Privacy-Preserving and Regular Language Search over Encrypted Cloud Data

Kaitai Liang, Fuchun Guo, Xinyi Huang, and Joseph K. Liu

***Abstract*—Using cloud-based storage service, users can remotely store their data to clouds but also enjoy the high quality data retrieval services, without the tedious and cumbersome local data storage and maintenance. However, the sole storage service cannot satisfy all desirable requirements of users. Over the last decade, privacy-preserving search over encrypted cloud data has been a meaningful and practical research topic for outsourced data security. The fact of remote cloud storage service that users cannot have full physical possession of their data makes the privacy data search a formidable mission. A naive solution is to delegate a trusted party to access the stored data and fulfill a search task. This, nevertheless, does not scale well in practice as the fully data access may easily yield harm for user privacy. To securely introduce an effective solution, we should guarantee the privacy of search contents, i.e. what a user wants to search, and return results, i.e. what a server returns to the user. Furthermore, we also need to guarantee privacy for the outsourced data, and bring no additional local search burden to user. In this paper, we design a novel privacy-preserving functional encryption based search mechanism over encrypted cloud data. A major advantage of our new primitive compared to the existing public key based search systems is that it supports an extreme expressive search mode, regular language search. Our security and performance analysis show that the proposed system is provably secure and more efficient than some public key searchable systems with high expressiveness.**

Keywords: Regular language, secure data search, cloud storage.

I. Introduction

Much like the popularity of portable personal electronic devices, cloud storage service has been booming over the last decade. Its outstanding advantages, such as considerable storage space, flexible accessibility and convenient data retrieval, strongly catch the attention of Internet users. Accordingly, to date not only individuals but also industries, research institutes prefer to remotely store their data to cloud servers, such that they can get rid of the burden of local data management and maintenance. This makes cloud storage service share a great piece of market cut in the field of data management even in the ear of big data.

Remotely data storage delivers convenience to Internet users and meanwhile, brings security concerns. The fact that users cannot have full physical possession of their data immediately rises up two serious practical questions: how to guarantee the confidentiality of the data, and how to retrieve the data. For the first question, we usually tackle it by leveraging existing encryption cryptographic mechanisms, such that all outsourced data are encrypted and unaccessible to cloud servers. The encryption technology, with no doubt, enables us to protect the confidentiality of the data. However, it limits the flexibility of data retrieve to some extent. The premise of encryption technique is to prevent a ciphertext holder from gaining access to the underlying knowledge of data. Without any knowledge related to the data, it looks impossible for a cloud server to fulfill any data retrieval task. A naive solution here for data retrieval is to allow the server to fully access the data, allocate the data and next return it to user. Nevertheless, this disgraces the meaning of encryption.

To support data retrieval without loss of confidentiality, Searchable Encryption (SE) mechanisms (e.g. [27], [9]) have been proposed in the literature. SE has been studied and widely employed in real-world applications where data search is outsourced to untrusted cloud servers. SE allows a server to search in encrypted data on behalf of a data owner without accessing the information of the *data and search query contents*. In an SE, a user encrypts a file database and its search keywords, and next uploads them to a cloud server. When retrieving a file, the user delivers a token related to the keyword to the server so that the server then locates the corresponding encrypted file from the encrypted database. The flexibility and scalability of a SE system mainly depend on how we design search token as well as search keyword.

From practical point of view, a more expressive search query yields a more precise data retrieval. We take an Electronic Health Records (EHRs) search as an example. In an EHRs system, a patient's medical record is usually encrypted and stored in a storage system. We suppose there is a patient Alice's encrypted record which is tagged with a keyword index "Alice". To search the medical record of Alice from its storage system, a hospital needs to find a file matching the keyword "Alice". However, "Alice", the search index, is quite common in usual. There are probably 10,000 patients associated with the same keyword. This definitely increases the workload of the hospital to locate the real "Alice" file they need from the rest of other encrypted records (with the same keyword).

To enhance the search expressiveness, one may replace a single keyword index with access formula, such as ("Alice" AND "1990" AND "CrystalLake") or ("Alice" AND "Age $<$ 20" AND "Student $\in$ NYU"). Actually, the most powerful expressive way to represent a search query is to leverage regular language. Using regular language to describe a data to

K. Liang is with the Department of Computer Science, Aalto University, Finland (e-mail: kaitai.liang@aalto.fi).

F. Guo is with School of Computer Science and Software Engineering, University of Wollongong, NSW 2522, Australia (e-mail: fuchun@uow.edu.au).

X. Huang is with the Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China, 350108. (e-mail: xyhuang@fjnu.edu.cn).

J. K. Liu is with Monash University, Australia. (e-mail: joseph.liu@monash.edu).

be encrypted is extremely common in daily life. For instance, a secretary of department may archive all encrypted meeting files by tagging respective descriptions, like "Monday's meeting about Project A with B team". A Facebook user may directly write down a description, e.g., "my birthday party with best friends Bob and Kate", for an uploaded photo. Furthermore, suppose a tax form is encrypted and archived in some tax authority. The authority may need to search one of the tax forms based on an exact sentence or paragraph of the tax form, such as "Alice have paid $ 8,000 tax in total in 2014", in which the number is encrypted.

Some more recent applications for regular language search are online genetic relatedness test and chemical compound search. Suppose a language space only contains "A,G,C,T", a search querier may upload a masked search pattern "ACG-GTTCT" to an encrypted genetic database to request the server to return all possible matching encrypted DNA sequences. Unfortunately, there is no SE supporting regular language search in the literature. Designing flexible and scalable regular language search without loss of data confidentiality and query privacy that becomes the main motivation of our work.

Searchable Symmetric Encryption (SSE) and Public key Encryption with Keyword Search (PEKS) are two types of SE. SSE generally enjoys better search efficiency than that of PEKS. It provides a limited level of expressiveness for search. It is not difficult to see that the limitation of expressiveness actually inherits from some original limitation design in symmetric encryption[1], such that it is difficult for SSE to support expressive search query (e.g. formula search, subset queries). Therefore, we deal with the case of PEKS to achieve more search expressiveness in this paper.

PEKS allows anyone to encrypt searchable contents but only a defined user group can generate search trapdoor. Moreover, it separates search contents from search keywords that offers flexibility for search expressiveness. The notion of PEKS is initially defined by Boneh et al. [5] in EUROCRYPT 2004. Afterwards, there are many PEKS systems have been proposed. We mainly concentrate on the following two systems below. In TCC 2007, Boneh and Waters [6] proposed a PEKS system supporting conjunctive, subset, and range search queries. It can be regarded as the most expressive PEKS scheme in the literature. However, it does not guarantee the privacy of search contents, i.e. a cloud server will know what a user wants to search. We will further explain the design limitation of [6] later. A latest research work for PEKS is proposed in IEEE INFOCOM 2014 by Zheng, Xu and Ateniese [31]. In [31], the authors explore PEKS into the context of Attribute-Based Encryption (ABE). Although their systems (note that they designed more than one system in [31]) achieve data and search contents privacy, they only support single keyword equality search that cannot fully achieve our goal, regular language search.

[1]It is undeniable that symmetric key encryption focuses more on protecting the confidentiality of the data but not the expressiveness of data share.

A. Our Contributions

We first define a new notion called searchable deterministic finite automata-based functional encryption. The notion is a general notion for PEKS. We next design a concrete construction satisfying the notion. In our construction, any system user can describe a data to be shared with regular language in an encrypted form, where the language description can be arbitrary length (e.g. an English sentence, or a paragraph). A valid data receiver can generate and deliver a search token represented as a Deterministic Finite Automata (DFA) to a cloud server, such that the cloud server can locate the corresponding ciphertexts and return them to the data receiver. In the search phase, the server knows nothing about the search contents and the underlying data. We further present extensive evaluation for our system to show its security, and the efficiency compared to two most related works [6], [31].

Supporting regular language search is a great advantage of our system that makes the system be the first of its type, to the best of our knowledge. It is undeniable that SSE (e.g. [12]) usually enjoys better efficiency in data searching compared to the public key based searchable encryption. However, our novel system can support any arbitrary alphabet/regular language search, so that it is more human-friendly readable for search keyword design. Besides, the system provides verifiable (data integrity) check for system users (due to public-key based feature). Moreover, our system does not need to require a data owner to pick up some special keywords before constructing keyword index structures, e.g., least frequent keyword [12], but also it only leverages a DFA structure to embed flexible search expressiveness, e.g., "AND, OR, NOT", unlike that of [12] only limited in "a keyword AND (formula)" expression.

B. Related Work

Song et al. [27] introduced the notion of SE, in which full text search over encrypted data is allowable. Following the notion, many SE systems have been proposed in the literature. All existing systems can be categorized into two types: SSE (e.g. [13], [14], [28], [7]) and PKES (e.g. [5]). Although SSE generally enjoys better search efficiency than that of PEKS, it provides limited expressiveness for search, e.g., single keyword match [16], [21], [22], conjunctive keyword search [12], [18], [11], and fuzzy search [8], [25], [10]. To achieve more expressiveness for search query, this paper deals with the case of PKES. A more important reason for us to start with public key technology is that the loose bind between keyword index structure and (uploaded) data (in a SSE system) does not support integrity check. This may give malicious server a chance to completely change the whole search index structure, such that the corresponding data owner fails to correctly search its data.

Boneh et al. [5] introduced the notion of PKES, and designed a concrete scheme an identity-based encryption (IBE). Later on, Abdalla et al. [1] presented a generic construction from anonymous IBE to SE. After that, some variants of PKES supporting single keyword search have been proposed, such as [2], [3]. More practical and search expressive PKES have been constructed, e.g., authorized keyword

search [19], verifiable keyword search [4], fuzzy (single) keyword search [30], [19], [18] with conjunctive keyword search, and [26] with range queries.

In TCC 2007, Boneh and Waters [6] designed the most expressive PKES for not only conjunctive but also keyword subset/range queries by leveraging hidden vector encryption technique. But it is built in composite order group that seriously affects its efficiency. We note that [20] introduces a way to convert the system into prime order group. Most importantly, the system cannot protect the privacy of search contents. Recently, Zheng, Xu and Ateniese [31] introduced a notion attribute-based keyword search, in which they explore the keyword search into the ABE setting without loss of search content privacy. Nevertheless, their research outcomes only support single keyword search.

Below we compare our work with [6] and [31] in terms of functionality and security in Table I. As to the efficiency comparison, we leave it to Section V. To the best of our knowledge, our scheme is the first to achieve privacy-preserving regular language keyword search. We note that our system is proved under the asymmetric l-Expanded BDHE (which will be introduced later), while [6], [31] rely on composite 3-party Diffie-Hellman assumption and the bilinear Diffie-Hellman assumption, and decisional linear assumption, respectively.

TABLE I: Comparison with [6], [31]

Sch.	Search	Security	Standard Model	Pairings Group
[6]	Subset, range conjunctive	CPA	✓	composite order
[31]	Single keyword	CPA	✗	symmetric
Ours	Regular Language	CPA	✓	asymmetric

II. Problem Statement

A. System Entities

We consider a cloud-based data store and search service involving four different entities, as illustrated in Fig. 1.

- A data encryptor can mark a specified keyword description for a data, and upload the encrypted data to a cloud server. It can be a data receiver as well (indicating a user encrypts data for itself).
- A data receiver can download encrypted data from cloud server, have fully decryption rights of the encrypted data, but also to construct a search token with help of a fully trusted authority.
- A fully trusted authority takes charge of generating the public parameters for the system, initializing the system, issuing a partial search token to a data receiver to help the data receiver construct a keyword search token.
- A cloud server: given a search token associated with a search policy[2] and a ciphertext tagged with unknown keyword description, it verifies whether the ciphertext and the token match or not. If there is a match, output 1 and return the corresponding ciphertext; otherwise, output 0 and return $\perp$.

[2]We note that we use a DFA to represent a search policy in this paper. We will introduce the definition of DFA in Appendix A.

B. System Algorithms

Definition 1: A Searchable DFA-Based Functional Encryption (S-DFA-FE) system consists of the following algorithms:

1) $(mpk, msk) \leftarrow Setup(1^n, \sum)$: on input a security parameter 1^n and the description of a finite alphabet $\sum$, output a master public key mpk and a master secret key msk.
2) $CT \leftarrow Enc(mpk, W = (w_1, w_2, ..., w_l))$: on input mpk and an l-length string W used for keyword description, output a ciphertext CT.
3) $tk \leftarrow TokenGen(msk, M = (Q, \mathcal{T}, q_0, F))$: on input a msk and a DFA description $M = (Q, \mathcal{T}, q_0, F)$, a fully trusted authority interacts with a data receiver to generate a search token tk. Specifically, the authority delivers a partial search token ptk to the receiver, and next the receiver constructs a fully search token tk. In this paper we require that the communication channel between the authority and the receiver must be secure, for example, using SSL.
4) $1/0 \leftarrow Test(tk, CT)$: on input a search token tk, and a ciphertext CT, output 1 if the DFA of the token accepts the string W embedded into the ciphertext, and 0 otherwise.

Note. We note that the system does not provide a decryption algorithm that uses a user's secret key to decrypt a ciphertext CT and outputs a message. This capability can be achieved by leveraging a standard public key system. From the point of view of encrypted search functionality, there is no need to explicitly support this capability.

C. System Architecture

We depict how our system works in Fig. 1.

- A fully trusted authority runs the setup algorithm $Setup$ to generate the public parameter mpk for system users and a cloud server, initialize the system and keep the master secret key msk secret.
- A data encryptor runs the encryption algorithm Enc to generate different ciphertexts (CT_{W_1}, ..., CT_{W_n}) for the data receiver with "masked-and-unknown" keyword description strings (W_1, ..., W_n), and further uploads the ciphertexts to the cloud.
- When needing the help of the trusted authority to construct a search token for some specified keyword description, the data receiver first builds up a secure channel (e.g. SSL) with the authority, and next sends a request with the corresponding DFA search policy to the authority. The authority then generates a "partial" search token and replies the token to the data receiver via the same secure channel. We note that we will improve the system later such that there is no need to require the fully trusted authority to participate into the generation of search token.
- After receiving the partial token from the authority, the data receiver generates a fully search token tk_{W_i},

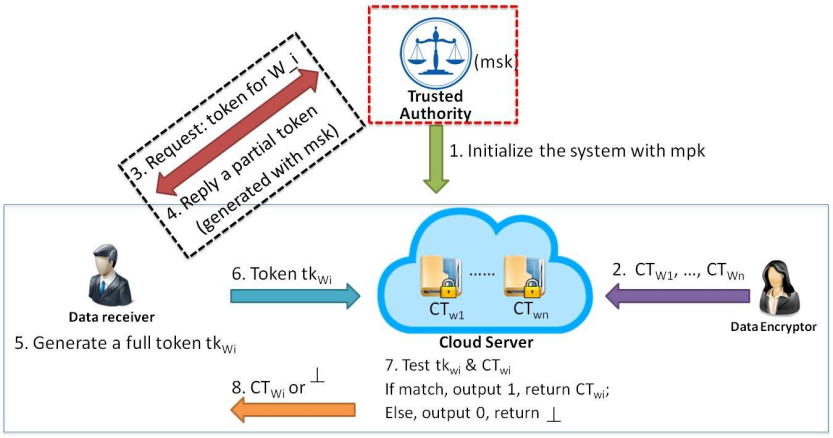

Fig. 1: System Architecture

and forwards the token to the cloud server in a public communication channel.

- The cloud server verifies whether there exists a CT_{W_i} matching the search token tk_{W_i}. If there is a match, the server outputs 1 and returns the corresponding ciphertext CT_{W_i}. Otherwise, the server outputs 0 and returns $\perp$.

Note. One might question that why we need a trusted authority in the generation of search token. In a practical point of view, a valid system user holding a secret key is capable of generating a search token for his/her encrypted files. We state that our system can be improved to achieve this requirement. We will discuss this after the description of our construction.

D. Threat Model

The below defined adversary model is to see whether a PPT adversary can tell a ciphertext is associated with a keyword description string or not, in which all the characters of the keyword string is from a public known finite alphabet. The defined model is used to prevent the attacks that given a ciphertext (resp. a search token), any invalid data receiver (i.e. the one without any decryption rights) cannot compromise the corresponding keyword description field.

Assumption. We assume that the cloud server and data receiver are semi-honest (i.e. honest-but-curious), while the authority is fully trusted. By semi-honest we mean that one will honestly run a protocol by following the specification of the protocol but curiously collecting some interesting information during the period of protocol running.

Definition 2: An S-DFA-FE system achieves keyword privacy if the advantage $Adv_{\mathcal{A}}^{KP}(1^n, \sum)$ is negligible for any PPT adversary $\mathcal{A}$ in the following experiment.

$$
\begin{aligned}
|Pr[b = b' : &(W_0^*, W_1^*, state_1) \leftarrow \mathcal{A}(\sum);\\
&(mpk, msk) \leftarrow Setup(1^n, \sum);\\
&(state_2) \leftarrow \mathcal{A}^{\mathcal{O}}(mpk, state_1);\\
&b \in_R \{0,1\}; CT^* \leftarrow Enc(mpk, W_b^*);\\
&b' \leftarrow \mathcal{A}^{\mathcal{O}}(CT^*, state_2)] - \frac{1}{2}|,
\end{aligned}
$$

where $state_1$, $state_2$ are the state information, W_0^*, W_1^* are two distinct keyword strings, and $\mathcal{O} = \{\mathcal{O}_{tg}, \mathcal{O}_{test}\}$. For $\mathcal{O}_{test}$, the oracle intakes (M, CT), and outputs a bit of value 0 or 1. But the oracle will only return $\perp$ for the query (M, CT), where CT is the challenge ciphertext and meanwhile M accepts CT's keyword string. For $\mathcal{O}_{tg}$, the oracle intakes M, and outputs a search token tk associated with M with an exception that M cannot accept W_b^*, where $b \in \{0,1\}$.

E. Design Goals

In this paper, our protocol design achieves the following functionality and security guarantees.

- Publicly description ability: to allow every system user to produce a ciphertext associated with some keyword string for others.
- Search token generation ability: to allow any valid system user obtaining the decryption rights of a data to generate a search token for the corresponding encrypted data search.
- Test verifiability: to allow a cloud server to allocate one (or more) matching ciphertext(s) by a given search token.
- Privacy-preserving ability: to guarantee the following aspects of privacy.
  1) Given a search token, a cloud server does not know any knowledge of the keyword(s) embedded into the DFA of the token[3].
  2) Given a ciphertext, a cloud server does not know any information of the keyword string tagged with the ciphertext.

III. Preliminaries

A. Asymmetric Pairings

Let $BSetup$ be an algorithm that on input the security parameter n, outputs the parameters of a bilinear map as

[3]Note that we here only give the token to the server for guessing the embedded keyword(s). Of course, the server can construct a ciphertext with a keyword K to test whether a search token with another keyword K' matches the ciphertext or not. This is called offline keyword-guessing attack. We will discuss this attack later.

$(p, g, \hat{g}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are multiplicative cyclic groups of prime order p, where $|p| = n$, and g is a random generator of $\mathbb{G}_1$, $\hat{g}$ is a random generator of $\mathbb{G}_2$. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ has three properties: (1) *Bilinearity*: for all $a, b \in_R \mathbb{Z}_p^*$, $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$; (2) *Non-degeneracy*: $e(g, \hat{g}) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ is the unit of $\mathbb{G}_T$; (3) *Computability*: e can be efficiently computed. Note that $\mathbb{G}_1$ and $\mathbb{G}_2$ are not the same.

B. Complexity Assumptions

Definition 3: **l-Expanded Bilinear Diffie-Hellman Exponent (l-Expanded BDHE) Assumption [29]**. We say that an algorithm $\mathcal{A}$ has advantage $Adv_{\mathcal{A}}^{l\text{-}BDHE}$ in solving the l-Expanded BDHE problem in $\mathbb{G}$ if $|Pr[\mathcal{A}(\hat{X}, e(g,g)^{a^{l+1}bs}) = 0] - Pr[\mathcal{A}(\hat{X}, T) = 0]| \geq \epsilon$, where the probability is over the random choice of generators $g \in \mathbb{G}_1$, the random choice of exponents $a, b, c_0, ..., c_{l+1}, d \in \mathbb{Z}_p^*$, $T \in_R \mathbb{G}_T$, the random bits used by $\mathcal{A}$, and $\hat{X}$ is a set of the following elements:

$$g, g^a, g^b, g^{ab/d}, g^{b/d}$$
$$\forall i \in [0, 2l+1], i \neq l+1, j \in [0, l+1]\ g^{a^i s}, g^{a^i bs/c_j},$$
$$\forall i \in [0, l+1]\ g^{a^i b/c_i}, g^{c_i}, g^{a^i d}, g^{abc_i/d}, g^{bc_i/d},$$
$$\forall i \in [0, 2l+1], j \in [0, l+1]\ g^{a^i bd/c_j},$$
$$\forall i, j \in [0, l+1], i \neq j\ g^{a^i bc_j/c_i}.$$

We say that the l-Expanded BDHE assumption holds in $\mathbb{G}_1$ if no PPT algorithm has advantage ϵ in solving the l-Expanded BDHE problem in $\mathbb{G}_1$.

Definition 4: **(Asymmetric) l-Expanded BDHE Assumption**. We say that an algorithm $\mathcal{A}$ has advantage $Adv_{\mathcal{A}}^{A\text{-}l\text{-}BDHE}$ in solving the asymmetric l-Expanded BDHE problem in ($\mathbb{G}_1$, $\mathbb{G}_2$) if $|Pr[\mathcal{A}(\hat{X}, e(g,\hat{g})^{a^{l+1}bs}) = 0] - Pr[\mathcal{A}(\hat{X}, T) = 0]| \geq \epsilon$, where the probability is over the random choice of generators $g \in \mathbb{G}_1$, $\hat{g} \in \mathbb{G}_2$, the random choice of exponents $a, b, c_0, ..., c_{l+1}, d \in \mathbb{Z}_p^*$, $T \in_R \mathbb{G}_T$, the random bits used by $\mathcal{A}$, and $\hat{X}$ is a set of the following elements:

$$g, \hat{g}, \hat{g}^a, g^a, \hat{g}^b, \hat{g}^{ab/d}, g^{ab/d}, g^{b/d}, \hat{g}^{b/d}$$
$$\forall i \in [0, 2l+1], i \neq l+1, j \in [0, l+1]\ g^{a^i s}, g^{a^i bs/c_j},$$
$$\forall i \in [0, l+1]\ g^{a^i b/c_i}, \hat{g}^{a^i b/c_i}, \hat{g}^{c_i}, \hat{g}^{a^i d}, \hat{g}^{abc_i/d}, \hat{g}^{bc_i/d},$$
$$\forall i \in [0, 2l+1], j \in [0, l+1]\ \hat{g}^{a^i bd/c_j},$$
$$\forall i, j \in [0, l+1], i \neq j\ \hat{g}^{a^i bc_j/c_i}.$$

We say that the asymmetric l-Expanded BDHE assumption holds in ($\mathbb{G}_1$, $\mathbb{G}_2$) if no PPT algorithm has advantage ϵ in solving the asymmetric l-Expanded BDHE problem in ($\mathbb{G}_1$, $\mathbb{G}_2$).

We can show that the above extended complexity assumption still holds in the generic group model by employing the same proof technology introduced in [29]. Specifically, we can see from the set $\hat{X}$ that there are five elements in $\mathbb{G}_1$ including g, $g^{b/d}$, g^a, $g^{ab/d}$, $g^{a^i s}$, $g^{a^i bs/c_j}$ and $g^{a^i b/c_i}$. Here, we show that these elements cannot help an adversary to compute an exponent value $a^{l+1}bs$. For the element g, it is easy to see that there does not exist a $\hat{g}^{a^{l+1}bs}$ in $\hat{X}$. Similarly, we need the $\mathbb{G}_2$ elements $\hat{g}^{a^{l+1}ds}$, $\hat{g}^{a^l ds}$, $\hat{g}^{a^l bs}$, $\hat{g}^{a^z b}$, $\hat{g}^{a^z c_j}$ and $\hat{g}^{a^z sc_i}$ for the $\mathbb{G}_1$ elements $g^{b/d}$, g^a, $g^{ab/d}$, $g^{a^i s}$, $g^{a^i bs/c_j}$ and $g^{a^i b/c_i}$, respectively, where $i + z = l + 1$. It is not difficult to see that the above $\mathbb{G}_2$ elements cannot be provided by the set $\hat{X}$.

C. Target Collision Resistant Hash Function.

Target Collision Resistant (TCR) hash function was introduced in [15]. A TCR hash function H guarantees that given a random element x which is from the valid domain of H, a PPT adversary $\mathcal{A}$ cannot find $y \neq x$ such that $H(x) = H(y)$. We let $Adv_{H,\mathcal{A}}^{TCR} = Pr[H(x) = H(y) \wedge x \neq y | x, y \in DH]$ be the advantage of $\mathcal{A}$ in successfully finding collisions from a TCR hash function H, where DH is the valid input domain of H. If a hash function is chosen from a TCR hash function family, $Adv_{H,\mathcal{A}}^{TCR}$ is negligible.

IV. An Expressive Keyword Search Mechanism

From a lemma stated in [5], we know that a keyword search mechanism can be built on top of an anonymous identity-based encryption system. Based on this valuable observation, we construct an expressive keyword search mechanism from an anonymous attribute-based encryption scheme. The anonymous attribute-based encryption can be regarded as a combination of masked technique and an extension of Waters functional encryption [29].

A. An Intuition for Our Basic Construction

We first take an expressive functional encryption system [29] as a starting point. We treat [29] in a different point of view, specifically in a searchable encryption view. We regard the description string associated with a ciphertext as a description for future search, and the secret key of a user as a search token. If the token matches the ciphertext (namely a successful decryption occurs in the functional encryption point of view), a cloud server will return the ciphertext.

Nevertheless, we cannot simply turn [29] to be a searchable encryption system. In [29], one (i.e. a data encryptor) needs to regard a description string (for a ciphertext) as one of the output of ciphertext, i.e. showing the string in a plain format. In this case, the privacy of the string, however, cannot be protected. We thus extend the system into the asymmetric pairing group such that we can guarantee the privacy of the search description string associated with a given ciphertext. That is, given a ciphertext tagged with some search description, a cloud server does not know what the corresponding description is. Furthermore, we also need to require the data encryptor to publicly show "the length of the string". Accordingly, we leak the length of the string only but not the plain string itself. One might doubt that the reason of this kind of information leak. We will explain it later.

To achieve the privacy of a search query, we require a trapdoor generator (i.e. a data receiver) to "hide" each character of a description string as well as each state information when constructing a DFA. The search query now is represented as the DFA, where DFA is much like a graph with direction includes a unique initial node (resp. state), many edges and ending nodes (resp. states). For each edge (with direction)

connecting two nodes (resp. states), there is a transaction, (x, y, σ), where x is the origin node, y is the destination one, and σ is the character of a string, say W. We note that all the characters are from a finite alphabet $\sum$. To hide the information of the character and states, σ as well as x and y should be masked by some way such that each transaction only shows the direction. For the initial state and accept states, the cloud server will not know the exact value of them, but be notified that which ones are the initial state and accept states, respectively. We will introduce a specific way to hide the information of characters and states later.

We assume that a DFA (graph) has many different direction paths, and each of them has the same initial state, i.e. the root of the graph, and a unique accept state, i.e. the end note of the path. We define the term "full path" below. If there exists a path from an initial state to an accept state intaking a string W as input, we state that this path is a full path with $|W|$ edges, where $|W|$ is the length of the string W.

We state that the system construction technique of [29] does deliver us the possibility of the above information hidden behavior. First of all [29] embeds each character of an alphabet into individual public elements, and each character for encryption is represented as a single ciphertext component as well. Secondly, a successful decryption relies on a fact that whether there exists a full path in a DFA (associated with a secret key) matches a string (associated with a ciphertext). After our anonymous conversion for the ciphertext, no one will know a given single ciphertext component corresponds to what character. Moreover, a masked DFA only delivers a graph with unknown states and characters to a cloud server. The only requirement for the cloud server (upon doing search matching) is to find out the full path(s) in a DFA. For instance, a masked DFA includes 2 full paths, one with length 3 and the other with length 4. The cloud server will need to first find out the paths as well as their respective length, and choose all ciphertexts from its storage encrypted backend associated with unknown description strings with length 3 and 4. The cloud server then makes some specific computation by using the DFA search token and the ciphertext so as to verify whether they match or not. The construction technology of [29] guarantees that only a length match pair of ciphertext and search token can make a possible search match. Thus, the premise of our search technique is that we need to let cloud server know the graph's direction and the corresponding full path(s) in a DFA search token and the length of description string in a ciphertext.

B. Scheme Notations

We summarize the notations used in our system in Table II.

TABLE II: Frequently Used Notations

1^n	security parameter
$\sum$	a finite alphabet
$H(\psi)$	a TCR hash function with input ψ
$a \in_R A$	a is randomly chosen from the field A
$\mathbb{Z}_p^*$	a (non-zero) positive field of all integers module a prime p
$[1, l]$	all integers from 1 to l
$M = (Q, \mathcal{T}, q_0, F)$	a DFA description including: Q is a set of all states, a state is $q_{i\in[1,\|Q\|]}$ $\mathcal{T}$ is a transaction function: $Q \times \sum \to Q$ a transaction denotes as $t_i = (q_{i-1}, q_i, w_i) \in \mathcal{T}$ $q_0 \in Q$ is the start state $F \in Q$ is a set of accept states
$\forall i \in I$	for all indexes i belonging to the set I
mpk	the master public key
msk	the master secret key
$W = (w_1, ..., w_l)$	a keyword string with length l, i.e. $\|W\| = l$
CT	a ciphertext associated with a keyword string
tk	a keyword search token

C. Details of Our Basic Construction

- **Setup**$(1^n, \sum)$**:** The setup algorithm is run by a fully trusted authority who will initialize the system, publish the system public parameters, and store the master secret key secretly.
  1) Choose $\beta_f, \beta_{end}, \beta_{start}, \alpha, \xi \in_R \mathbb{Z}_p^*$ and for each σ in $\sum$ choose $\beta_\sigma \in_R \mathbb{Z}_p^*$. Choose a TCR hash function: $H : \mathbb{G}_T \to \mathbb{G}_1$.
  2) Choose $g \in \mathbb{G}_1$ and $\hat{g} \in \mathbb{G}_2$. Set $h_{start} = g^{\beta_{start}}$, $\hat{h}_{start} = \hat{g}^{\beta_{start}}$, $h_{end} = g^{\beta_{end}}$, $\hat{h}_{end} = \hat{g}^{\beta_{end}}$, $f = g^{\beta_f}$, $\hat{f} = \hat{g}^{\beta_f}$, $z = g^{\xi}$, $\hat{z} = \hat{g}^{\xi}$, and set a pair of elements corresponding to a σ in $\sum$ as $h_\sigma = g^{\beta_\sigma}$ and $\hat{h}_\sigma = \hat{g}^{\beta_\sigma}$.
  3) Set the master secret key as $msk = (\hat{g}^{-\alpha}, \hat{z})$, and the master public key as $mpk = (e(g, \hat{g})^{\alpha}, g, \hat{g}, z, f, \hat{f}, h_{start}, h_{end}, \hat{h}_{start}, \hat{h}_{end}, \forall\sigma \in \sum h_\sigma, \hat{h}_\sigma)$.
- **Enc**$(mpk, W = (w_1, ..., w_l))$**:** the encryption algorithm is run by a data encryptor who would like to send a data with an arbitrary length string W.
  1) Choose random elements $s_0, ..., s_l \in_R \mathbb{Z}_p^*$.
  2) Set the ciphertext CT as:
     $C_1 = H(e(g, \hat{g})^{\alpha s_l})$, $C_{start1} = C_{0,1} = g^{s_0}$, $C_{start2} = (h_{start})^{s_0}$,
     for $i = 1$ to l:
     $C_{i,1} = g^{s_i}$, $C_{i,2} = h_{w_i}^{s_i} z^{s_{i-1}}$, $C_{i,3} = f^{s_i}$,
     finally $C_{end1} = C_{l,1} = g^{s_l}$, $C_{end2} = (h_{end})^{s_l}$.
  3) Output $CT = (l, C_1, C_{start1}, C_{start2}, \{C_{i,1}, C_{i,2}, C_{i,3}\}_{i=[1,l]}, C_{end1}, C_{end2})$.
- **TokenGen**$(msk, M = (Q, \mathcal{T}, q_0, F))$:
  1) the partial token generation algorithm is run by the fully trusted authority.
     The description of M includes a set Q of states $q_0, ..., q_{|Q|-1}$ and a set of transitions $\mathcal{T}$ where each transition $t \in \mathcal{T}$ is a triple $(x, y, \sigma) \in Q \times Q \times \sum$. q_0 is designated as a unique start state and $F \subseteq Q$ is the set of accept states. The algorithm chooses $\hat{D}_0, \hat{D}_1, ..., \hat{D}_{|Q|-1} \in_R \mathbb{G}_2$ (associating $\hat{D}_i$ with q_i), for each $t \in \mathcal{T}$ it chooses $r_t \in_R \mathbb{Z}_p^*$, $\forall q_x \in F$ it chooses $r_{end_x} \in_R \mathbb{Z}_p^*$. The trusted authority constructs the partial search token as follows. First it sets:

$$K_{start1} = \hat{D}_0 \cdot (\hat{h}_{start})^{r_{start}}, K_{start2} = \hat{g}^{r_{start}},$$

     For each transaction $t = (x, y, \sigma) \in \mathcal{T}$, the algorithm sets:

$$\begin{aligned} K_{t,1} &= \hat{D}_x^{-1} \cdot \hat{z}^{r_t}, K_{t,2} = \hat{g}^{r_t}, \\ K_{t,3} &= \hat{D}_y \cdot (\hat{h}_\sigma)^{r_t}, \end{aligned}$$

For each $q_x \in F$ it computes:

$$K_{end_{x,1}} = \hat{g}^{-\alpha} \cdot \hat{D}_x \cdot (\hat{h}_{end})^{r_{end_x}}, K_{end_{x,2}} = \hat{g}^{r_{end_x}}.$$

Finally, set the partial search token as

$$\Big(K_{start1}, K_{start2},$$

$$\forall t \in \mathcal{T}(K_{t,1}, K_{t,2}, K_{t,3}), \forall q_x \in F(K_{end_{x,1}}, K_{end_{x,2}})\Big).$$

We note that the partial search token is delivered to the corresponding receiver via a secure communication channel (such as SSL).

2) The data receiver then constructs the full search token as follows. It chooses new random elements $\bar{D}_0, \bar{D}_1, ..., \bar{D}_{|Q|-1} \in_R \mathbb{G}_2$, $\theta_1, \theta_2 \in_R \mathbb{Z}_p^*$, $\forall q_x \in F$ it chooses $\theta_{3,x} \in_R \mathbb{Z}_p^*$. First it sets:

$$\begin{aligned} K_{start1} &= \hat{D}_0 \cdot (\hat{h}_{start})^{r_{start}} \cdot \bar{D} \cdot (\hat{h}_{start})^{\theta_1}, \\ K_{start2} &= \hat{g}^{r_{start}} \cdot \hat{g}^{\theta_1}, K_{start3} = \hat{g}^{\theta_2}, \end{aligned}$$

For each transaction $t = (x, y, \sigma) \in \mathcal{T}$, the algorithm sets:

$$\begin{aligned} K_{t,1} &= \hat{D}_x^{-1} \cdot \hat{z}^{r_t} \cdot \bar{D}_x^{-1}, K_{t,2} = \hat{g}^{r_t}, \\ K_{t,3} &= \hat{D}_y \cdot (\hat{h}_\sigma)^{r_t} \cdot \bar{D}_y \cdot \hat{f}^{\theta_2}, \end{aligned}$$

For each $q_x \in F$ it computes:

$$\begin{aligned} K_{end_{x,1}} &= \hat{g}^{-\alpha} \cdot \hat{D}_x \cdot (\hat{h}_{end})^{r_{end_x}} \cdot \bar{D}_x \cdot (\hat{h}_{end})^{\theta_{3,x}}, \\ K_{end_{x,2}} &= \hat{g}^{r_{end_x}} \cdot \hat{g}^{\theta_{3,x}}. \end{aligned}$$

Finally, set the fully search token as

$$tk = \Big(M^*, K_{start1}, K_{start2}, K_{start3},$$

$$\forall t \in \mathcal{T}(K_{t,1}, K_{t,2}, K_{t,3}), \forall q_x \in F(K_{end_{x,1}}, K_{end_{x,2}})\Big),$$

where M^* is identical to M except that each state and string of the DFA turns to be a masked symbol $*$.

- **Test**(tk, CT)**:** a cloud server proceeds the following test to verify whether a given ciphertext CT matches a search token tk or not. If yes, output 1 and output 0 otherwise. In the view of the cloud server, CT is associated with an unknown string $* = (*_1, ..., *_l)$ and the search token tk is associated with a search pattern represented by a DFA $M^* = (Q, \mathcal{T}, q_0, F)$. The cloud server first checks whether there exist a sequence of $l+1$ states $*_0, *_1, ..., *_l$ and l transitions $t_1, ..., t_l$ fitting an unknown string with length l (note by fitting we mean they only match in length), where $*_0$ is a starting state and $*_l \in F$, and for $i = 1, ..., l$, we have $t_i = (*_{i-1}, *_i, *_i) \in \mathcal{T}$. If there does not exist the corresponding sequence, the cloud server outputs 0 indicating a mismatch. Otherwise, the server proceeds.
  First compute:

$$\begin{aligned} B_0 &= e(C_{start1}, K_{start1}) \cdot e(C_{start2}, K_{start2})^{-1} \\ &= e(g, \hat{D}_0 \cdot \bar{D}_0)^{s_0}. \end{aligned}$$

For $i = 1$ to l, compute:

$$\begin{aligned} B_i &= B_{i-1} \cdot e(C_{(i-1),1}, K_{t_i,1}) \cdot e(C_{i,2}, K_{t_i,2})^{-1} \\ &\quad \cdot e(C_{i,1}, K_{t_i,3}) \\ &= e(g, \hat{D}_{u_i} \cdot \bar{D}_{u_i})^{s_i} \cdot e(g^{\sum_{v=1}^{i} s_v}, \hat{f}^{\theta_2}). \end{aligned}$$

Since M accepts w, we have that $u_l = q_x$ for some $q_x \in F$ and $B_l = e(g, \hat{D}_x \cdot \bar{D}_x)^{s_l} \cdot e(g^{\sum_{v=1}^{l} s_v}, \hat{f}^{\theta_2})$. Further compute

$$\begin{aligned} B_{end} &= B_l \cdot e(C_{end_{x,1}}, K_{end_{x,1}})^{-1} \\ &\quad \cdot e(C_{end_{x,2}}, K_{end_{x,2}}) \cdot e(C_{end_{x,3}}, K_{start3}) \\ &= e(g, \hat{g})^{\alpha \cdot s_l} \cdot e(g^{\sum_{v=1}^{l} s_v}, \hat{f}^{\theta_2}). \end{aligned}$$

If

$$\begin{aligned} C_1 &= H(B_{end}/e(\prod_{i=1}^{l} C_{i,3}, K_{start3})) \\ &= H(\frac{e(g, \hat{g})^{\alpha \cdot s_l} e(g^{\sum_{v=1}^{l} s_v}, \hat{f}^{\theta_2})}{e(\prod_{i=1}^{l} f^{s_i}, \hat{g}^{\theta_2})}) \\ &= H(e(g, \hat{g})^{\alpha \cdot s_l}), \end{aligned}$$

output 1 (indicating a match); otherwise, output 0.

D. Improvement for Our Basic Construction

Hide DFA. In our basic construction, we require that the description of the DFA associated with a given search token should be masked with respect to the corresponding keyword strings as well as the states information. To hide these information, we may leverage an efficient and secure way, i.e. using pseudorandom function technique [17]. For each DFA, a data receiver can choose two random pseudorandom keys, say $key^{(1)}$ and $key^{(2)}$, and further replace each keyword character w_i and each state q_j with the values $\xi_i^{(1)} = PRF(key^{(1)}, w_i, i)$ and $\xi_j^{(2)} = PRF(key^{(2)}, q_j, j)$, where PRF is a pseudorandom function, i is the i-th position of the string W, and j is the j-th position of a successful path. Therefore, the DFA description M^* includes a set of masked states $Q^* = \{\xi_j^{(2)} | j \in [1, |Q|]\}$, a set of transactions $\mathcal{T}^* = \{\forall t = (\xi_i^{(2)}, \xi_{i+1}^{(2)}, \xi_i^{(1)}) \in \mathcal{T}\}$, the start state $\xi_0^{(2)}$ and a set of accept states $F^* = \{\xi_j^{(2)} | j \in [1, |F^*|]\}$.

One might think that it is easy to employ our masked technique into some existing ABE systems so as to propose another expressive searchable encryption system as well. However, this is not always the truth. We can take [6] as an example. We may choose to hide a predicate I into a random value set as well, where I is the search predicate and also an output of a ciphertext. By masking the predicate, we guarantee that no one (except the pseudorandom key holder) knows the values in I. Nevertheless, this indeed limits the capability of search. It is not difficult to see that in the search test algorithm of [6], a cloud server needs to know the information of a subset S (which is a set of non-wildcard value of I). Now I cannot be seen by the server, such that S is unknown as well. This definitely will lead to a fail search test. From this example, we can see that not all existing ABE systems show their potential to be extended to become a searchable encryption system.

Search Token Generation. From the description of our basic construction, we can see that the construction is based on somewhat "ABE" infrastructure where there exists a fully trusted authority working as a private key generator with knowledge of msk. With the help of this trusted authority, a data receiver is able to generate a search token for a specified DFA. In this "ABE" infrastructure, the secure interaction with the trusted authority is necessary for the generation of a search token. To eliminate the interaction, we may choose to extend the construction in the identical way as [6]. Namely, we allow user to become a trusted authority for itself to generate msk as its personal secret key. Specifically, we let each system user generate its own $msk = (\hat{g}^{-\alpha}, \hat{z})$ to make it become a personal secret key component and meanwhile, $e(g, \hat{g}^{\alpha}), z$ will become the corresponding public key component for the same user. We state that this additional secret key msk only is for the user who generates it, and other users cannot gain access to it. This basically is identical to public key based system - other users cannot reach one's secret key. It is clear to see that a data receiver (i.e. a valid decryptor) can generate his/her own search token with knowledge of its secret key msk, such that we do not need the help of the trusted authority here. This definitely reduces the communication cost of the system. We state this slight revision for the construction will not yield any effect in keyword matching test and data encryption phases.

Offline Keyword-Guessing. Most of existing public key based searchable encryption cannot hold against offiline keyword-guessing attacks. In the attack, an adversary is allowed to obtain many search tokens, for example, a search token corresponding to a keyword string W. The adversary can also run the encryption algorithm intaking a keyword string K' and a public key of a user to achieve a ciphertext. With the token and the ciphertext, the adversary can proceed to the test algorithm to verify whether the token is embedded with K or not (if the algorithm outputs 1 that indicates $K = K'$). The reason of launching this type of attack is that the encryption algorithm and the corresponding public key is publicly known. This is also the typical property of asymmetric encryption mechanism. We state that our system (as well as [6], [31]) cannot hold against this attack due to their natural public key based property. How to prevent public key based searchable encryption against the attack is an unsolved interesting problem.

V. Evaluation

A. Security Analysis

In this section, we present security analysis for our system. Since the security of the system includes two aspects: keyword privacy for ciphertext, and for search token, we deliver two respective analysis below.

Theorem 1: Our S-DFA-FE system achieves keyword privacy under the asymmetric l-Expanded BDHE assumption.

Proof: Here we first present an analysis in practical point of view, and next deliver a theoretical analysis.

Practical Analysis. We closely take a look at a ciphertext CT associated with a search description string W. Recall that we embed each character w_i of the string into the i-th element of the ciphertext: $C_{i,1} = g^{s_i}$, $C_{i,2} = h_{w_i}^{s_i} z^{s_{i-1}}$ and $C_{i,3} = f^{s_i}$. Given these elements, a PPT adversary may choose to make brute force pairings calculation on them so as to find out which element corresponds to which character. The adversary can first try to calculate $e(C_{i,2} = h_{w_i}^{s_i} z^{s_{i-1}}, \hat{g}) = e(h_{w_i}^{s_i}, \hat{g}) e(z^{s_{i-1}}, \hat{g})$. It is easy to see that the equation is equal to $e(\hat{h}_{w_i}, g^{s_i}) e(\hat{z}, g^{s_{i-1}})$. Here all components are known by the adversary except $\hat{z}$. Without knowledge of $\hat{z}$, the adversary cannot make a successful pairings match, such that it cannot tell whether $e(h_{w_i}^{s_i} z^{s_{i-1}}, \hat{g})$ contains $\hat{h}_{w_i}$ or not. Furthermore, it is not difficult to see that even the adversary replaces $\hat{g}$ in his first calculation with $\hat{h}_{start}$, $\hat{h}_{end}$ or $\hat{f}$, it still cannot have successful pairings match.

On the other hand, the adversary may choose to guess all characters of the string W as it knows the length l of W. Since we allow all system insiders or outsiders to know the finite alphabet $\sum$, the adversary has knowledge of $\sum$ and $|\sum|$. If the adversary tries to make a successful guess for this l length string, its probability is $|\sum|^{-l}$. We state that when $|\sum|$ and l are sufficient large, the probability trends to negligible.

Theoretical Analysis. In the theoretical proof, we will reduce the keyword privacy to the hard problem of asymmetric l-Expanded BDHE. The proof here will share many similarities with that of [29], we below hence mainly present the differences in our proof. We note that the game challenger is $\mathcal{B}$ given the problem instance of asymmetric l-Expanded BDHE.

- Initialization. The adversary outputs W_0^* and W_1^* with equal length l^*.
- Setup. $\mathcal{B}$ sets $w^* = W_b^*$, where $b \in \{0,1\}$. $\mathcal{B}$ chooses random elements v_z, v_{start}, $v_{end}, \beta_f \in \mathbb{Z}_p^*$, and $\forall \sigma \in \sum$ chooses $v_\sigma \in \mathbb{Z}_p^*$. It then sets $e(g,\hat{g})^{\alpha} = e(g^a, \hat{g}^b)$, $g = g$, $\hat{g} = \hat{g}$, $z = g^{v_z} g^{ab/d}$, $\hat{z} = \hat{g}^{v_z} \hat{g}^{ab/d}$, $h_{start} = g^{v_{start}} \prod_{j\in[1,l^*]} g^{-a^j b/c_j}$, $\hat{h}_{start} = \hat{g}^{v_{start}} \prod_{j\in[1,l^*]} \hat{g}^{-a^j b/c_j}$, $h_{end} = g^{v_{end}} \prod_{j\in[2,l^*+1]} g^{-a^j b/c_j}$, $\hat{h}_{end} = \hat{g}^{v_{end}} \prod_{j\in[2,l^*+1]} \hat{g}^{-a^j b/c_j}$, $\forall \sigma \in \sum$, $h_\sigma = g^{v_\sigma} g^{-b/d} \prod_{j\in[0,l^*+1] s.t. w_j^* \neq \sigma} g^{-a^{(l^*+1-j)} b/c_{(l^*+1-j)}}$, $\hat{h}_\sigma = \hat{g}^{v_\sigma} \hat{g}^{-b/d} \prod_{j\in[0,l^*+1] s.t. w_j^* \neq \sigma} \hat{g}^{-a^{(l^*+1-j)} b/c_{(l^*+1-j)}}$, $f = g^{\beta_f}$ and $\hat{f} = \hat{g}^{\beta_f}$. $\mathcal{B}$ outputs $mpk = (e(g,\hat{g})^{\alpha}, g, \hat{g}, z, f, \hat{f}, h_{start}, h_{end}, \hat{h}_{start}, \hat{h}_{end}, \forall \sigma \in \sum h_\sigma, \hat{h}_\sigma)$.
- Phase 1.
  1) Search Token Queries. $\mathcal{B}$ constructs search token much like the secret key construction in the proof of [29]. $\hat{D}_k$ will be represented as $\prod_{i\in S_k} \hat{g}^{a^{i+1} b}$ below. For each $q_k \in Q$, we need a set S_k of indices between 0 and l^*. For i=0,1,...,l^*, we put $i \in S_k$ if and only if $w^{*(i)}$ matches M_k, where $k \in [0, |Q| - 1]$, $M_k = (Q, \mathcal{T}, q_k, F)$ and $w^{*(i)}$ denotes last i symbols of w^*. $\mathcal{B}$ starts by implicitly setting $r_{start} = \sum_{i\in S_0} c_{i+1}$, such that $K_{start2} = \hat{g}^{r_{start}} \hat{g}^{\theta_1} = (\prod_{i\in S_0} \hat{g}^{c_{i+1}}) \hat{g}^{\theta_1}$, $K_{start1} = \hat{D}_0 (\hat{h}_{start})^{r_{start}} \bar{D}_0 (\hat{h}_{start}^{\theta_1}) = (K_{start2})^{v_{start}} (\prod_{j\in[1,l^*], i\in S_0, j\neq i+1} \hat{g}^{-a^j b c_{i+1}/c_j}) \bar{D}_0 (\hat{h}_{start})^{\theta_1}$, $K_{start3} = \hat{g}^{\theta_2}$, where θ_1, θ_2 and $\bar{D}_0$ is chosen by $\mathcal{B}$. For all $q_x \in F$, $\mathcal{B}$ implicitly sets $r_{end_x} = \sum_{i\in S_x, i\neq 0} c_{i+1}$, such

that $K_{end_{x,2}} = \hat{g}^{r_{end_x}} \hat{g}^{\theta_{3,x}} = (\prod_{i \in S_x, i \neq 0} \hat{g}^{c_{i+1}}) \hat{g}^{\theta_{3,x}}$, $K_{end_{x,1}} = \hat{g}^{-\alpha} \hat{D}_x (\hat{h}_{end})^{r_{end_x}} \bar{D}_x (\hat{h}_{end})^{\theta_{3,x}} = (K_{end_{x,2}})^{v_{end}} (\prod_{j \in [2,l^*+1], i \in S_x, i \neq 0, j \neq i+1} \hat{g}^{-a^j b c_{i+1}/c_j}) \bar{D}_x (\hat{h}_{end})^{\theta_{3,x}}$, where $\theta_{3,x}$ and $\bar{D}_x$ are chosen by $\mathcal{B}$. The construction of the components for each $t = (x, y, \sigma) \in \mathcal{T}$ are similar to that of [29] except that $\mathcal{B}$ will additionally multiply $\bar{D}_x^{-1}$, $\bar{D}_y$ and $\hat{f}^{\theta_2}$ to $K_{t,1,i}$ and $K_{t,3,i}$, respectively.

2) Test queries. $\mathcal{B}$ first generates the search token as in the previous step, and next runs the Test algorithm to output 1 or 0.

- Challenge Phase. $\mathcal{B}$ implicitly sets $s_i = sa^i$ and sets $C_1 = H(T)$. It next sets $C_{start1} = g^s$, $C_{start2} = (g^s)^{v_{start}} \prod_{j \in [1,l^*]} g^{-a^j bs/c_j}$; for $i = 1$ to l^*: $C_{i,1} = g^{a^i s}$, $C_{i,2} = (g^{a^i s})^{v_{w_i^*}} (g^{a^{i-1} s})^{v_z} \prod_{j \in [0,l^*+1] s.t. w_j^* \neq w_i^*} g^{(-a^{l^*+1-j+i}) bs/c_j}$, $C_{i,3} = g^{a^i s \beta_f}$; finally sets $C_{end1} = g^{a^{l^*} s}$ and $C_{end2} = (g^{a^{l^*} s})^{v_{end}} \prod_{j \in [2,l^*+1]} g^{-a^{l^*+j} bs/c_j}$.
- Phase 2. Same as Phase 1.
- Guess. $\mathcal{A}$ outputs a guess bit b'. If $b = b'$, $\mathcal{B}$ outputs 1 (guessing $T = e(g, \hat{g})^{a^{l+1} bs}$); else, it outputs 0 (guessing $T \in_R \mathbb{G}_T$).

This completes the simulations. For the challenge ciphertext, if $T = e(g, \hat{g})^{a^{l+1} bs}$, the ciphertext is a valid ciphertext of W_b^*, such that the probability of the adversary in outputting $b = b'$ is $1/2 + \mu$. Otherwise, T is a random element in $\mathbb{G}_T$ such that the ciphertext is a random one. The probability of the adversary to guess a correct b' is $1/2$. Therefore, the probability of $\mathcal{B}$ correctly decides a T with the problem instance is $1/2(1/2 + \mu + 1/2) = 1/2 + \mu/2$. $\mathcal{B}$ can solve the asymmetric l-Expanded BDHE problem with advantage $\mu/2$ if the adversary can win the keyword privacy game with advantage μ. ■

Theorem 2: Given a search token, a PPT adversary cannot compromise the information of characters embedded into the DFA under the assumption of the psuedorandom function (we use in masking DFA) is secure.

Proof: We suppose an adversary can break the security of pseudorandom function with probability adv^{PRF}. Since all characters are masked as $\xi_i^{(1)}$, the adversary either needs to break the PRF or correctly guesses the pseudorandom key $key^{(1)}$, where $key^{(1)}$ belongs to a valid key space $\{0,1\}^{poly(1^n)}$. Therefore, the adversary may know the embedded characters with probability $adv^{PRF} + 2^{-poly(1^n)}$. On the other hand, it is possible for the adversary to guess the characters by unfolding the information of all states. We recall that inside a DFA given an origin state and a character as input, it deterministically outputs a destination state. Besides, the alphabet $\sum$ is known by the adversary. The adversary hence can try all unknown characters from $\sum$ to determine a transaction $t_i = (x_i, y_i, w_i)$. Similarly, the adversary can reveal all states information with probability $adv^{PRF} + 2^{-poly(1^n)}$, where $2^{-poly(1^n)}$ is the probability of guessing the pseudorandom key $key^{(2)}$. In summary, the adversary can achieve the knowledge of characters embedded into the DFA with probability $adv = 2adv^{PRF} + 2^{1-poly(1^n)}$. Since we assume the psuedorandom function is secure, we have that adv^{PRF} is negligible. Therefore, adv is negligible as $2^{1-poly(1^n)}$ is negligible with a sufficient large n. ■

B. Performance Analysis

As we mentioned in the introduction part, Boneh and Waters [6] proposed an expressive searchable encryption supporting conjunctive, subset and range search, while Zheng, Xu and Ateniese [31] introduced a key-policy attribute-based system supporting keyword search. Here, we make a comprehensive comparison among our construct, [6] and [31] in terms of computation and communication cost. Table IV shows the comparison of computational cost, and Table V shows the communication comparison.

We now define the notations used in the Tables. Let $|\mathbb{G}|$ and $|\mathbb{G}_T|$ denote the bit-length of an element in groups $\mathbb{G}$ and $\mathbb{G}_T$, l denote the length of a keyword search string, $|TL|$ denote the number of leaf in an access tree, $|S|$ denote the number of attribute in an attribute set, $c_p, c_{ex}^{(1)}, c_{ex}^{(2)}$ denote the computation cost of a bilinear pairing, an exponentiation in $\mathbb{G}$, and an exponentiation in $\mathbb{G}_T$, respectively. Note that we regard $\mathbb{G}_1$ and $\mathbb{G}_2$ (of our system) as a $\mathbb{G}$ in the theoretical analysis for simplicity purpose. However, we will treat the two groups in a different way in the following up practical analysis. Similarly, the subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$, $\mathbb{G}_{T,p}$ and $\mathbb{G}_{T,q}$ (of [6] whereby $n = pq$) will be simply seen as $\mathbb{G}$ and $\mathbb{G}_T$ in the theoretical analysis only, respectively.

To make a clear and fair comparison, we further make the following assumptions. Since [6], [31] and our system provide different functionalities in keyword search, we need to define a common function among them for a fair comparison. We hence define the comparison is based on a single equality keyword match. We assume our system shares the same l with [6], namely, the length of keyword string in our system is equal to that of keyword searchable field in [6]. Here we also have $|S| = l$ for an equality match test in [6]. In [31], we will set $|TL|$ (i.e. the number of attribute in an access tree) and $|Attr|$ (i.e. the number of attribute embedded into a ciphertext) to be equal to l as well, although they are defined by a specific access policy but not any information of a search keyword. We note that it can be seen from this point of view that the efficiency of keyword search of [31] mainly depends on the complexity of access policy. If a complex access policy is used, there must be a low search efficiency. This can be regarded one of the limitations in [31]. For our DFA-based construction, a single keyword match indicates that a designed DFA needs one and only successful path from an initial state to a unique accept state.

Before proceeding to the theoretical complexity analysis, we first present the complexity assumption comparison among our scheme, [6] and [31] in Table III. By ROM we mean random oracle model. From the Table I and Table III, we can see that the efficiency of [31] may be better than that of [6] and ours, since [31] is built in the ROM with symmetric pairings and prime order group. This fact will be further confirmed and shown in our practical analysis.

From Table IV, we observe that our scheme shares the same efficiency with [31], while [6] suffers from linearly cost in the

TABLE III: Complexity Assumption Comparison with [6], [31]

Schemes	Complexity Assumption	Pairings Group
[6]	Composite 3-party Diffie-Hellman	composite order group
[31]	Decisional Linear	symmetric prime order group
Ours	Asymmetric l-Expanded BDHE	asymmetric prime order group

TABLE IV: Computation Comparison with [6], [31]

Schemes	Computation Cost		
	Enc	Token Gen	Test
[6]	$O(l)c_{ex}^{(1)} + O(1)c_{ex}^{(2)}$	$O(l)c_{ex}^{(1)}$	$O(l)c_p$
[31]	$O(l)c_{ex}^{(1)}$	$O(l)c_{ex}^{(1)}$	$O(l^2)c_p + O(l)c_{ex}^{(2)}$
Ours	$O(l)c_{ex}^{(1)} + O(1)c_{ex}^{(2)}$	$O(l)c_{ex}^{(1)}$	$O(l)c_p$

test phase, i.e. $O(l^2)$ pairings. It is worth mentioning that our scheme supports more powerful search functionality than [31].

TABLE V: Communication Comparison with [6], [31]

Schemes	Size/Length	
	Ciphertext	Token
[6]	$O(1)\|\mathbb{G}_T\| + O(l)\|\mathbb{G}\|$	$O(l)\|\mathbb{G}\|$
[31]	$O(l)\|\mathbb{G}\|$	$O(l)\|\mathbb{G}\|$
Ours	$O(l)\|\mathbb{G}\|$	$O(l)\|\mathbb{G}\|$

Table V shows that the systems have similar complexity in the size of search token, while [6] needs an extra $\mathbb{G}_T$ element in the ciphertext metric.

In conclusion, our scheme achieves regular language keyword search without requiring a great amount of additional computation and communication cost.

C. Practical Analysis

For the system simulation, we leverage the Java Pairing Based Cryptography Library [23] to calculate the system running time shown in Table VI. Our testbed is: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz, 3 GB RAM, Ubuntu 10.04. For the fairness of the practical comparison, we will use different pairing types - one is Type a with 160-bit group order (the embedding degree of the curve is 2) for the implementation of [31]'s KP-ABKS scheme; one is Type $a1$ with 1024 bits based field size and $k = 2$ for the implementation of [6]'s hidden vector encryption construction; and one is Type d with 159 bits based field and $k = 6$ for the implementation of our system (in which we assume a search token only has one final successful state). We note that the above pairing types are chosen based on the recommendation introduced in [24], and all the data is without preprocessing. We suppose all schemes listed in the Tables must at least achieve a security level comparable to a symmetric key cryptosystem with an 80-bit key. That is, an elliptic curve cryptosystem with around 160-bit key is needed. Therefore, we set $n = 160$ bits, the group elements in $\mathbb{G}_{\xi_1}$ are set to be 160 bits, and the group elements from $\mathbb{G}_T$ and $\mathbb{G}_{T,\xi_2}$ are set to be 1024 bits, respectively, where $\xi_1 \in \{1, 2, p, q\}$ and $\xi_2 \in \{p, q\}$. We further set the following four experimental samples: Test 1: $l = 10$; Test 2: $l = 30$; Test 3: $l = 60$; Test 4: $l = 100$. Table VII is the comparison of concrete communication cost.

TABLE VI: Comparison in System Running Time

Schemes	Algorithms (Running Time ms)		
	Enc	Token Gen	Test
[6]	$Test_1$: 20628.458	16989.681	6830.796
	$Test_2$: 60491.738	50302.781	19841.836
	$Test_3$: 120286.658	100272.431	39358.396
	$Test_4$: 200013.218	166898.631	65380.476
[31]	$Test_1$: 260.232	446.112	344.208
	$Test_2$: 631.992	1189.632	974.008
	$Test_3$: 1189.632	2304.912	1918.708
	$Test_4$: 1933.152	3791.952	3178.308
Ours	$Test_1$: 242.204	1151.185	1224.16
	$Test_2$: 664.204	3124.645	3322.72
	$Test_3$: 1297.204	6084.835	6470.56
	$Test_4$: 2141.204	10031.755	10667.68

TABLE VII: Comparison in Communication Cost

Schemes	Components Length (bit)		
	Public Key	Ciphertext	Token
[6]	$Test_1$: 11264	7744	3360
	$Test_2$: 30464	20544	9760
	$Test_3$: 59264	39744	19360
	$Test_4$: 97664	65344	32160
[31]	$Test_1$: 640	2080	3520
	$Test_2$: 640	5280	9920
	$Test_3$: 640	10080	19520
	$Test_4$: 640	16480	32320
Ours	$Test_1$: 5664	5600	5600
	$Test_2$: 12064	15200	15200
	$Test_3$: 21664	29600	29600
	$Test_4$: 34464	48800	48800

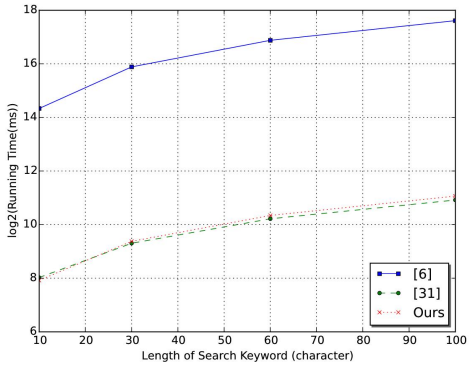

Fig. 2: Running Time Comparison in Encryption

To clearly show the comparison, we use the line charts to depict the experimental results below. Fig. 2, 3 and 4 show the running time in encryption, search token generation and keyword matching test, while Fig. 5, 6 and 7 present the comparison results in the length of public key, ciphertext and

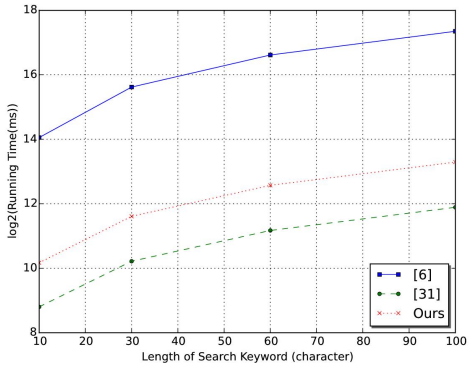

Fig. 3: Running Time Comparison in Token Generation

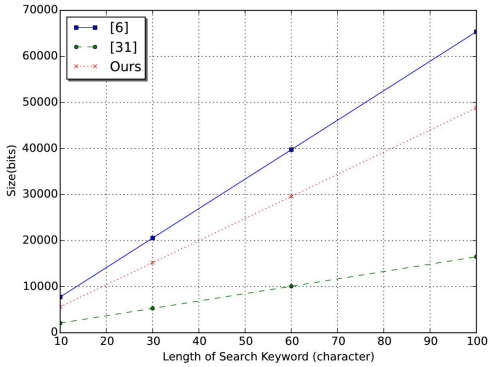

Fig. 6: Comparison for Ciphertext Size

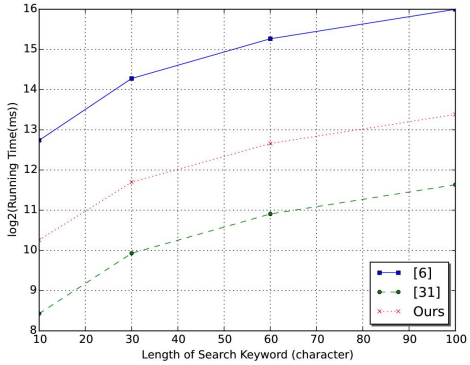

Fig. 4: Running Time Comparison in Test

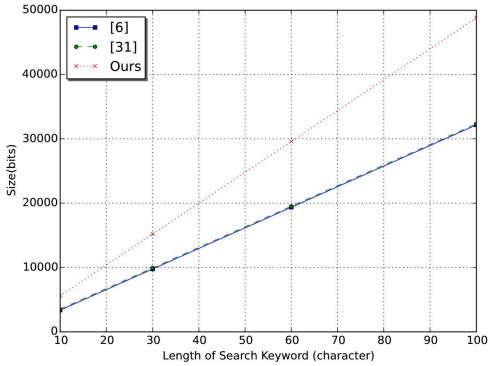

Fig. 7: Comparison for Token Size

search token. We note that the unit of the vertical axis of Fig. 2, 3 and 4 is millisecond, and that of others is bit.

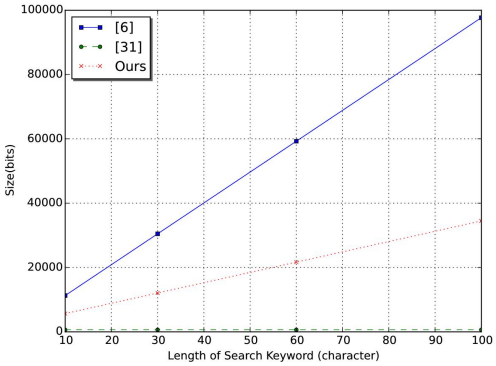

Fig. 5: Comparison for Public Key Size

From the running time experimental results, we can see that all systems experience a climbing trend with the increase of l. Specifically, [6] suffers from significantly increase of running time in encryption, keyword token generation and test algorithms. However, our system and [31] have small range of ascendancy. In addition, the systems share approximately the same running cost in each metric.

As illustrated in the running time figures, while the search keyword length is ≤ 100, our system achieves efficient and acceptable complexity in encryption, decryption and keyword search. Based on this fact, we set some limitations for our system while it is implemented in practice. To avoid heavy complexity in search queries, we may present a limitation for the design of DFA that each valid path (with direction inside the DFA) may intake less than 100 symbols for each search query. This constraint, actually, limits the expressiveness of AND/Not gate(s) search query. Recall that one direction path is seen as a query with AND/Not gates in our system. Besides, it is better to limit at most 5-6 OR gates in the design of DFA.

From the size of public key comparison (Fig. 5), we can see that [31] enjoys the constant cost as the increase of l. This is because it only supports a single keyword equality match but not expressive search. [6] and our system aim to present more expressive search query such that we need to represent each character element individually. This basically explains the reason why more expressiveness requires more complexity. Fig. 6 shows that although our system requires more space for the storage of ciphertext as compared to [31], it outperforms [6]. In Fig. 7, we see that our system suffers from the largest storage cost for search token, while other systems have similar space cost (two lines are almost overlapped).

We state that our system needs larger space for search token is due to a reason that the DFA associated with the token provides a fine-grained character (which is from a given alphabet) match pattern. In the designed structure of the DFA, each character is seen as an individual component. We note

that the pattern and the structure is the premise of allowing us to achieve regular language search. However, the premise delivers us shortage in search token storage cost. How to reduce the size of search token without loss of expressiveness for keyword search is an interesting open problem, which is our future work.

VI. Conclusions

We designed a novel searchable encryption system supporting regular language search and privacy-preserving property for search content as well as description content for ciphertext. To introduce our construction, we started with Waters functional encryption system, and further extended it to achieve anonymous property for ciphertext. We next redesigned the DFA structure and made use of it to support regular language search. We meanwhile presented extensive analysis to show that our system is provably secure and efficient. This paper motivates some open problems for our future works. We need to shorten the size of search token without loss of expressiveness, and to secure the system from offline keyword-guessing attacks.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] J. Baek, R. Safavi-Naini, and W. Susilo. On the integration of public key data encryption and public key encryption with keyword search. In*ISC*, vol. 4176 of *LNCS*, pp. 217–232. Springer, 2006.

[3] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, vol. 4622 of *LNCS*, pp. 535–552. Springer, 2007.

[4] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, vol. 6841 of *LNCS*, pp. 111–131. Springer, 2011.

[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, vol. 3027 of *LNCS*, pp. 506–522. Springer, 2004.

[6] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, vol. 4392 of *LNCS*, pp. 535–554. Springer, 2007.

[7] C. Bösch, A. Peter, B. Leenders, H. W. Lim, Q. Tang, H. Wang, P. H. Hartel, and W. Jonker. Distributed searchable symmetric encryption. In *PST*, pp. 330–337. IEEE, 2014.

[8] C. Bösch, Q. Tang, P. H. Hartel, and W. Jonker. Selective document retrieval from encrypted database. In *ISC*, vol. 7483 of *LNCS*, pp. 224–241. Springer, 2012.

[9] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *PKC*, vol. 5443 of *LNCS*, pp. 196–214. Springer, 2009.

[10] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.*, 25(1):222–233, 2014.

[11] 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*. The Internet Society, 2014.

[12] 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*, vol. 8042 of *LNCS*, pp. 353–373. Springer, 2013.

[13] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, vol. 3531 of *LNCS*, pp. 442–455, 2005.

[14] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, vol. 6477 of *LNCS*, pp. 577–594. Springer, 2010.

[15] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, January 2004.

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

[17] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[18] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In *ACNS*, vol. 3089 of *LNCS*, pp. 31–45. Springer, 2004.

[19] Y. Hwang and P. Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *Pairing*, vol. 4575 of *LNCS*, pp. 2–22. Springer, 2007.

[20] V. Iovino and G. Persiano. Hidden-vector encryption with groups of prime order. In *Pairing*, vol. 5209 of *LNCS*, pp. 75–88. Springer, 2008.

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

[22] K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *FC*, vol. 7397 of *LNCS*, pp. 285–298. Springer, 2012.

[23] J. Library. http://gas.dia.unisa.it/projects/jpbc/benchmark.html#.U5FXwZS1bLd/, 2013. Online; accessed 18-March-2015.

[24] B. Lynn. *On the Implementation of Pairing-based Cryptosystems*. PhD thesis, Stanford University, June 2007.

[25] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, vol. 5444 of *LNCS*, pp. 457–473. Springer, 2009.

[26] E. Shi, J. Bethencourt, H. T. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *S&P*, pp. 350–364. IEEE Computer Society, 2007.

[27] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *S&P*, pp. 44–55. IEEE Computer Society, 2000.

[28] C. Wang, K. Ren, S. Yu, and K. M. R. Urs. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *INFOCOM*, pp. 451–459. IEEE, 2012.

[29] B. Waters. Functional encryption for regular languages. In *CRYPTO*, vol. 7417 of *LNCS*, pp. 218–235. Springer, 2012.

[30] P. Xu, H. Jin, Q. Wu, and W. Wang. Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. *IEEE Trans. Computers*, 62(11):2266–2277, 2013.

[31] Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *INFOCOM*, pp. 522–530. IEEE, 2014.

Appendix

A. Definition of DFA

We here give a overview of DFA as of [29]. A DFA M is a five-tuple $(Q, \sum, \delta, q_0, F)$ in which:

1) Q is a set of states.
2) $\sum$ is a finite set of symbols called the alphabet.
3) $\delta : Q \times \sum \rightarrow Q$ is a function known as a transition function.
4) $q_0 \in Q$ is called the start state.
5) $F \subseteq Q$ is a set of accept states.

Note we use $\mathcal{T}$ to denote the set of transitions associated with the function δ, where $t = (x, y, \sigma) \in \mathcal{T}$ iff $\delta(x, \sigma) = y$. Suppose that $M = (Q, \sum, \delta, q_0, F)$. M accepts a string $w = w_1, w_2, ..., w_l \in \sum^*$ if there is a sequence of states $r_0, r_1, ..., r_n \in Q$ where:

1) $r_0 = q_0$;
2) For $i = 0$ to $n - 1$ $\delta(r_i, w_{i+1}) = r_{i+1}$;
3) $r_n \in F$.

Note we use $ACCEPT(M, w)$ to denote that the machine M accepts w, and $REJECT(M, w)$ to denote that M does not accept w. A DFA M recognizes a language L if M accepts all $w \in L$ and rejects all $w \notin L$; such a language is called regular.