathbf{x}}[t, \alpha]) = (\mathrm{id}_2, \hat{\mathbf{x}}[t, \alpha]) \tag{1}$$

We have for $\hat{\mathbf{x}}$ that $\hat{\mathbf{x}}[t_1, \alpha] = \hat{\mathbf{x}}[t_2, \beta] \implies \mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]$, which gives the left direction " $\Leftarrow$". For the other direction "$\Rightarrow$", we instead use the other property of $\hat{\mathbf{x}}$, that is $(\mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]) \wedge (\mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[\mathbf{s}[t_2]] \neq \emptyset) \implies \hat{\mathbf{x}}[t_1, \alpha] = \hat{\mathbf{x}}[t_2, \beta]$. Thus, if $\mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[s[t_2]] \neq \emptyset$, then the equation 1 is proven. Given that $(\mathrm{id}_1, \mathbf{x}[t, \alpha]) = (\mathrm{id}_2, \mathbf{x}[t, \alpha])$, then $\mathrm{id}_1 = \mathrm{id}_2$. But then the intersection must contain at least this id and so is nonempty.

When calculating the ResInds value for the transcript, which are the document identifiers matching the query, the simulator finds these values using the RP and SRP. This is indicated in the code as a function Real_Results(RP, SRP). For each queried keyword, its matching identifiers can be taken from RP if it is in a conjunction with any other terms and from SRP if it is not. The set union and intersection of the leaf results can then be taken to find the final output. □

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.*

Proof. Due to the properties of PRFs, we can see from our scheme that no client can generate a valid search token for some non-authorized keyword e.g., w', unless he could correctly guess the value $(g_j^{1/w'} \mod n)$ for $j \in [3]$. Suppose that there exists an adversarial client $\mathcal{A}$ who can produce a valid search token for some non-authorized keyword w', implying that he can get the correct value $(g_j^{1/w'} \mod n)$ for some $j \in [3]$. In this case, we can use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ to solve the strong RSA problem with a non-negligible probability as follows.

Given a random strong RSA instance (n, h_j) where $h_j \leftarrow \mathbb{Z}_n^*$, the algorithm $\mathcal{B}$ invokes the system setup algorithm $\mathsf{EDBSetup}$ and returns the system public key PK to $\mathcal{A}$. After that, $\mathcal{A}$ submits a private key extraction query $\mathbf{w} = (w_1, \ldots, w_n)$ of his choosing along with his attribute set S. To respond, $\mathcal{B}$ sets $g_j = (h_j^{\prod_{i=1}^n w_i} \mod n)$, which implies $h_j = (g_j^{1/\prod_{i=1}^n w_i} \mod n)$, and runs the algorithm $sk_S \leftarrow \mathsf{ABE.KeyGen}(msk, S)$. At last, it sets $sk_{\mathbf{w}} = h_j$ and sends back the pair $(sk_S, sk_{\mathbf{w}})$ as the requested private key. Eventually, $\mathcal{A}$ outputs his guess v for some non-authorized keyword $w' \notin \mathbf{w}$. $\mathcal{B}$ then verifies the correctness (i.e., $v = (g_j^{1/w'} \mod n)$) of his guess by checking if $v^{w'} = g_j \mod n$. If so, $\mathcal{B}$ can solve the strong RSA instance as below:

Recall that keywords are mapped to different primes, so we have $gcd(\prod_{i=1}^{n} w_i, w') = 1$. By the extended Euclidean algorithm, we can find integers a, b such that $a(\prod_{i=1}^{n} w_i) + bw' = 1$, and then get the value $h_j^{1/w'} = (g_j^{1/w'})^a \cdot h_j^b = (h_j^{\prod_{i=1}^{n} w_i/w'})^a \cdot h_j^b \mod n$. At last, $\mathcal{B}$ returns the solution $(w', h_j^{1/w'})$.

Hence, from the brief analysis we can see that no client is able to generate a valid search token except with a negligible probability. □

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.*

Proof. The main idea for the proof of this theorem remains the same as that of [7]. Roughly speaking, to handle adaptivity, the simulator with input N chooses N random group elements and adds them to XSet. To simulate the response for each query, the simulator adaptively "assign" elements of the XSet to id-keyword pairs. This is in contrast to the non-adaptive simulator, where it first initializes the H array and then adds the elements to the XSet, as determined by the leakage. Now the simulator first chooses the XSet values, and then initializes H adaptively.

6 Further Extension

For sake of simplicity, we only presented our protocol and its security analysis for the case of conjunctive queries. Similar to [7,17], 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 (stag, 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 $2 \le i \le m$, 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 this 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 Analysis

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. [17] 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 [7,17], 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 [7,17], which is summarized in Table 2.

Table 2. Leakage Information

Leakage items	Cash et al. [7]	Jarecki et al. [17]	Our Scheme
$N = \sum_{i=1}^{d} \lvert W_i \rvert$	✓	✓	✓
$\bar{\mathbf{s}}$	✓	✓	✓
SP[σ]	✓	✓	✓
RP[t, α]=DB[**s**[t]]$\cap$DB[**x**[t, α]]	✓	✓	✓
SRP[t, i] = DB[s[t, i]]	✓	✓	✓
IP[t_1, t_2, α, β]	✓	✓	✓
XT[t] = $\lvert \mathbf{x}[t] \rvert$	✓	✓	✓

Both our protocol and MULTI [17] 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 3.

Table 3. Functionality Analysis

Reference	query-type	multi-user	interaction*	access control
Cash et al. [7]	Boolean	No	-	No
Jarecki et al. [17]	Boolean	Yes	Yes	No
Our scheme	Boolean	Yes	No	Yes

$*$: 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 [17] (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 $(\mathsf{stag}, \mathsf{xtoken})$ 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 [17] 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

Table 4. Communication overhead between client and data owner & their computation 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. [17]	$(m-1)\lvert \mathbb{G} \rvert$	$(m-1) \cdot \exp$	$\lvert \mathsf{DB}[w_1] \rvert (m-1) \cdot \exp$
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.

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 4. 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., AuthEnc in [17] and ABE.KeyGen (which is only computed once for each client) in our protocol.

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. [17] 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 [17] 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.

References

1. M. R. Asghar, G. Russello, B. Crispo, and M. Ion. Supporting complex queries and access policies for multi-user encrypted databases. In *CCSW'13, Berlin, Germany, November 4, 2013*, pages 77–88, 2013.
2. N. Attrapadung, J. Herranz, F. Laguillaumie, B. Libert, E. de Panafieu, and C. Ràfols. Attribute-based encryption schemes with constant-size ciphertexts. *Theor. Comput. Sci.*, 422:15–38, 2012.
3. F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In *ISPEC 2008, Sydney, Australia, April 21-23, 2008*, pages 71–85, 2008.
4. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE S&P '07, 20-23 May 2007, Oakland, California, USA*, pages 321–334, 2007.
5. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT 2004, Interlaken, Switzerland, May 2-6, 2004*, pages 506–522, 2004.
6. D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS '14, San Diego, California, USA, February 23-26, 2014*, 2014.
7. D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13, Santa Barbara, CA, USA, August 18-22, 2013*, pages 353–373, 2013.
8. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT 2010, Singapore, December 5-9, 2010. Proceedings*, pages 577–594, 2010.
9. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *ACM CCS '99, Singapore, November 1-4, 1999*, pages 46–51, 1999.

10. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS '06, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88, 2006.
11. C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *Data and Applications Security XXII, 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, London, UK, July 13-16, 2008, Proceedings*, pages 127–143, 2008.
12. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 123–139, 1999.
13. E. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
14. P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In *ACNS '04, Yellow Mountain, China, June 8-11, 2004*, pages 31–45, 2004.
15. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS '06, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 89–98, 2006.
16. S. Hohenberger and B. Waters. Attribute-based encryption with fast decryption. *IACR Cryptology ePrint Archive*, 2013:265, 2013.
17. S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM CCS'13*, pages 875–888. ACM, 2013.
18. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *FC 2013, Okinawa, Japan, April 1-5, 2013*, pages 258–274, 2013.
19. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM CCS '12, Raleigh, NC, USA, October 16-18, 2012*, pages 965–976, 2012.
20. K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *FC '12, Kralendijk, Bonaire, Februray 27-March 2, 2012*, pages 285–298, 2012.
21. G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300 – 317, 1976.
22. R. A. Popa and N. Zeldovich. Multi-key searchable encryption. *IACR Cryptology ePrint Archive*, 2013:508, 2013.
23. M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128 – 138, 1980.
24. M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Secure anonymous database search. In *CCSW 2009, Chicago, IL, USA, November 13, 2009*, pages 115–126, 2009.
25. C. V. Rompay, R. Molva, and M. Önen. Multi-user searchable encryption in the cloud. In *ISC 2015, Trondheim, Norway, September 9-11, 2015*, pages 299–316, 2015.
26. V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2008. Also available on the Internet.
27. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE S & P '00, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.
28. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC '11, Taormina, Italy, March 6-9, 2011*, pages 53–70, 2011.
29. Y. Yang, H. Lu, and J. Weng. Multi-user private keyword search for cloud computing. In *CloudCom '11, Athens, Greece, November 29 - December 1, 2011*, pages 264–271, 2011.

Algorithm 5 Game$_0$ and $\boxed{\text{Game}_1}$

```
function Initialize(DB, w, s, x)   // For each $t \in [T]$, the keywords associated with this query satisfy that $\mathbf{s}[t] \cup \mathbf{x}[t, \cdot] \subseteq \mathbf{w}$.
    $p, q, n, msk, mpk,\ g_1, g_2, g_3 \xleftarrow{\$} \mathbb{Z}_n^*,\ K_S, K_X, K_I, K_Z \xleftarrow{\$} \{0,1\}^\kappa$
    $\mathsf{EDB} \leftarrow \{\}$
    $\mathsf{XSet} \leftarrow \emptyset$
    $(\mathrm{id}_i, W_i) \leftarrow \mathsf{DB}$
    for $w \in W$ do
        $(\mathrm{id}_1, \ldots, \mathrm{id}_{T_w}) \leftarrow \mathsf{DB}[w]$
        $\sigma \xleftarrow{\$} \mathrm{Perm}([T_w])$
        $\mathrm{WPerms}[w] \leftarrow \sigma$
        $\mathrm{stag} \leftarrow F(K_S,\ g_1^{1/w} \mod n)$
        $\boxed{\mathrm{stags}[w] \leftarrow \mathrm{stag}}$
        for $c \in [T_w]$ do
            $l \leftarrow F(\mathrm{stag}, c)$
            $e \leftarrow \mathrm{ABE.Enc}(mpk, \mathrm{id}_{\sigma[c]} || k_{\mathrm{id}_{\sigma[c]}}, \mathbb{A})$
            $\mathrm{xind} \leftarrow F_p(K_I, \mathrm{id}_{\sigma[c]});\ z \leftarrow F_p(K_Z,\ g_2^{1/w} \mod n||c)$
            $y \leftarrow \mathrm{xind} \cdot z^{-1}$
            $\mathsf{EDB}[l] = (e, y)$
        end for
    end for
    $\mathsf{XSet} \leftarrow \mathrm{XSetSetup}(p, q, K_X, K_I, \mathsf{DB})$
    $\mathsf{sk}_{\mathbf{w}} \leftarrow \mathrm{ClientKGen}(p, q, \mathbf{w})$
    for $t \in [T]$ do
        $\mathsf{query\_stag} \leftarrow F(K_S,\ (\mathsf{sk}_{\mathbf{w}}^{(1)})^{\prod_{w \in \mathbf{w} \setminus \{\mathbf{s}_t\}} w} \mod n)$
        $\boxed{\mathsf{query\_stag} \leftarrow \mathrm{stags}[\mathbf{s}_t]}$
        $\mathbf{t}[t] \leftarrow \mathrm{TransGen}(\mathsf{EDB}, \mathsf{XSet}, \mathsf{sk}_{\mathbf{w}}, K_X, K_Z, \mathbf{s}[t], \mathbf{x}[t, \cdot], \mathsf{query\_stag})$
    end for
    return $(\mathsf{EDB}, \mathsf{XSet}, \mathbf{t})$
end function

function XSetSetup(p, q, K_X, K_I, DB)
    $(\mathrm{id}_i, W_i) \leftarrow \mathsf{DB}$ ; $\mathsf{XSet} \leftarrow \emptyset$
    for $w \in W$ and $\mathrm{id} \in \mathsf{DB}[w]$ do
        $\mathrm{xind} \leftarrow F_p(K_I, \mathrm{id})$
        $\mathrm{xtag} \leftarrow g^{F_p(K_X,\ g_3^{1/w} \mod n) \cdot \mathrm{xind}}$
        $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathrm{xtag}\}$
    end for
    return XSet
end function

function ClientKGen(p, q, w)
    $\mathsf{sk}_{\mathbf{w}} \leftarrow \emptyset$
    for $i \in [3]$ do
        $\mathsf{sk}_{\mathbf{w}}^{(i)} \leftarrow g_i^{1/\prod_{j=1}^{n} w_j} \mod n$
    end for
    $\mathsf{sk}_{\mathbf{w}} \leftarrow (\mathsf{sk}_{\mathbf{w}}^{(1)}, \mathsf{sk}_{\mathbf{w}}^{(2)}, \mathsf{sk}_{\mathbf{w}}^{(3)})$
    return $\mathsf{sk}_{\mathbf{w}}$
end function

function TransGen(EDB, XSet, sk_w, K_X, K_Z, s_t, x_t, query_stag)
    for $\alpha \in [|\mathbf{x}_t|]$ do
        for $c \in [T_c]$ do
            $\mathsf{xtoken}[\alpha, c] \leftarrow g^{F_p\left(K_Z,\ (sk_{\mathbf{w}}^{(2)})^{\prod_{w \in \mathbf{w} \setminus \{\mathbf{s}_t\}} w} \mod n||c\right) \cdot F_p\left(K_X,\ (sk_{\mathbf{w}}^{(3)})^{\prod_{w \in \mathbf{w} \setminus \{\mathbf{x}_t[\alpha]\}} w} \mod n\right)}$
        end for
    end for
    $\mathrm{Res} \leftarrow \mathrm{Search}(\mathsf{EDB}, \mathsf{XSet}, (\mathsf{query\_stag}, \mathsf{xtoken}))$
    $\mathrm{ResInds} \leftarrow \mathsf{DB}[\mathbf{s}_t, \mathbf{x}_t]$
    return $((\mathsf{query\_stag}, \mathsf{xtoken}), \mathrm{Res}, \mathrm{ResInds})$
end function
```

Algorithm 6 Game$_2$ and $\boxed{\text{Game}_3}$

```
function Initialize(DB, w, s, x)
    $p, q, n, msk, mpk,\ g_1, g_2, g_3 \xleftarrow{\$} \mathbb{Z}_n^*,\ f_X, f_I, f_Z \xleftarrow{\$} \mathrm{Fun}(\{0,1\}^\kappa, Z_p^*)$
    $\mathsf{EDB} \leftarrow \{\}$
    $\mathsf{XSet} \leftarrow \emptyset$
    $(\mathrm{id}_i, W_i) \leftarrow \mathsf{DB}$
    for $w \in W$ do
        $(\mathrm{id}_1, \ldots, \mathrm{id}_{T_w}) \leftarrow \mathsf{DB}[w]$
        $\sigma \xleftarrow{\$} \mathrm{Perm}([T_w])$
        $\mathrm{WPerms}[w] \leftarrow \sigma$
        $\mathrm{stag} \leftarrow \{0,1\}^\kappa$
        $\mathrm{stags}[w] \leftarrow \mathrm{stag}$
        for $c \in [T_w]$ do
            $l \leftarrow F(\mathrm{stag}, c)$
            $e \leftarrow \mathrm{ABE.Enc}(mpk, \mathrm{id}_{\sigma[c]} || k_{\mathrm{id}_{\sigma[c]}}, \mathbb{A})$
            $\boxed{e \leftarrow \mathrm{ABE.Enc}(mpk, 0^\kappa, \mathbb{A})}$
            $\mathrm{xind} \leftarrow f_I(\mathrm{id}_{\sigma[c]});\ z \leftarrow f_Z(g_2^{1/w} \mod n||c)$
            $y \leftarrow \mathrm{xind} \cdot z^{-1}$
            $\mathsf{EDB}[l] \leftarrow (e, y)$
        end for
    end for
    $\mathsf{XSet} \leftarrow \mathrm{XSetSetup}(p, q, f_X, f_I, \mathsf{DB})$
    $\mathsf{sk}_{\mathbf{w}} \leftarrow \mathrm{ClientKGen}(p, q, \mathbf{w})$
    for $t \in [T]$ do
        $\mathsf{query\_stag} \leftarrow \mathrm{stags}[\mathbf{s}_t]$
        $\mathbf{t}[t] \leftarrow \mathrm{TransGen}(\mathsf{EDB}, \mathsf{XSet}, \mathsf{sk}_{\mathbf{w}}, f_X, f_Z, \mathbf{s}[t], \mathbf{x}[t, \cdot], \mathsf{query\_stag})$
    end for
    return $(\mathsf{EDB}, \mathsf{XSet}, \mathbf{t})$
end function

function XSetSetup(p, q, f_X, f_I, DB)
    $(\mathrm{id}_i, W_i) \leftarrow \mathsf{DB}$ ; $\mathsf{XSet} \leftarrow \emptyset$
    for $w \in W$ and $\mathrm{id} \in \mathsf{DB}[w]$ do
        $\mathrm{xind} \leftarrow f_I(\mathrm{id})$
        $\mathrm{xtag} \leftarrow g^{f_X(g_3^{1/w} \mod n) \cdot \mathrm{xind}}$
        $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathrm{xtag}\}$
    end for
    return XSet
end function

function ClientKGen(p, q, w)
    $\mathsf{sk}_{\mathbf{w}} \leftarrow \emptyset$
    for $i \in [3]$ do
        $\mathsf{sk}_{\mathbf{w}}^{(i)} \leftarrow g_i^{1/\prod_{j=1}^{n} w_j} \mod n$
    end for
    $\mathsf{sk}_{\mathbf{w}} \leftarrow (\mathsf{sk}_{\mathbf{w}}^{(1)}, \mathsf{sk}_{\mathbf{w}}^{(2)}, \mathsf{sk}_{\mathbf{w}}^{(3)})$
    return $\mathsf{sk}_{\mathbf{w}}$
end function

function TransGen(DB, EDB, XSet, sk_w, f_X, f_Z, s_t, x_t, query_stag)
    for $\alpha \in [|\mathbf{x}_t|]$ do
        for $c \in [T_c]$ do
            $\mathsf{xtoken}[\alpha, c] \leftarrow g^{f_Z\left((sk_{\mathbf{w}}^{(2)})^{\prod_{w \in \mathbf{w} \setminus \{\mathbf{s}_t\}} w} \mod n||c\right) \cdot f_X\left((sk_{\mathbf{w}}^{(3)})^{\prod_{w \in \mathbf{w} \setminus \mathbf{x}_t[\alpha]\}} w} \mod n\right)}$
        end for
    end for
    $\mathrm{Res} \leftarrow \mathrm{Search}(\mathsf{EDB}, \mathsf{XSet}, (\mathsf{query\_stag}, \mathsf{xtoken}))$
    $\mathrm{ResInds} \leftarrow \mathsf{DB}[\mathbf{s}_t, \mathbf{x}_t]$
    return $((\mathsf{query\_stag}, \mathsf{xtoken}), \mathrm{Res}, \mathrm{ResInds})$
end function
```

Algorithm 7 Game$_4$, $\boxed{\text{Game}_5}$ and $\boxed{\boxed{\text{Game}_6}}$

```
function INITIALIZE(DB, $\mathbf{w}, \mathbf{s}, \mathbf{x}$)
    $p, q, n, msk, mpk, \quad g_1, g_2, g_3 \xleftarrow{\$} \mathbb{Z}_n^*, \quad f_X, f_I, f_Z \xleftarrow{\$} \text{Fun}(\{0,1\}^\kappa, \mathbb{Z}_p^*)$
    EDB ← {}
    XSet ← $\emptyset$
    $(\text{id}_i, W_i) \leftarrow$ DB
    for $w \in W$ and each $\text{id}_i$ do
        $X[w] \leftarrow g^{f_X(g_3^{1/w} \mod n)}$ ; $\text{xind} \leftarrow f_I(\text{id}_i)$
        $H[\text{id}_i, w] \leftarrow X[w]^{\text{xind}}$
        $\boxed{\boxed{H[\text{id}_i, w] \xleftarrow{\$} \mathbb{G}}}$
    end for
    for $w \in W$ do
        $(\text{id}_1, \ldots, \text{id}_{T_w}) \leftarrow$ DB$[w]$
        $\sigma \xleftarrow{\$} \text{Perm}([T_w])$
        $\text{WPerms}[w] \leftarrow \sigma$
        $\text{stag} \leftarrow \{0,1\}^\kappa$
        $\text{stags}[w] \leftarrow \text{stag}$
        for $c \in [T_w]$ do
            $l \leftarrow F(\text{stag}, c)$
            $e \leftarrow \text{ABE.Enc}(mpk, 0^\kappa, \mathbb{A})$
            $\text{xind} \leftarrow f_I(\text{id}_{\sigma[c]})$; $z \leftarrow f_Z(g_2^{1/w} \mod n||c)$
            $y \leftarrow \text{xind} \cdot z^{-1}$
            $\boxed{y \xleftarrow{\$} \mathbb{Z}_p^*}$
            EDB$[l] \leftarrow (e, y)$
        end for
        for $u \in W \setminus w$ do
            for $c = T_w + 1, \ldots, T_c$ do
                $Y[w, u, c] \leftarrow X[u]^{f_Z(g_2^{1/w} \mod n||c)}$
                $\boxed{\boxed{Y[w, u, c] \xleftarrow{\$} \mathbb{G}}}$
            end for
        end for
    end for
    XSet ← XSETSETUP(DB, $H$)
    $\text{sk}_{\mathbf{w}} \leftarrow$ CLIENTKGEN($p, q, \mathbf{w}$)
    for $t \in [T]$ do
        query_stag ← stags$[\mathbf{s}_t]$
        $\mathbf{t}[t] \leftarrow$ TRANSGEN(DB, EDB, XSet, $H, Y, \mathbf{s}[t], \mathbf{x}[t, \cdot]$, query_stag)
    end for
    return (EDB, XSet, $\mathbf{t}$)
end function

function XSETSETUP(DB, $H$)
    $(\text{id}_i, W_i) \leftarrow$ DB ; XSet ← $\emptyset$
    for $w \in W$ and id $\in$ DB$[w]$ do
        XSet ← XSet $\cup \{H[\text{id}, w]\}$
    end for
    return XSet
end function

function TRANSGEN(DB, EDB, XSet, $H, Y, \mathbf{s}_t, \mathbf{x}_t$, query_stag)    // $\mathbf{x}_t[\alpha] = \mathbf{x}[t, \alpha]$ denotes the $\alpha$-th sterm of the $t$-th query.
    $(\text{id}_1, \ldots, \text{id}_{T_s}) \leftarrow$ DB$[\mathbf{s}_t]$; $\sigma \leftarrow \text{WPerms}[\mathbf{s}_t]$
    for $\alpha \in [|\mathbf{x}_t|]$ do
        for $c \in [T_s]$ do
            $l \leftarrow F(\text{query\_stag}, c)$
            $(e, y) \leftarrow$ EDB$[l]$
            xtoken$[\alpha, c] \leftarrow H[\text{id}_{\sigma[c]}, \mathbf{x}_t[\alpha]]^{1/y}$    // $y = f_I(\text{id}_{\sigma[c]}) \cdot (f_Z(g_2^{1/\mathbf{s}_t} \mod n||c))^{-1}$.
        end for
        for $c = T_s + 1, \ldots, T_c$ do
            xtoken$[\alpha, c] \leftarrow Y[\mathbf{s}_t, \mathbf{x}_t[\alpha], c]$
        end for
    end for
    Res ← SEARCH(EDB, XSet, (query_stag, xtoken))
    ResInds ← DB$[\mathbf{s}_t, \mathbf{x}_t]$
    return ((query_stag, xtoken), Res, ResInds)
end function
```

Algorithm 8 Game$_7$ and Game$_8$

```
function INITIALIZE(DB, $\mathbf{w}, \mathbf{s}, \mathbf{x}$) // Game$_7$, Game$_8$
    $msk, mpk$
    EDB ← {}
    XSet ← $\emptyset$
    $(\text{id}_i, W_i) \leftarrow$ DB
    for $w \in W$ and each $\text{id}_i$ do $H[\text{id}_i, w] \xleftarrow{\$} \mathbb{G}$ end for
    for $w \in \mathbf{s}$ do $\text{WPerms}[w] \xleftarrow{\$} \text{Perm}([T_s])$ end for
    for $w \in W$ do
        $\text{stag} \leftarrow \{0,1\}^\kappa$
        $\text{stags}[w] \leftarrow \text{stag}$
        for $c \in [T_w]$ do
            $l \leftarrow F(\text{stag}, c)$
            $e \leftarrow \text{ABE.Enc}(mpk, 0^\kappa, \mathbb{A})$
            $y \xleftarrow{\$} \mathbb{Z}_p^*$
            EDB$[l] \leftarrow (e, y)$
        end for
        for $u \in W \setminus w$ do
            for $c = T_w + 1, \ldots, T_c$ do
                $Y[w, u, c] \xleftarrow{\$} \mathbb{G}$
            end for
        end for
    end for
    XSet ← XSETSETUP(DB, $H$)
    for $t \in [T]$ do
        query_stag ← stags$[\mathbf{s}_t]$
        $\mathbf{t}[t] \leftarrow$ TRANSGEN(DB, EDB, XSet, $H, Y, \mathbf{s}[t], \mathbf{x}[t, \cdot]$, query_stag, $t$)
    end for
    return (EDB, XSet, $\mathbf{t}$)
end function

function XSETSETUP(DB, $H$) // Game$_7$, Game$_8$
    $(\text{id}_i, W_i) \leftarrow$ DB ; XSet ← $\emptyset$
    for $w \in W$ and id $\in$ DB$[w]$ do
        if $\exists t, \alpha$ s.t. id $\in$ DB$[\mathbf{s}[t]] \wedge \mathbf{x}[t, \alpha] = w$ then
            XSet ← XSet $\cup \{H[\text{id}, w]\}$
        else
            $h \xleftarrow{\$} \mathbb{G}$ ; XSet ← XSet $\cup \{h\}$
        end if
    end for
    return XSet
end function

function TRANSGEN(DB, EDB, XSet, $H, Y, \mathbf{s}_t, \mathbf{x}_t$, query_stag, $t$)    // Game$_8$ only
    $(\text{id}_1, \ldots, \text{id}_{T_s}) \leftarrow$ DB$[\mathbf{s}_t]$; $\sigma \leftarrow \text{WPerms}[\mathbf{s}_t]$
    for $\alpha \in [|\mathbf{x}_t|]$ do
        for $c = 1, \ldots, T_s$ do
            $l \leftarrow F(\text{query\_stag}, c)$
            $(e, y) \leftarrow$ EDB$[l]$
            if $\text{id}_{\sigma[c]} \in$ DB$[\mathbf{s}_t] \cap$ DB$[\mathbf{x}_t[\alpha]]$ then    // the equivalent condition used in XSETSETUP.
                xtoken$[\alpha, c] \leftarrow H[\text{id}_{\sigma[c]}, \mathbf{x}_t[\alpha]]^{1/y}$
            else if $\exists t' \neq t$ s.t. $\text{id}_{\sigma[c]} \in$ DB$[\mathbf{s}_{t'}] \wedge \mathbf{x}_t[\alpha] \in \mathbf{x}_{t'}$ then
                xtoken$[\alpha, c] \leftarrow H[\text{id}_{\sigma[c]}, \mathbf{x}_t[\alpha]]^{1/y}$
            else
                xtoken$[\alpha, c] \xleftarrow{\$} \mathbb{G}$
            end if
        end for
        for $c = T_s + 1, \ldots, T_c$ do
            xtoken$[\alpha, c] \xleftarrow{\$} \mathbb{G}$
        end for
    end for
    Res ← SEARCH(EDB, XSet, (query_stag, xtoken))
    ResInds ← DB$[\mathbf{s}_t, \mathbf{x}_t]$
    return ((query_stag, xtoken), Res, ResInds)
end function
```

Algorithm 9 Simulator code to calculate EDB

```
$mpk, msk$
for $i \in \bar{\mathbf{s}}$ do
    $\mathrm{stags}[i] \xleftarrow{\$} \{0,1\}^{\kappa}$
    $j \leftarrow 0$
    for $c \in [\mathrm{SP}[i]]$ do
        $l \leftarrow F(\mathrm{stags}[i], c)$
        $e \leftarrow \mathrm{ABE.Enc}(mpk, 0^{\kappa}, \mathbb{A})$
        $y \xleftarrow{\$} \mathbb{Z}_p^*$
        $\mathsf{EDB}[l] \leftarrow (e, y)$
        $j \leftarrow j + 1$
    end for
end for
for $i = j+1, \ldots, N$ do
    $l \xleftarrow{\$} \{0,1\}^{\kappa}$
    $e \leftarrow \mathrm{ABE.Enc}(mpk, 0^{\kappa}, \mathbb{A})$
    $y \xleftarrow{\$} \mathbb{Z}_p^*$
    $\mathsf{EDB}[l] = (e, y)$
end for
```

Algorithm 10 Simulator code to calculate XSet

```
for $w \in \hat{\mathbf{x}}$ and $\mathrm{id} \in \cup_{t\in[T],\alpha\in[A]} \mathrm{RP}[t,\alpha]$ do
    $H[\mathrm{id}, w] \xleftarrow{\$} \mathbb{G}$
end for
$\mathsf{XSet} \leftarrow \emptyset$
$j \leftarrow 0$
for $w \in \hat{\mathbf{x}}$ and $\mathrm{id} \in \cup_{\{(t,\alpha):\ \hat{\mathbf{x}}[t,\alpha]=w\}} \mathrm{RP}[t,\alpha]$ do
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{H[\mathrm{id}, w]\}$
    $j \leftarrow j + 1$
end for
for $i = j+1, \ldots, N$ do
    $h \xleftarrow{\$} \mathbb{G}$
    $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{h\}$
end for
```

Algorithm 11 Simulator code to calculate **t**

```
for $w \in \bar{\mathbf{s}}$ do
    $\mathrm{WPerms}[w] \xleftarrow{\$} \mathrm{Perm}([\mathrm{SP}[w]])$
end for
for $\tau \in [T]$ do
    $\mathsf{query\_stag} = \mathrm{stags}[\bar{\mathbf{s}}[\tau]]$
    for $w_x \in [\mathrm{XT}[\tau]]$ do
        $R \leftarrow \mathrm{RP}[\tau, w_x] \cup \bigcup_{t'\in[T],\beta\in[\mathrm{XT}[\tau]]} \mathrm{IP}[\tau, t', w_x, \beta]$
        $c \leftarrow 1$
        for $\mathrm{id} \in \mathrm{WPerms}[\bar{\mathbf{s}}[\tau]]$ do
            $(\mathrm{id}_1, \mathrm{id}_2, \cdots, \mathrm{id}_{Ts}) \leftarrow \mathrm{SRP}[\tau]; \sigma \leftarrow \mathrm{WPerms}[\bar{\mathbf{s}}[\tau]]$
            for $c \in [T_s]$ do
                if $\mathrm{id}_{\sigma[c]} \in R$ then
                    $l \leftarrow F(\mathsf{query\_stag}, c)$
                    $(e, y) \leftarrow \mathsf{EDB}[l]$
                    $\mathsf{xtoken}[\tau, w_x, c] \leftarrow H[\mathrm{id}_{\sigma[c]}, \hat{\mathbf{x}}[\tau, w_x]]^{1/y}$
                else
                    $\mathsf{xtoken}[\tau, w_x, c] \xleftarrow{\$} \mathbb{G}$
                end if
                $c \leftarrow c + 1$
            end for
        end for
        for $c = \mathrm{SP}[\bar{\mathbf{s}}[\tau]] + 1, \ldots, T_c$ do $\mathsf{xtoken}[\tau, w_x, c] \xleftarrow{\$} \mathbb{G}$ end for
    end for
    $\mathrm{Res} \leftarrow \textsc{Search}(\mathsf{EDB}, \mathsf{XSet}, (\mathsf{query\_stag}, \mathsf{xtoken}))$
    $\mathrm{ResInds} \leftarrow \textsc{Real\_Results}(\tau, \mathrm{RP}, \mathrm{SRP})$
    $\mathbf{t}[\tau] = ((\mathsf{query\_stag}, \mathsf{xtoken}), \mathrm{Res}, \mathrm{ResInds})$
end for
```