This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

Efficient Encrypted Data Search with Expressive Queries and Flexible Update

Jianting Ning, Jiageng Chen, Kaitai Liang, *Member, IEEE,* Joseph K. Liu, Chunhua Su, and Qianhong Wu

Abstract—Outsourcing encrypted data to cloud servers that has become a prevalent trend among Internet users to date. There is a long list of advantages on data outsourcing, such as the reduction cost of local data management. How to securely operate encrypted data (remotely), however, is the top-rank concern over data owner. Liang et al. proposed a novel encrypted cloud-based data share and search system without loss of privacy. The system allows users to flexibly search and share encrypted data as well as updating keyword field. However, the search complexity of the system is of extreme inefficiency, $O(nd)$, where d is the total number of system files and n is the size of query formula. This paper, for the first time, leverages the "oblivious cross search" technology in public key searchable encryption context to reduce the search complexity to *only* $O(nf(w))$, where $f(w)$ is the number of files embedded with the "least frequent keyword" w. The new scheme maintains efficient encrypted data share and keyword field update as well. This paper further revisits the security models for payload security, keyword privacy and search token privacy (i.e. search pattern privacy) and meanwhile, presents security and efficiency analysis for the new scheme.

Index Terms—Secure data search, share, flexible query, update, efficiency.

1 Introduction

Searchable encryption (SE) [30] enables a data owner to fulfill search over encrypted outsourced data without loss of data and query secrecy. The data owner needs to build an encrypted search index structure for a cloud server. It next can generate a search token with its secret information, so that the server can locate and return all encrypted data matching the query without knowing the exact "contents" of query and the underlying data. SE is applicable to many real-world cloud applications (e.g. CipherCloud). So far, it is unknown that if there exists a trivial way to combine SE with other secure sharing and computing technologies (e.g. proxy re-encryption and homomorphic encryption), so that a data owner may flexibly operate (e.g., sharing, searching and even computing) its encrypted data stored in cloud [24].

To partially resolve the unknown, an attribute-based searchable proxy re-encryption scheme is proposed in [20]. It is the first of its type to fill the difficulties and technical gaps between SE and proxy re-encryption. Its computation and communication complexity, however, strongly depend on the size of attribute set/policy. Specifically, the size of system public key, re-encryption key and search token are linear in the size of attribute set and furthermore, data share leads to the expansion of ciphertext size. Besides, the keyword update is not flexible enough as the system only allows the update to be done in data sharing stage. The efficiency bottleneck and flexible keyword update are two interesting open problems left by [20].

To tackle the problems, Liang et al. [19] proposed a searchable proxy re-encryption with flexible keyword update in the context of identity-based encryption, in which system users can update keyword at any time and meanwhile, data search and share complexity are much more efficient than that of [20].

Motivation. Although outperforming [20] in the merit of data search, share and keyword update, [19] still cannot achieve "efficient" search. Specifically, the scheme requires the complexity $O(nd)$ to fulfill a formula query, where n is the size of the formula and d is the total number of encrypted files. In other words, a cloud server has to check through all encrypted files for a single keyword search, i.e. $O(d)$ where $n = 1$. While $n > 1$, the server needs to repeat the above throughout check n times. The search complexity then is linear with the size of the formula as well as that of encrypted database. In practice, response time or online client waiting time is significantly related to the search efficiency. How to reduce the complexity that mainly motivates this work. Besides, this work will attempt to consider the privacy of search token.

Contributions. While deploying more expressive search (e.g. conjunctive), the public key based SE (PKSE) technology leads to expensive computation and communication cost (e.g. [5] with composite order group). It is hard to resolve the efficiency bottleneck beyond the search expressiveness in the context of PKSE.

Inspired by the "oblivious cross search" technique introduced in [7], we propose a feasible solution, to some extent,

J. Ning is with the school of Mathematics and Computer Science, Fujian Normal University, China, and School of Information Systems, Singapore Management University, Singapore (email: jtning88@gmail.com).
J. Chen is with the Computer School, Central China Normal University, China (email: jiageng.chen@mail.ccnu.edu.cn)
K. Liang is with the Department of Computer Science, University of Surrey, U.K. (email: k.liang@surrey.ac.uk)
J. K. Liu is with the Faculty of Information Technology, Monash University, Australia (email: joseph.liu@monash.edu)
C. Su is with the School of Information Science, Japan Advanced Institute of Science and Technology, Japan (email: suchunhua@gmail.com)
Q. Wu is with the School of Electronics and Information Engineering, Beihang University, China (email: qianhong.wu@buaa.edu.cn)
This is the extended version of the conference paper [19].

to fill the gap between efficiency and search expressiveness in this paper. By using the solution, we reduce the search complexity of [19] from $O(nd)$ to $O(nf(w))$. Our technical roadmap is described as follows. We employ $TSet$ and $XSet$ into our scheme, in which $TSet$ is an encrypted and scrambled set for database $DB = (m_i, W_i)_{i=1}^{d}$, and $XSet$ can be seen as an extra "linked" level between keywords and files. We create "links" between keywords and files in $TSet$, and similarly, generate the "copy" links in $XSet$. The files here become the connection between $TSet$ and $XSet$. Much like the idea presented in [7], our solution leverages the search in the form of $Q = w_1 \wedge \xi(w_2, ..., w_n)$, where w_1 is "least frequent keyword" that is used to identify the "smallest" encrypted files set, and ξ is the unconstrained query formula. The corresponding encrypted files of w_1 can be located in $TSet$, while the query ξ is performed in $XSet$. Since the connection between the two sets are the files, if there are existing some files satisfying the search in both $TSet$ and $XSet$, they are the output of the query Q. The search complexity now is proportional to the number of files embedded with w_1.

To allow one to search some single keyword in $TSet$, we reuse the $TSet$ instantiation in [7]. To achieve keyword update, we need to redesign the search token, and to revise $TSet$ instantiation. Note that $TSet$ contains many $T[w]$ (which will be introduced later), in which a $T[w]$ corresponds to all encrypted files embedded the same w. Since the encrypted files are stored in $T[w]$ randomly, the server has no chance to know which one will be going to have keyword update later. We mask the "keyword-file link" into extra ciphertext components, so that the server can first locate $T[w]$ and next use pairing computation to find the specified item in $T[w]$ (to be updated). We design an algorithm called $LotAlg$ to fulfill the above "double" trace function. To update $TSet$ accordingly, we add a new algorithm called $UpTSet$ into $TSet$ instantiation. We have to require system user to store the location information of $XSet$ as a matrix locally, so that he/she can inform the server that the item located in the position (i, j) needs to be updated.

This paper is an extension but not trivial incremental progress of the conference version [19]. The contributions of this paper are described as follows.

- In this paper, we mainly follow the system definition given in [19] but with an exception that we revise the data sharing algorithm so as to only allow a data owner to share its encrypted files embedded with a specific keyword with others. Keyword update is necessary no more in the data sharing phase, as user can update the keyword field after being granted the decryption rights of the shared data.
- We define a "weak" search token privacy model matching our system definition in this paper.
- It is the first time to employ SSE's "oblivious cross search" technique into a PKSE scheme. We also effectively integrate the technique with keyword update.
- The new system maintains effective data share, search and keyword update functionalities and meanwhile, it achieves constant size in public key, re-encryption key, search token and ciphertext, and no linearly cost exists in the construction phases of re-encryption key and search token. In other words, we reduce the search complexity without compromising other features and efficiency.
- The security and efficiency analysis show that out system has a relatively better potential in the deployment of large scale database.
- This paper designs algorithms for $TSet$ update and redundancy elimination, which may be of independent interest.

Related Work. Song et al. [30] introduced the first notion of SE. Two main streams of SE have been defined: one is symmetric SE (SSE), and the other is public key based SE (PKSE). Some light-weight cryptographic tools (e.g. pseudorandom function) are usually used in SSE, so that an SEE construction (e.g. [8]) enjoys relatively high search efficiency. In contrast, a PKSE scheme (e.g. [4]) leverages public key technology that yields the loss of efficiency in search. SSE and PKSE have respective pros and cons. SSE cannot easily check data integrity unless data owner downloads all encrypted data from server. Although supporting data integrity check at anytime and by any system user, PKSE suffers from poor search efficiency and less query expressiveness. This paper, for the first time, employs SSE's technique into PKSE so as to tackle the query efficiency and expressiveness issue.

Following [4], Abdalla et al. [1] proposed a generic PKSE construction anonymous identity-based encryption. To date many PKSE variants have been proposed, e.g., authorized keyword search [14], verifiable keyword search [2], fuzzy keyword search [32], conjunctive keyword search [11], range query [29], [5] supporting conjunctive, subset, and range search queries, and attribute-based keyword search [34].

To delegate decryption rights to others, Blaze, Bleumer and Strauss [3] defined the concept of proxy re-encryption (PRE). A PRE scheme enables a semi-trusted proxy to convert a ciphertext of a message intended for a user to another ciphertext of the same message intended for another user without "seeing" the massage. In this paper, we consider a "multiple conversion" case, which we call it multi-hop proxy re-encryption. PRE has been used in various contexts, for example traditional PRE [6], [21], [22], identity-based PRE, e.g., [12], [17], [18], and functional PRE, e.g., [16].

It is unknown that if there is a way to integrate a SSE/PKSE with a PRE scheme to yield a secure protocol. Some difficulties stand in front of the combination. We need to consider the keyword privacy into two ciphertext levels - an original ciphertext as well as its corresponding re-encrypted ciphertext. Besides, we need to guarantee that an adversary with search ability cannot break the payload security for ciphertexts. To combine PKSE with PRE, Shao et al. [28] introduced a new primitive called PRE with keyword search (PREKS). Hu and Liu propose a "search-but-no-decryption" PREKS scheme. These schemes are built based on bidirectional technique to achieve data sharing, but they cannot hold against collusion attacks where a proxy colludes with another system user to reveal data owner's secret key. To eliminate this attack, Fang et al. [10] designed a new PREKS system but with the price that the loss of searchability after data sharing.

A new PREKS is recently proposed in [20]. It is the first of its type to explore PREKS in the attribute-based context. It, however, suffers from heavy search complexity due to the leverage of attribute-based technology to relate the size of search token to the size of attribute set. Similarly, Zheng,

Xu and Ateniese's attribute-based searchable system [34] does also "overkill" the search efficiency for the purpose of achieving attribute property. Knowing the bottleneck of the previous mentioned schemes, Liang et al. [19] proposed a novel system in the context of identity-based to achieve better search efficiency. However, [19] still suffers from linear search complexity. A search for a formula with size n, in [19], has to take $O(nd)$. Besides, the scheme has not considered the search token (i.e. search pattern) privacy yet. This paper targets to tackle the above open problems of [19]. Note some recent research works have been proposed for real-world applications and attacks on SE, e.g., the searchable chain of PKSE [15], passive attacks over SSE [26] and the PKSE applied to Internet of Things [13]. We notice that there exist some researches on SSE over the topics of forward and backward security [31], [36], [37].

Since this work is the first to combine identity-based encryption, PRE, SE and oblivious cross search techniques, it is briefly compared with the most related works, namely the seminal PKSE [4], an identity-based PRE [12], attribute-based PKSE [20], a recent identity-based PKSE [19] and a recent PKSE scheme [13] in Table 1. In the table, $|S|$ denotes the size of attribute set/policy, $\perp$ denotes "not applicable", "rk" denotes re-encryption key, "token" denotes search token and "ROM" is short for random oracle model, respectively. By ciphertext expansion we mean that the size of ciphertext will be expanded after the ciphertext is shared. We use "linear" and "constant" to denote if a given size (e.g., the size of ciphertext) grows linearly with keyword/attribute/re-encryption hops or it is constant (no matter how many numbers of keyword/attribute/hop it has). It can be seen that this work is the first to achieve $O(nf(w))$ efficiency in search without loss of data share and keyword update functionalities in the context of PKSE. We will present the efficiency analysis and practical comparison with [13] in terms of computation and communication cost in Section 4.

2 Problem Statement

2.1 System Entities

• A *data encryptor* forms a database DB as $(m_i, W_i)_{i=1}^d$, next encrypts DB to be EDB, and finally uploads the EDB to a cloud server, where d is the number of files in DB.
• A *data receiver* owns the underlying data of EDB (stored in the cloud) intended for it and meanwhile, it (with the valid decryption rights) can construct search token for the EDB queries. It is allowed to share its EDB to a specified system user. It further can update the keyword field of EDB by delivering the server a keyword update token.
• A *trusted key issue center* is to generate a secret key for each system user. It also takes part in the generation of a special key for decryption rights delegation.
• A *cloud server* stores system users' EDB. Given a search/share/update token, it can locate and return/share/update the corresponding EDB matching the token.

Note that a data encryptor can be also a data receiver for itself, i.e. outsourcing its own encrypted data to the cloud.

2.2 System Algorithms

The system definition is similar to that of [19].

Definition 1. It consists of the following algorithms:
• $(mpk, msk) \leftarrow Setup(1^k)$. On input a security parameter k, the system setup algorithm outputs a master public key mpk and a master secret key msk, where $k \in \mathbb{N}$. Hereafter, we implicitly regard mpk as an input for the following algorithms.
• $(pk_{ID}, sk_{ID}) \leftarrow KeyGen(msk, ID)$. On input msk and an identity ID, the key pair generation algorithm outputs a public key and secret key pair (pk_{ID}, sk_{ID}) for a system user with identity $ID \in \mathbb{Z}_q^*$. Hereafter, we assume that pk_{ID} implicitly includes the identity ID.
• $EDB \leftarrow Enc(pk_{ID}, DB)$. On input a pk_{ID}, and a DB (i.e. $(m_i, W_i)_{i=1}^d$, including the files' identities m_d, and a keyword description set $W \in \{0,1\}^*$, $d = |DB|$), the data encryption algorithm outputs an encrypted EDB.
• $TK \leftarrow TKGen(sk_{ID}, w)$. On input a sk_{ID}, and a keyword description w, the search token generation algorithm outputs a search token TK, which is used to search user ID's EDB with keyword description w.
• $rk_{ID_i \to ID_j|w_i} \leftarrow ReKeyGen(msk, ID_i, ID_j, w_i)$. On input msk, an ID_i, an ID_j, and a w_i, the re-encryption key generation algorithm outputs a re-encryption key $rk_{ID_i \to ID_j|w_i}$ which can be used to share the encrypted files tagged with w_i in ID_i's EDB to ID_j.
• $uptk_{w_i \to w_j} \leftarrow UpTKGen(sk_{ID}, m_d, w_i, w_j)$. On input a sk_{ID}, a file identity m_d, an old keyword description w_i tagged with m_d and a new one w_j, the keyword update token generation algorithm outputs a update token $uptk_{w_i \to w_j}$ which can be used to update the encrypted file m_d in EDB (intended for ID) with the old description w_i to the one with the new one w_j.
• $EDB \leftarrow ReEnc(rk_{ID_i \to ID_j|w_i}, EDB)$. On input a share token $rk_{ID_i \to ID_j|w_i}$ and an EDB, the re-encryption algorithm converts all encrypted files tagged with w_i in ID_i's EDB into those encrypted files of the same message under ID_j and w_i. We note that this conversion maintains keyword update ability for ID_j. In addition, we state that the user ID_i cannot obtain the decryption and search rights of the new (re-encrypted) ciphertext after the conversion.
• $EDB \leftarrow Update(uptk_{w_i \to w_j}, EDB)$. On input a keyword update token $uptk_{w_i \to w_j}$ and an EDB, the keyword update algorithm updates all ciphertexts with an old keyword w_i to those with the new keyword w_j within the given EDB.
• $1/0 \leftarrow Search(TK, EDB)$. On input a search token TK generated by the user with decryption rights on EDB, and an EDB, the search algorithm outputs 1 if they match, and 0 otherwise. We further note that a cloud server will choose to return either the matching ciphertext(s) or nothing to the user based on the outputs of the algorithm.
• $Dec(sk_{ID}, EDB)$. On input a sk_{ID} and an EDB, the ciphertext decryption algorithm outputs all files m_d.

The main system work flow is described as follows.
• The setup phase. A trusted authority first runs the algorithm $Setup$ to generate the mpk for all system users,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

TABLE 1: Comparison with the related works

	Data Share	rk/token size	ciphertext size/expansion	Keyword Update	Search Complexity	Complexity Assumption	Security Model
[4]	$\perp$	$\perp$/ constant	constant/ $\times$	$\times$	$O(nd)$	Computational Bilinear Diffie-Hellman	ROM
[12]	✓	constant/ $\perp$	constant/ ✓	$\times$	$\perp$	Decisional Bilinear Diffie-Hellman	ROM
[20]	✓	linear/ linear	linear/ ✓	✓	$O(\|S\|nd)$	Decisional Bilinear Diffie-Hellman Exponent & Decisional l-Bilinear Diffie-Hellman Exponent	Standard
[19]	✓	constant/ constant	constant/ $\times$	✓	$O(nd)$	generic group model	ROM
[13]	$\times$	constant/ constant	constant/ $\perp$	$\times$	$O(nd)$	Computational Bilinear Diffie-Hellman	ROM
This work	✓	constant/ constant	constant/ $\times$	✓	$O(nf(w))$	Decisional Diffie-Hellman & generic group model	ROM

trusted key issue center and a cloud server, to initialize the system and to keep the msk only for the key issue center.

• The key pair generation phase. The trusted key issue center generates a key pair for a system user via running the algorithm $KeyGen$. A user ID publishes the public key pk_{ID} and keeps sk_{ID} secret.

• The encryption phase. A data encryptor, Bob, runs the algorithm Enc to generate an EDB for Alice, with the corresponding keyword description $K1$, and further uploads the EDB to the cloud server. Note here for simplicity we set $|W| = 1$ and $K1$ is the only keyword in W.

• The data search phase.

1. Alice (with the decryption rights of the EDB) runs the algorithm $TKGen$ and delivers the search token corresponding to $K1$ to the server.
2. The server intakes the token and the EDB to run the algorithm $Search$. If finding a match, the server outputs 1 and returns the corresponding ciphertexts, and outputs 0 and returns nothing otherwise.
3. If receiving a successful return from the server, Alice runs the algorithm Dec with its secret key to recover the underlying message.

• The keyword description update phase.

1. To update keyword description from $K1$ to $K2$, Alice (with the decryption rights of the ciphertext) runs the algorithm $UpTKGen$ to construct a keyword update token $K1 \rightarrow K2$, and next delivers it along with a search token for $K1$ to the server.
2. If there are encrypted files tagged with $K1$, the server runs the algorithm $Update$ intaking the token and the EDB to update the keyword description to $K2$.

• The ciphertext share phase.

1. To share its encrypted data tagged with $K1$ to Carol, Alice can generate a re-encryption key from Alice $\rightarrow$ Carol with the help of the trusted key issue center by running the algorithm $ReKeyGen$.
2. If there are encrypted files tagged with $K1$, the server runs the algorithm $ReEnc$ to convert the ciphertext's decryption rights to Carol.
3. Carol can retrieve the shared files by searching.

2.3 Threat Models

We define the data confidentiality model, the keyword privacy model and search token privacy model below. Generally speaking, the first model is used to guarantee that a Probabilistic Polynomial Time (PPT) adversary cannot compromise the information of a message by given an encryption of the message; the second model is to ensure a PPT adversary cannot reveal the keyword embedded in a given ciphertext; the last model is for preventing a PPT adversary from extracting the keyword embedded into a given search token. Note we will consider the privacy of keyword update token as well. This can be captured in the keyword privacy model.

We assume that cloud server, data encryptor and data receiver are honest-but-curious, while the authority for system initialization and the key issue center are fully trusted. We further assume either data encryptor or data receiver will not collude with the server to reveal the underlying keyword description and database. Note that we here leave the collusion attacks for our future work. By honest-but-curious (i.e. semi-honest) we mean that one will honestly run a protocol by following the specification of a protocol but curiously collecting some information (in which it is interested) during the protocol execution.

Definition 2. Our system achieves chosen plaintext (CPA) security if the advantage $Adv_{\mathcal{A}}^{CPA}$ is negligible for any PPT adversary $\mathcal{A}$ in the following experiment.

$$|Pr[b = b' : (mpk, msk) \leftarrow Setup(1^k); (DB_0, DB_1, ID^*, state) \leftarrow \mathcal{A}^{\mathcal{O}}(mpk); b \in_R \{0,1\}; EDB^* \leftarrow Enc(pk_{ID^*}, DB_b^*); b' \leftarrow \mathcal{A}^{\mathcal{O}}(EDB^*, state)] - \frac{1}{2}|,$$

where $state$ is the state information, DB_0, DB_1 are two equal-size databases with the form $(m_i, W_i)_{i=1}^d$, $W^* = \sum W_i$ is the challenge keyword set, ID^* is the challenge identity, $\mathcal{O} = \{\mathcal{O}_{pk}, \mathcal{O}_{sk}, \mathcal{O}_{rk}, \mathcal{O}_{uptk}, \mathcal{O}_{TK}\}$. By querying the public key oracle $\mathcal{O}_{pk}$, $\mathcal{A}$ is given the corresponding public key of the system user (it issues). For the secret key oracle $\mathcal{O}_{sk}$, intaking ID, the oracle outputs sk_{ID} for $\mathcal{A}$, where $ID \neq ID^*$ indicating the challenge identity cannot be corrupted by $\mathcal{A}$. For the re-encryption key oracle $\mathcal{O}_{rk}$, intaking a tuple (ID_i, ID_j, w_i), the oracle outputs $rk_{ID_i \rightarrow ID_j|w_i}$. If ID_i (resp. ID_j) is in an honest re-encryption path including ID^* (note ID_i may be equal to ID^*). and meanwhile, ID_j (resp. ID_i) is in a corrupted re-encryption path, $\mathcal{O}_{rk}$ outputs $\perp$. By a re-encryption path (of a given ciphertext) we mean a path that is used to record the re-encryption history of the ciphertext among different system users (in which the nodes of the path stands for users), for example, a re-encryption path (of a ciphertext) between user A and user C could be A - B - C. If one of the users is corrupted, then the

path is defined as a corrupted one; otherwise, it is an honest path. For the keyword update token oracle $\mathcal{O}_{uptk}$, intaking a tuple (ID, w_i, w_j), the oracle outputs a token $uptk_{w_i \to w_j}$ for keyword description update. For the search token oracle $\mathcal{O}_{TK}$, intaking a tuple (ID, w), the oracle outputs a search token TK. We here do not offer re-encryption, update and search oracles to $\mathcal{A}$. $\mathcal{A}$, however, can run the corresponding re-encryption, update and search algorithms with the re-encryption keys, keyword update tokens and search tokens given by the above defined oracles. We further note that the given two databases are shared with the same keyword set W^*. It won't affect the security level of the game, since we focus on the database secrecy other than that of keyword.

Definition 3. Our system achieves keyword privacy if the advantage $Adv_{\mathcal{A}}^{KP}$ is negligible for any PPT adversary $\mathcal{A}$ in the following experiment.

$$|Pr[b = b' : (mpk, msk) \leftarrow Setup(1^k); (DB, W_0^*, W_1^*, ID^*, state) \leftarrow \mathcal{A}^{\mathcal{O}}(mpk); b \in_R \{0,1\}; EDB^* \leftarrow Enc(pk_{ID^*}, W_b^*, DB); b' \leftarrow \mathcal{A}^{\mathcal{O}}(EDB^*, state)] - \frac{1}{2}|,$$

where $state$ is the state information, m is the challenge message, W_0^*, W_1^* are two challenge distinct keyword sets (for clarity, we highlight them as an "individual" input for Enc), ID^* is the challenge identity, and $\mathcal{O} = \{\mathcal{O}_{pk}, \mathcal{O}_{sk}, \mathcal{O}_{rk}, \mathcal{O}_{uptk}, \mathcal{O}_{TK}\}$. The oracle $\mathcal{O}_{pk}$ returns public keys for $\mathcal{A}$. For the secret key oracle $\mathcal{O}_{sk}$, intaking ID, the oracle outputs sk_{ID}, where $ID \neq ID^*$. For the re-encryption key oracle $\mathcal{O}_{rk}$, intaking a tuple (ID_i, ID_j, w_i), the oracle outputs $rk_{ID_i \to ID_j|w_i}$. If ID_i (resp. ID_j) is in an honest re-encryption path including ID^* and meanwhile, ID_j (resp. ID_i) is in a corrupted re-encryption path, $\mathcal{O}_{rk}$ outputs $\perp$. For the keyword update token oracle $\mathcal{O}_{uptk}$, intaking a tuple (ID, w_i, w_j), the oracle outputs a token $uptk_{w_i \to w_j}$ for keyword description update. For the search token oracle $\mathcal{O}_{TK}$, intaking a tuple (ID, w), the oracle outputs a search token TK. If $ID = ID^*$ and w is in a keyword update path including at least one of the challenge keywords (note w may be equal to one of the challenge keywords), $\mathcal{O}_{TK}$ outputs $\perp$. If $ID \neq ID^*$ is in a re-encryption path including ID^* and meanwhile, w is in a keyword update path including at least one of the challenge keywords, $\mathcal{O}_{TK}$ outputs $\perp$ as well. By a keyword update path we mean a path records all the keywords (of a given ciphertext) which have been updated so far, e.g., w_1 - w_2 - w_3.

Definition 4. Our system achieves *weak* search token privacy if the advantage $Adv_{\mathcal{A}}^{STP}$ is negligible for any PPT adversary $\mathcal{A}$ in the following experiment.

$$|Pr[b = b' : (mpk, msk) \leftarrow Setup(1^k); (w_0^*, w_1^*, ID^*) \leftarrow \mathcal{A}(mpk); b \in_R \{0,1\}; TK_0 \leftarrow TKGen(sk_{ID^*}, w_R^*), TK_1 \leftarrow TKGen(sk_{ID^*}, w_b^*); b' \leftarrow \mathcal{A}^{\mathcal{O}}(TK_0, TK_1)] - \frac{1}{2}|,$$

where w_0^*, w_1^* are two challenge distinct keyword set, each of the set includes a pair of distinct keywords, w_R^* is a random keyword set with two distinct keywords in the keyword space, ID^* is the challenge identity, and $\mathcal{O} = \{\mathcal{O}_{TK}\}$. For the search token oracle $\mathcal{O}_{TK}$, intaking a tuple (ID, w), the oracle outputs a search token TK. If $ID = ID^*$, $w = w_b^*$, $\mathcal{O}_{TK}$ outputs $\perp$.

3 System Construction

3.1 The Intuition

Let a bilinear map tuple be (q, g, $\hat{g}$, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e), where $\mathbb{G}_1$, $\mathbb{G}_2$ ($\mathbb{G}_1$ and $\mathbb{G}_2$ are not the same group) and $\mathbb{G}_T$ are multiplicative cyclic groups of prime order q, $|q| = k$, and g is a random generator of $\mathbb{G}_1$, $\hat{g}$ is a random generator of $\mathbb{G}_2$. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ has three properties: (1) *Bilinearity*: for all $a, b \in_R \mathbb{Z}_q^*$, $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$; (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.

A concrete encrypted cloud-based data share and search construction is proposed in [19]. The construction supports single keyword search, conjunctive keyword and even formula queries. For example, a system user can launch a query $(w_1 \wedge w_2 \wedge w_3) \vee w_4$ to a "curious-but-honest" cloud server. The query indicates that the server needs to return all the ciphertexts containing the keywords w_1, w_2 and w_3 at the same time, and those with the keyword w_4. The direct way for the server to fulfill the search is to first obtain the set A_1 of the ciphertexts $\mathbb{C}_{w_1}$ embedded with w_1, the set A_2 of $\mathbb{C}_{w_2}$ with w_2, the set A_3 of $\mathbb{C}_{w_3}$ with w_3, and the set A_4 of $\mathbb{C}_{w_4}$ with w_4, and finally return the intersection of the three sets, $A_1 \wedge A_2 \wedge A_3$, and the set A_4 to the user.

However, the search efficiency is extremely low. This is because the server has to exhaustedly search the whole encrypted database for each keyword. Namely, the complexity of each keyword search is linear to the size of the database. Accordingly, the complexity of the formula query search is at least $O(mn)$, where m is the size of the formula, and n is the size of the database (i.e. how many encrypted data are stored in the database).

To tackle the efficiency problem, we choose to leverage the latest technique introduced in [7], which we named it as "oblivious cross search". Below, we show that how to apply oblivious cross search into our construction to relieve the complexity to $O(f(w)m)$, where $f(w)$ is the complexity of locating the "least frequent keyword" w in the database. Note that we refer readers to the Section 3.1.1 in [7] about the way of choosing least frequent keyword in a database. The premise of the design is to build up a second level of search index for the cross searching. Meanwhile, we also need a new and black-box building block, called *T-Set* protocol [7]. The primitive includes three main algorithms, namely $(TSet, K_T) \leftarrow TSetSetup(T)$, $stag \leftarrow TSetGetTag(K_T, w)$ and $T[w] \leftarrow TSetRetrieve(TSet, stag)$, where T is an array of lists of equal-length bit strings indexed by the elements of keywords, so that for any function $n(k)$ of the security parameter k, for each keyword w $T[w]$ is a list $t = (s_1, ..., s_{T_w})$ of strings (note $T_w = |T[w]|$, $n(k) = |s_i|$ and $i \in [1, T_w]$). The function of the primitive are the followings: (1) Put the tuples of w and the corresponding related file identity into $T[w]$; all $T[w]$ form a T; (2) Intake T, the algorithm $TSetSetup$ outputs an encrypted "scrambled" set $TSet$ and a secret information K_T; (3) Intake a keyword w and K_T, the algorithm $TSetGetTag$ outputs a trapdoor $stag$ under w; the $stag$ is used to locate the corresponding $T[w]$ later; (4) Intake $stag$ and $TSet$, the algorithm $TSetRetrieve$ identifies the location and next returns $T[w]$. We will introduce the usage of $TSet$ later. Both $T[w]$ and $TSet$ are designed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

for a level of search index and further will be efficiently used in cross searching. We state that the above techniques allow server to fasten the search complexity (i.e. shortening search response time/client online waiting time) but they also require data owner and cloud server to increase some level of computation and storage. We further note that this "increased burden" for data owner and server are acceptable in practice (please see Section 4 for efficiency analysis).

In addition to the above techniques, we make use of IBE and PRE technologies. We use IBE to embed the keyword for ciphertext which is a typical public key searchable encryption mode; while the PRE is used for encrypted data sharing for the case where the encrypted files can be further shared to others in cloud server.

We assume the notation m to be a file identity in $\{0,1\}^k$, DB to be a dataset filling with files, $ID \in \mathbb{Z}_q^*$ is the identity of system user, $w \in \mathbb{Z}_q^*$. In practice, we may put $w \in \{0,1\}^*$ in a Target Collision Resistant (TCR) hash function [9] to yield an element in $\mathbb{Z}_q^*$ before using it. Notations used in our construction are summarized in Table 2.

TABLE 2: Frequently Used Notations

k	security parameter
ID	user identity
msk	master secret key
mpk	master public key
PKG	private key generator
sk_{ID}/pk_{ID}	the user ID's secret/public key
DB/EDB	database/encrypted database
$DB(w)$	a set of files' identities where the files are with w
d	the file number of DB, i.e. $\|DB\| = d$
m	file identity
W/w	keyword set/keyword
$List_{sk}$	a list for storing users' secret keys
$List_{up}$	a list for storing keyword update details
$List_{rk}$	a list for storing re-encryption details
T	an array for storing tuples of w and m
$T[w]$	a list for storing all m embedded with w
$XSet$	a set for storing $xtag$
$TSet$	a scramble and secure set for storing all $T[w]$
$XMat$	a set for storing (w, m, c_1, c_2)
L	a list for storing (e, y)
c_1	a counter:the repetition of w
c_2	a counter:the update number of m for its keyword field
$uptk_{w_i \to w_j}$	a keyword update token, from w_i to w_j
δ_i	a fresh random factor for w_i
$\sigma_{i \to j}$	a fresh random factor for re-encryption from ID_i to ID_j
TK	a query/search token
Pos	a list for storing position value

3.2 The Construction

Below we show how to design an efficient search construction by using oblivious cross search technique.

Setup. A fully trusted authority will initialize the system, and set up msk and mpk below.
(1). Choose an asymmetric pairing group $(q, g, \hat{g}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$.
(2). Choose $\theta, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \beta_1, \beta_2 \in_R \mathbb{Z}_q^*$, and set $h_1 = g^{\beta_1}$, $h_2 = g^{\beta_2}$, $g_1 = g^{\alpha_1}$, $g_2 = g^{\alpha_2}$, $g_3 = g^{\alpha_3}$, $g_4 = g^{\alpha_4}$, $g_5 = g^{\alpha_5}$, $K = g^{\theta}$, $\hat{h}_1 = \hat{g}^{\beta_1}$,$\hat{h}_2 = \hat{g}^{\beta_2}$, $\hat{g}_1 = \hat{g}^{\alpha_1}$, $\hat{g}_2 = \hat{g}^{\alpha_2}$, $\hat{g}_3 = \hat{g}^{\alpha_3}$, $\hat{g}_4 = \hat{g}^{\alpha_4}$, $\hat{g}_5 = \hat{g}^{\alpha_5}$, and $\hat{K} = \hat{g}^{\theta}$.
(3). Choose TCR hash functions $H_1 : \mathbb{G}_T \to \mathbb{G}_1$, $H_2 : \mathbb{G}_T \to \mathbb{Z}_q^*$, and $H_3 : \{0,1\}^k \to \mathbb{Z}_q^*$.
(4). Output $msk = (\hat{g}_2^{\alpha_1}, \alpha_5, \hat{h}_1, \hat{K})$, and $mpk = (q, h_1, h_2, \hat{h}_2, g, \hat{g}, g_1, \hat{g}_1, g_2, \hat{g}_2, g_3, \hat{g}_3, g_4, \hat{g}_4, g_5, K, F, H_1, H_2)$.

KeyGen. This algorithm is run by a trusted PKG.
(1). Choose an $r \in_R \mathbb{Z}_q^*$, and set $(\hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r, \hat{g}^r)$ for the user ID.
(2). Set $g_6 = g^{\alpha_6}$, $g_7 = g^{\alpha_7}$, $g_8 = g^{\alpha_8}$, $g_9 = g^{\alpha_9}$, $\hat{g}_6 = \hat{g}^{\alpha_6}$, $\hat{g}_7 = \hat{g}^{\alpha_7}$, $\hat{g}_8 = \hat{g}^{\alpha_8}$, $\hat{g}_9 = \hat{g}^{\alpha_9}$, where $\alpha_6, \alpha_7, \alpha_8, \alpha_9 \in_R \mathbb{Z}_q^*$.
(3). Make use of a list $List_{sk}$ to store the tuple $(ID, \hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r, \hat{g}^r, r, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$.
(4). Output sk_{ID} as $(\hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r, \hat{g}^r, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$, and pk_{ID} as $(g_6, g_7, \hat{g}_7, g_8, \hat{g}_8, g_9)$.

Enc. The algorithm is run by a data encryptor to encrypt m with a description w for a user ID. (1). Parse a DB to be $(m_i, W_i)_{i=1}^d$, where W_i is the keyword set. Initialize an empty array T (indexed by keywords from W) and an empty set $XSet$.
(2). For each $w \in W$, build the tuple list $T[w]$ and $XSet$ as
• Initialize an empty list L.
• For all $m \in DB(w)$, initialize a counter $c_1 = 0$, set $C_1 = m \cdot e(g_2, \hat{g}_1)^t$, $C_2 = g^t$, $C_3 = (h_1^{ID}g_3)^t$, $C_4 = (g_5 g_4^{-H_2(m)})^t$, $C_5 = e(g_4, \hat{g}_4)^t$, $C_6 = H_1(e(h_2, \hat{g}_4)^t)$, $C_7 = K^t$, $y = (g_7^{H_3(w||c_1)}g_6)^t$, $xtag = (g_8^{H_3(w)}g_9)^t$, where $t \in_R \mathbb{Z}_q^*$.
• Set $e = (C_1, C_2, C_3, C_4, C_5, C_6, C_7)$, and append (e, y) to L, add $xtag$ to $XSet$. The structure of $XSet$ is shown in Fig. 1. We assume that $XSet$ is formed as an $n \times t$ matrix, and meanwhile, the user will locally save an $n \times t$ matrix $XMat$, where the position (i, j) of $XMat$ stores a tuple $(w_i \to m_j, c_1)$, the position (i, j) of $XSet$ stores a $xtag$ indicating a relationship (w_i, m_j), $|W| = n$ and $t \leq kd$. For instance, (i, j) in $XMat$ could be $(1, 0)$, meaning w_i is tagged with m_j with no repetition. We further assume that $XMat$ will be automatically synchronized by user if there is any update, and moreover, (i, j) in $XSet$ is "linked" to the (i, j) in $XMat$.
• Put L into $T[w]$. Note that we show the structure of T in Fig. 1.
(3). Run $(TSet, K_T) \leftarrow TSetSetup(T)$, where $TSet$ is a "scrambled and encrypted" version of T, and ID is implicitly included in $TSet$ so that the server knows the $TSet$ belongs to the user ID. Output K_T and $EDB = (TSet, XSet)$. We note that the EDB receiver needs to know K_T and $XMat$. The data encryptor may choose a key k_0 to mask K_T and $XMat$, encrypt k_0 for the receiver, and further publish all the encryption on a bulletin board, so that the receiver can download and save the necessary data locally.

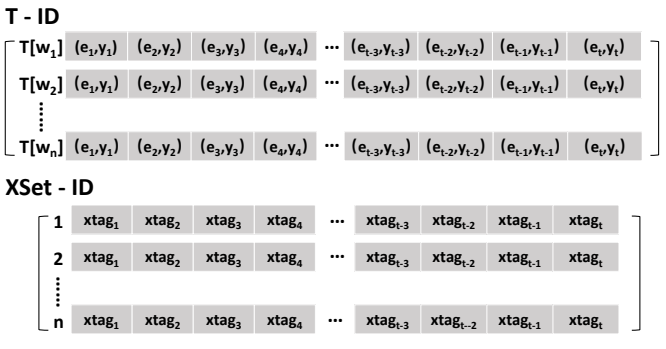


Fig. 1: The Structures Used To Store T and $XSet$ Belonging To User ID

UpTKGen. To update the description from w_i to w_j, the user ID uses a list $List_{up}$ to store tuples $(z, * \to ID,$

$w_i \rightarrow w_{i+1}$, $\delta_i \rightarrow \delta_{i+1}$, $c_i \rightarrow c_{i+1}$, $\delta_i^{c_i} \rightarrow \delta_{i+1}^{c_{i+1}}$, σ), where $z \in [1, |List_{up}|]$, $*$ is wildcard, $\delta_i \in_R \mathbb{Z}_q^*$ can randomize the i-th keyword update token (chosen by the user), the random seed $\delta_i^{c_i}$ is used to randomize the counter c_i (of the keyword w_i), and σ will be introduced later. Each user maintains his own $List_{up}$ in the system. The $List_{up}$ records a keyword update path; each keyword description w_i is tagged with "fresh" random factors δ_i and δ_i^c; a given tuple in the list indicates a re-encryption path from an identity $*$ to ID - if $\perp \rightarrow ID$, ID does not have any delegator and σ is equal to 1. Accordingly, if a ciphertext for ID is tagged with a keyword that has not been updated yet by a keyword update token, the tuple stored in $List_{up}$ is (z, $* \rightarrow ID$, $w_i \rightarrow \perp$ or $* \rightarrow w_i$, $\delta_i \rightarrow \perp$, $\delta_i^c \rightarrow \perp$, $*$), in which we say w_i (with its random factor δ_i) is the starting node of the current keyword update path (held by ID), and we hereafter may use δ^* and δ^{c*} to specify such δ_i and δ_i^c. To generate the token $uptk_{w_i \rightarrow w_j}$, the user first recovers δ_i and δ_i^c(corresponding to the current keyword description w_i) from $List_{up}$ and next chooses a new $\delta_j, \delta_j^c \in_R \mathbb{Z}_q^*$ for w_j. The user ID works as follows.

(1). Search $List_{up}$ to recover the starting random factor δ^* of the keyword update path including w_i, the current keyword w_i and its random factors δ_i, δ_i^c, and recovers $\sigma_{x \rightarrow y}$ from the re-encryption relationship remark $ID_x \rightarrow ID_y = ID$ (suppose ID has a delegator only; the multi-delegator case has been discussed in [19]). Note that if there is no keyword update (via ID's keyword update token) yet, $\delta_i = \delta^*$, $\delta_i^c = \delta^{c*}$; if $\perp \rightarrow ID$, the value of σ is equal to 1. Run $locToken = (\omega_{p0}, \omega_{p1}, \omega_{p2}) \leftarrow LotAlg.locTokenGen(TSet, w_i, m, K_T, sk_{ID})$.

(2). Set $\omega_{r1} = (\alpha_7 H_3(w_j||c_1') + \alpha_6)\delta_j \delta_j^{c_1'} / (\alpha_7 H_3(w_i||c_1) + \alpha_6)\delta_i \delta_i^{c_1}$, $\omega_{r2} = (\alpha_8 H_3(w_j) + \alpha_9)\delta_j / (\alpha_8 H_3(w_i) + \alpha_9)\delta_i$, where c_1 and c_1' can be retrieved from $XMat$.

(3). Run $stag \leftarrow TSetGetTag(K_T, w_j)$ to achieve a $stag$ which will be used by the server to locate the $T[w_j]$ from $TSet$.

(4). Retrieve the row and column numbers i and j from $XMat$ by using $(w_j \rightarrow m, c_1)$ and $(w_i \rightarrow m, c_1)$. Put $(i,j)^{(w_i)}$ and $(i,j)^{(w_j)}$ to a list Pos. Note the user will update $XMat$ locally, for example, by setting the old (i,j) tuple to be $(0,0)$ and the new tuple to be $(1,0)$.

(5). Finally, output $uptk_{w_i \rightarrow w_j} = (Pos, stag, \omega_{r1}, \omega_{r2}, locToken)$.

ReKeyGen. When a user ID_i decides to share encrypted data under keyword w_i with another ID_j, a re-encryption key is generated and delivered to the server as follows.

(1).The PKG sets $rk_1 = (\hat{h}_1^{ID_i} \hat{g}_3)^{r_i} (\hat{h}_1^{ID_j} \hat{g}_3)^{-r_j} \hat{K}^{\xi}$, $rk_2 = \hat{g}^{r_i - r_j}$ and $rk_3 = \hat{g}^{\xi}$, where $\xi \in_R \mathbb{Z}_q^*$, r_i and r_j are stored in the $List_{sk}$ corresponding to ID_i and ID_j, respectively.

(2). By sharing the data with ID_j, ID_i delegates the keyword update and search abilities to ID_j. To get rid of the re-encryption key construction cost, ID_i can choose to share $List_{up}$ with the PKG. The PKG constructs a list $List_{rk}$ to store tuples (z, $ID_i \rightarrow ID_j$, w_i, $\sigma_{i \rightarrow j}$), where z is the index for a tuple, $\sigma_{i \rightarrow j} \in_R \mathbb{Z}_q^*$ is chosen by the PKG and will be set to 1 for the case where the user has no delegator. The PKG here maintains re-encryption path in $List_{rk}$.

(3). The PKG verifies if ID_i has a single delegator, say ID_o, in all the re-encryption paths. Note we have considered the case where ID_i has multiple delegators in [19]. If yes, set $rk_4 = (\sigma_{i \rightarrow j}/\sigma_{o \rightarrow i})(\alpha_7^{(ID_j)} H_3(w_i||c_1) + \alpha_6^{(ID_j)}) / ((\alpha_7^{(ID_i)} H_3(w_i||c_1) + \alpha_6^{(ID_i)})(\delta_i \delta_i^{c_1} / \delta^* \delta^{c*}))$, and $rk_5 = (\sigma_{i \rightarrow j}/\sigma_{o \rightarrow i})(\alpha_8^{(ID_j)} H_3(w_i) + \alpha_9^{(ID_j)}) / ((\alpha_8^{(ID_i)} H_3(w_i) + \alpha_9^{(ID_i)})(\delta_i / \delta^*))$; if no, construct rk_4, rk_5 as above except for setting $\sigma_{o \rightarrow i} = 1$, where δ_i is related to the current keyword w_i embedded in the ciphertext (of ID_i), and δ^* can be traced back in $List_{up}$ with knowledge of ID_i and w_i. The PKG then may encrypt $\sigma_{i \rightarrow j}$ for the corresponding delegatee ID_j via a simple IBE encryption[1] and next to publish the encryption to a bulletin board, so that the delegatee can download the ciphertext, recover the $\sigma_{i \rightarrow j}$ and store it into $List_{up}$. Note the encryption does not need to be a part of re-encryption key.

(4). The user ID_i runs $stag_{w_i} \leftarrow TSetGetTag(K_T^{(ID_i)}, w_i)$. It further locates (i, j) from $XMat$ by w_i, m, c_1, and puts (i, j) to Pos.

(5). The user ID_j runs $\hat{stag} \leftarrow TSetGetTag(K_T^{(ID_j)}, w_i)$ to achieve a $rk_6 = \hat{stag}$ which will be used by the server to locate the $T[w_i]$ from $TSet$ belonging to ID_j.

(6). Finally, the re-encryption key $rk_{ID_i \rightarrow ID_j|w_i}$ is set to be $(Pos, rk_1, rk_2, rk_3, rk_4, rk_5, rk_6, stag_{w_i})$.

TKGen. To generate a search token TK for query $\bar{w} = (w_1, ..., w_n)$, the user ID works as follows.

(1). Run $stag \leftarrow TSetGetTag(K_T, w_1)$. Recall that w_1 is the least frequent keyword.

(2). For $c_1 = 1, 2, ...$ till server sends "stop",

- For each $i = 2, ..., n$, set $xtoken[c_1, i] = ((\hat{g}^{(\alpha_8 H_3(w_i) + \alpha_9)/((\alpha_7 H_3(w_1||c_1) + \alpha_6)(\delta_i^{c_1}/\delta^{c*}))})^r, \hat{g}^r)$, where r is a fresh random seed in $\mathbb{Z}_q^*$, δ^{c*} and $\delta_i^{c_1}$ are the current and the original random seeds for the counter which can be retrieved from $List_{up}$.
- Set $xtoken[c_1] = (xtoken[c_1, 2], ..., xtoken[c_1, n])$.

(3). Put all $xtoken$ and $stag$ to TK and send TK to the server.

Update. The algorithm fulfills the "update" function, namely updating the keyword description of encrypted files and sharing encrypted files among system users. The update functionality will not expand the size of encrypted files. No matter how many times an encrypted file is updated, its size remains constant.

(1). Update keyword description (see Fig. 2) - the server runs the algorithm $Update(uptk_{w_i \rightarrow w_j}, EDB)$ as

- To locate the items, run $(e_i, y_i) \leftarrow LotAlg.locateItem(locToken, TSet)$.
- For each tuple (e_i, y_i), set $y_i = y_i^{\omega_{r1}}$, and next updates the resulting y_i into (e_i, y_i).
- Run $TSet \leftarrow UpTSet(TSet, stag, UpE)$, where $UpE \leftarrow (e_i, y_i)$. Note that we will introduce the algorithm $UpTSet$ and the update of old tuples (e_i, y_i) in $TSet$ later.
- Retrieve $(i,j)^{w_i}$ from Pos, locates all $xtag$ in $XSet$, and updates $xtag = xtag^{\omega_{r2}}$. It further sets the $xtag$s of the positions $(i,j)^{w_i}$ to "null" and puts the new $xtag$s to the positions $(i,j)^{w_j}$.

(2). Share encrypted data (see Fig. 3) - the server runs the algorithm $ReEnc(rk_{ID_i \rightarrow ID_j|w}, EDB)$ as:

- To locate the items, the server runs $(e_i, y_i) \leftarrow T[w] = L \leftarrow TSetRetrieve(TSet^{ID_i}, stag_{w_i})$.

1. The encryption may be as $C_0 = \sigma_{i \rightarrow j} \cdot H_0(Y)$, $C_1 = Y \cdot e(g_2, \hat{g}_1)^t$, $C_2 = g^t$, $C_3 = (h_1^{ID_j} g_3)^t$, where $Y \in \mathbb{G}_T$, $t \in_R \mathbb{Z}_q^*$, $H_0 : \mathbb{G}_T \rightarrow \mathbb{Z}_q^*$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

• For each (e_i, y_i), the server sets

$$C_1 = C_1 \cdot e(C_2, rk_1)^{-1} \cdot e(C_3, rk_2) \cdot e(C_7, rk_3)$$
$$= \frac{m \cdot e(g_2, \hat{g}_1)^t \cdot e((h_1^{ID_i} g_3)^t, \hat{g}^{r_i - r_j}) \cdot e(K^t, \hat{g}^{\xi})}{e(g^t, (\hat{h}_1^{ID_i} \hat{g}_3)^{r_i} (\hat{h}_1^{ID_j} \hat{g}_3)^{-r_j} \hat{K}^{\xi})}$$
$$= \frac{m \cdot e(g_2, \hat{g}_1)^t \cdot e(g^t, (\hat{h_1}^{ID_j} \hat{g}_3)^{r_j})}{e((h_1^{ID_i} g_3)^t, \hat{g}^{r_j})}, y_i = y_i^{rk_4}.$$

• The server runs $TSet^{ID_j} \leftarrow UpTSet(TSet^{ID_j}, rk_6, UpE)$, where $UpE \leftarrow (e_i, y_i)$.
• The server retrieves (i, j) from Pos, locates all $xtag$s in $XSet^{ID_i}$, and further appends $xtag^{rk_5}$ to the (i, j) of $XSet^{ID_j}$.

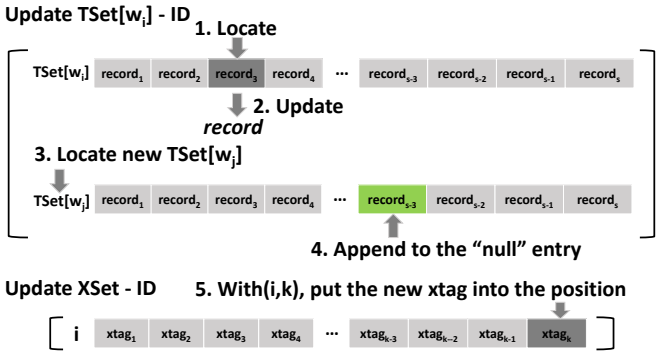


Fig. 2: Keyword Description Update for User ID Only

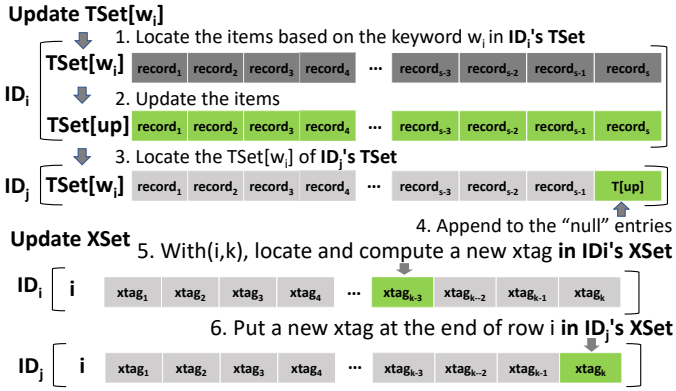


Fig. 3: Share Encrypted Data - from User ID_i to User ID_j

Search. With the search token TK, $TSet$ and $XSet$, the server runs the search process as follows.
(1). The server extracts $stag$ from TK, and set $L \leftarrow TSetRetrieve(TSet, stag)$.
(2). For $c = 1, ..., |L|$, the server works as follows:
• Retrieve each (e_c, y_c) from L.
• If $\forall i = 2, ..., n$, check if $e(y_c, xtoken[c, i]_1) = e(xtag, xtoken[c, i]_2)$. If yes, send e_c to the user.
(3). When the last tuple in L is reached, the server sends "stop" to the user and halt.
After receiving all e from the server, the user can easily run $m = C_1 \cdot e(C_3, sk_2)/e(C_2, sk_1)$ to recover all file identities, where C_3, C_2 are the components of each e.

Dec. The user ID recovers m as follows.
(1). Recover $sk_1 = \hat{g}_2^{\alpha_1} (\hat{h}_1^{ID} \hat{g}_3)^r$, $sk_2 = \hat{g}^r$ from sk_{ID}.
(2). Recover the message as $m = C_1 \cdot e(C_3, sk_2)/e(C_2, sk_1)$.

For the original ciphertext, one can compute as

$$C_1 \cdot e(C_3, sk_2)/e(C_2, sk_1)$$
$$= m \cdot e(g_2, \hat{g}_1)^t e((h_1^{ID} g_3)^t, \hat{g}^r)/e(g^t, \hat{g}_2^{\alpha_1} (\hat{h}_1^{ID} \hat{g}_3)^r)$$
$$= m \cdot e(g_2, \hat{g}_1)^t / e(g^t, \hat{g}_2^{\alpha_1}) = m.$$

For the re-encrypted ciphertext, one can decrypt as

$$C_1 \cdot e(C_3, sk_2)/e(C_2, sk_1)$$
$$= \frac{m \cdot e(g_2, \hat{g}_1)^t e(g^t, (\hat{h}_1^{ID_j} \hat{g}_3)^{r_j}) e((h_1^{ID_i} g_3)^t, \hat{g}^{r_j})}{e((h_1^{ID_i} g_3)^t, \hat{g}^{r_j}) e(g^t, \hat{g}_2^{\alpha_1} (\hat{h}_1^{ID_j} \hat{g}_3)^{r_j})}$$
$$= m \cdot e(g_2, \hat{g}_1)^t / e(g^t, \hat{g}_2^{\alpha_1}) = m,$$

where ID_i is the delegator of the ciphertext, and ID_j is the delegatee, i.e. the current ciphertext holder, who can decrypt the message by using sk_{ID_j}.
(3). The user can also reveal the pseudorandom keys k_1, k_2 and k_3 by $C_0 \oplus H_3(e(g_2, \hat{g}_1)^t)$. With the keys, the user obtains search ability.

3.3 The Construction of $LotAlg$

The algorithm $LotAlg$ (which is executed by both data owner and server) includes two sub-algorithms, namely $locTokenGen$ and $locateItem$. It can locate the items (which are stored in $TSet$) needed to be updated. The sub-algorithm $locTokenGen$ (run by data owner) intakes $TSet$, w, m, K_T, sk_{ID}, and outputs a locate token $locToken$ that helps server locate the items; while $locTokenGen$ (run by server) takes $locToken$, $TSet$ as input, and outputs all matching tuples (e_i, y_i). In the algorithms, ω_{p0} is used to locate the set $T[w]$, and ω_{p1}, ω_{p2} are to identify the files m_d within $T[w]$. Since α_5 is a part of msk belonging to PKG, the PKG will involve in Step 1. (2) to help generate ω_{p1}, ω_{p2}.

Algorithm 1 $LotAlg$ - Locate Item in $TSet$

1. $locToken \leftarrow locTokenGen(TSet, w, m, K_T, sk_{ID})$
(1). Run $stag_w \leftarrow TSetGetTag(K_T, w)$.
(2). Generate $(r_{tk}, (\hat{h}_2 \hat{g}_4^{-r_{tk}})^{\frac{\delta^*}{(\alpha_5 - H_2(m))\delta_i \sigma_{x \to y}}})$, where $r_{tk} \in_R \mathbb{Z}_q^*$.
(3). Set the locate token as $locToken = (\omega_{p0}, \omega_{p1}, \omega_{p2}) = (stag_w, r_{tk}, (\hat{h}_2 \hat{g}_4^{-r_{tk}})^{\frac{\delta^*}{(\alpha_5 - H_2(m))\delta_i \sigma_{x \to y}}})$.
2. $(e_i, y_i) \leftarrow locateItem(locToken, TSet)$
(1). Retrieve $T[w] = L \leftarrow TSetRetrieve(TSet, stag_w)$.
(2). For each e_i within $T[w]$ $(i \in [1, t])$, check $H_1(e(C_4, \omega_{p2}) C_5^{\omega_{p1}}) = C_6$, where C_4, C_5, C_6 are elements in each e_i.
(3). Output all the tuples (e_i, y_i) that make the above equation hold.

3.4 The Constructions of $TSet$, $XSet$ and $XMat$

We state that the construction of $TSet$ can follow the one introduced in [7]. We refer the reader to [7] for more construction details of $TSet$. The $TSet$ is instantiated as a hash table with B buckets of size S each. Here, we define the size of B to be $|W|$, i.e. total number of keywords in a DB, and the size of S to be $|T[w]|$, i.e. the size of $|DB(w)|$. Accordingly, the size of the hash table is $\sum_{w \in W} |DB(w)|$. We use $TSet[i]$ $(i \in [1, B])$ and $TSet[i, j]$ $(j \in [1, S])$ to denote $T[w_i]$ and the jth item of $T[w_i]$, respectively. The jth

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

item includes a $record$ tuple ($label$, $value$), in which $label$ is used to indicate if files tagged with the same w, and $value$ is used to store (e, y). We show the structure of $TSet$ in Fig. 4.

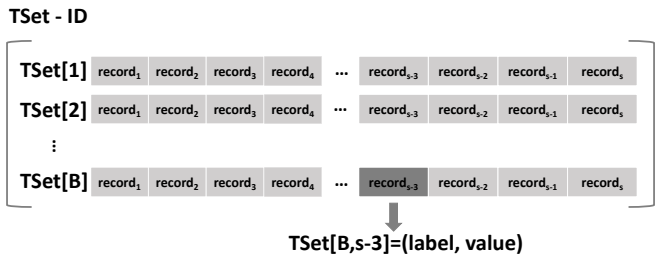


Fig. 4: The Structure of $TSet$

To adapt the design of [7] to our construction, we need to fix the size of B, $|T[w]|$ and S, and meanwhile fix the universal keyword set W *in advance*, where S has to be sufficient large. For example, we can set $B = |W|$, the number of keywords, to be 1,000,000 throughout the whole system, and meanwhile, for each keyword, we allow it to be repeatedly tagged with at most 10,000 files. If $T[w]$ is not fully taken, we then automatically set the corresponding tuple (e, y) to be "null". For instance, a tuple $T[w]$ can be $((e_1, y_1), (e_2, y_2), ..., (e_t, y_t))$, where (e_t, y_t)="null". Accordingly, we need to set $TSet[b, j].value \leftarrow$"null" in the algorithm $TSetSetup(T)$, but still setting $TSet[b, j].label \leftarrow L$.

The structure of $XSet$ (storing $xtag$s), $n \times t$ matrix (in which the rows are for keywords and the columns are for files), is reflected onto $XMat$ (storing 1/0 tuples). The position (i, j) indicates the relationship between a keyword w_i and a file m_j. Both of the structures also need to fix the size in advance, where the n and t must be identical to the size of universal keyword set and file set defined in $TSet$. We note that the above operations and set design are required to be done by data owner rather than server.

3.5 Feasible Update Function

Below we define and design the algorithm $UpTSet$. The algorithm intakes $TSet$, $stag \leftarrow F(K_T, w)$ and a set UpE, and outputs a new $TSet$, where UpE is a set for $((e_1, y_1), ..., (e_{|UpE|}, y_{|UpE|}))$. The $UpTSet$ can update the new tuples,

Algorithm 2 $UpTSet$ - Update the $TSet$

1. Set a bit $\beta = 1$, and a counter $z = 1$,
2. For each (e, y), repeat the following loop while $\beta = 1$:
(1). Set $(b, L, K) \leftarrow H(F(stag, z))$, retrieve an array $B \leftarrow TSet[b]$,
(2). Search for index $c \in \{1, ..., S\}$ s.t. $B[j].label = L$,
(3). Randomly choose a $B[j].value$ ="null", and reset $B[j].value \leftarrow (\beta|s) \oplus K$, where $s \leftarrow (e, y)$, if the item is the last one in $B[j]$, set $\beta = 0$, and $\beta = 1$ otherwise.
(4). If either the above reset cannot be done or all (e, y) are not reset yet, increment z.

say (e_{w_j}, y_{w_j}) to the new "null" positions in $TSet$. With similar technique and knowledge of $stag_{w_i}$, the server can also locate the old tuples (e_{w_i}, y_{w_i}) in $TSet$ and further set the $B.[j].value$ to "null".

3.6 Get Rid of Redundancy

The update of $TSet$ will definitely incur the redundancy of (e, y). For example, updating a tuple (e_{w_i}, y_{w_i}) to become (e_{w_j}, y_{w_j}) by using the algorithm $UpTSet$, the old tuple (e_{w_i}, y_{w_i}) is still there, stored in $TSet$. After being shared with many (e_w, y_w) from ID_i, ID_j may find out that some (e, y) may point to the same keyword and file identity. We state that these "overlap" cannot be noticed by the server, as they are in the "well-formed" encryption. The overlap phenomenon will be told while ID_j proceeds to the decryption of e. To reduce the redundancy, we design the algorithm $ReMov$ for the server. The algorithm intakes $TSet$, $locToken$, and outputs a new $TSet$, where $locToken = (stag, \omega_{p1}, \omega_{p2})$ (please refer to Algorithm 1).

Algorithm 3 $ReMov$ - Remove the redundancy of $TSet$

1. Set a bit $\beta = 1$, and two counters $z = 1$, $q = 0$,
2. Repeat the following loop while $\beta = 1$:
(1). Set $(b, L, K) \leftarrow H(F(stag, z))$ and retrieve an array $B \leftarrow TSet[b]$,
(2). Search for index $c \in \{1, ..., S\}$ s.t. $B[j].label = L$,
(3). Let $v \leftarrow B[j].value \oplus K$, where β is the first bit of v, and the rest of v is s.
(4). Extract e from s, check $H_1(e(C_4, \omega_{p2})C_5^{\omega_{p1}}) = C_6$, where C_4, C_5, C_6 are elements in e.
(5). If the above equation holds, set $q++$; otherwise, proceed.
(6). If $q > 1$, reset $B[j].value$ ="null"; otherwise, proceed. (7). Increment z.

A data owner can generate $locToken$ for the server to run $ReMov$ at any time. Recall that $locToken$ includes $stag$ and ω_{p1}, ω_{p2}. $stag$ is used to locate the $T[w]$, while ω_{p1}, ω_{p2} are for tracking a specific pair of (e, y) within $T[w]$. $locToken$ is an individual "pointer" for the file(s) embedded with a keyword w. Whilst the equation of the step 2. (4) has been repeated, it indicates the current (e, y) is a redundant tuple.

4 System Analysis

4.1 Security Analysis

We make use of the generic bilinear group model and the random oracle model to prove that no PPT adversary can break the chosen plaintext security and keyword privacy. We consider three random encodings δ_1, δ_2, δ_T of the additive group $\mathbb{F}_q$ with injective maps $\delta_1, \delta_2, \delta_T : \mathbb{F}_q \rightarrow \{0, 1\}^k$, where $k > 3log(q)$. For $i = 1, 2, T$, set $\mathbb{G}_i = \{\delta_i(x) : x \in \mathbb{F}_q\}$. The game simulator $\mathcal{B}$ is given oracles to compute the induced group action on $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and an oracle to compute a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. It is also given random oracles for representing hash functions.

Theorem 1. Let Q_1 be a bound on the total number of group elements an adversary $\mathcal{A}$ receives from queries of hash functions, groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and the bilinear map e, and from interactions with the chosen plaintext security game. We have that the advantage of $\mathcal{A}$ in winning the game is $O(Q_1^2/q)$.

Proof 1. In the normal chosen plaintext security game, a challenge ciphertext includes $C_1 = m_b \cdot e(g, \hat{g})^{\alpha_1 \alpha_2 t}$. We can revise the game so that C_1 is either $e(g, \hat{g})^{\xi}$ or $e(g, \hat{g})^{\alpha_1 \alpha_2 t}$ instead, where $\xi \in_R \mathbb{F}_q$. We state that any adversary with advantage ϵ in the normal game can be converted into an

adversary with advantage $\epsilon/2$ in the revised game. The adversary is required to distinguish $m_0 e(g,\hat{g})^{\alpha_1\alpha_2 t}$ from $e(g,\hat{g})^{\xi}$, and $e(g,\hat{g})^{\xi}$ from $m_1 e(g,\hat{g})^{\alpha_1\alpha_2 t}$. Below we let g^x, $\hat{g}^y$, and $e(g,\hat{g})^z$ denote $\delta_1(x)$, $\delta_2(y)$ and $\delta_T(z)$, respectively. Note $\mathcal{B}$ will maintain $List_{sk}$, $List_{rk}$ and $List_{up}$ as in the real scheme.

• Setup Phase. $\mathcal{B}$ chooses θ, α_1, α_2, α_3, α_4, α_5, β_1, $\beta_2 \in_R \mathbb{F}_q$, and further sets h_1, $\hat{h}_1$, h_2, $\hat{h}_2$, g_1, $\hat{g}_1$, g_2, $\hat{g}_2$, g_3, $\hat{g}_3$, g_4, $\hat{g}_4$, g_5, $\hat{g}_5$, K and $\hat{K}$ as in the real scheme. $\mathcal{B}$ sends the master public key mpk to $\mathcal{A}$.

• Random Oracle Queries. When $\mathcal{A}$ queries H_1 on a $\mathbb{G}_T$ element, $\mathcal{B}$ chooses a random $s \in \mathbb{F}_q$ and outputs g^s. Similarly, $\mathcal{B}$ responds the corresponding values (in $\mathbb{F}_q$) to other random oracles (H_2 and H_3) queries.

• Phase 1.

1) Public key and Secret Key Queries. For an identity ID, $\mathcal{B}$ chooses r, α_6, α_7, $\alpha_8, \alpha_9 \in_R \mathbb{F}_q$, and next computes $\hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r$, $\hat{g}^r$, $g_6 = g^{\alpha_6}$, $g_7 = g^{\alpha_7}$, $g_8 = g^{\alpha_8}$, $g_9 = g^{\alpha_9}$, $\hat{g}_6 = \hat{g}^{\alpha_6}$, $\hat{g}_7 = \hat{g}^{\alpha_7}$, $\hat{g}_8 = \hat{g}^{\alpha_8}$, $\hat{g}_9 = \hat{g}^{\alpha_9}$. $\mathcal{B}$ sends the secret key tuple $(\hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r, \hat{g}^r, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$ and the corresponding public key to $\mathcal{A}$. Finally, $\mathcal{B}$ stores $(ID, \hat{g}_2^{\alpha_1}(\hat{h}_1^{ID}\hat{g}_3)^r, \hat{g}^r, r, \alpha_6, \alpha_7, \alpha_8, \alpha_9)$ into $List_{sk}$.
2) Re-Encryption Key Queries. $\mathcal{A}$ issues the tuple (ID_i, ID_j, w_i) to $\mathcal{B}$. $\mathcal{B}$ recovers the tuples $(ID_i, \hat{g}_2^{\alpha_1}(\hat{h}_1^{ID_i}\hat{g}_3)^{r_{ID_i}}, \hat{g}^{r_{ID_i}}, r_{ID_i}, \alpha_6^{ID_i}, \alpha_7^{ID_i}, \alpha_8^{ID_i}, \alpha_9^{ID_i})$ and $(ID_j, \hat{g}_2^{\alpha_1}(\hat{h}_1^{ID_j}\hat{g}_3)^{r_{ID_j}}, \hat{g}^{r_{ID_j}}, r_{ID_j}, \alpha_6^{ID_j}, \alpha_7^{ID_j}, \alpha_8^{ID_j}, \alpha_9^{ID_j})$ from $List_{sk}$, and next computes $rk_2 = \hat{g}^{r_{ID_i} - r_{ID_j}}$, $rk_1 = (\hat{h}_1^{ID_i}\hat{g}_3)^{r_{ID_i}}(\hat{h}_1^{ID_j}\hat{g}_3)^{-r_{ID_j}}\hat{K}^{\nu}$, $rk_3 = \hat{g}^{\nu}$, $rk_4 = (\sigma_{i\to j}/\sigma_{o\to i})\frac{(\alpha_7^{(ID_j)}H_3(w_i^{(ID_i)}||c_1)+\alpha_6^{(ID_j)})}{((\alpha_7^{(ID_i)}H_3(w_i^{(ID_i)}||c_1)+\alpha_6^{(ID_i)})(\delta_i\delta_i^{c_1}/\delta_i^*\delta_i^{c*}))}$, $rk_5 = (\sigma_{i\to j}/\sigma_{o\to i})\frac{(\alpha_8^{(ID_j)}H_3(w_i)+\alpha_9^{(ID_j)})}{((\alpha_8^{(ID_i)}H_3(w_i)+\alpha_9^{(ID_i)})(\delta_i/\delta^*))}$ as in the real scheme, where $\nu \in_R \mathbb{F}_q$ and the values of σ and δ are from $List_{rk}$ and $List_{up}$. $\mathcal{B}$ further generates $Pos, rk_6, locToken, stag_{w_i}$ as in the real game. Finally, $\mathcal{B}$ sends the re-encryption key to $\mathcal{A}$. Besides, $\mathcal{B}$ will publish an encryption of $\sigma_{i\to j}$, and add $\sigma_{i\to j}$ to the corresponding tuple stored in $List_{up}$.
3) Keyword Update Token Queries. $\mathcal{A}$ issues a tuple (ID, w_i, w_j) to $\mathcal{B}$. $\mathcal{B}$ generates Pos, $stag$, $locToken$, $\omega_{r1} = (\alpha_7 H_3(w_j||c_1') + \alpha_6)\delta_j\delta_j^{c_1'}/(\alpha_7 H_3(w_i||c_1) + \alpha_6)\delta_i\delta_i^{c_1}$, $\omega_{r2} = (\alpha_8 H_3(w_j) + \alpha_9)\delta_j/(\alpha_8 H_3(w_i) + \alpha_9)\delta_i$ as in the real scheme, and returns $uptk_{w_i\to w_j}$ to $\mathcal{A}$.
4) Search Token Queries. $\mathcal{B}$ computes the search token $xtoken[c_1] = (xtoken[c_1,2], ..., xtoken[c_1,n])$ as in the real scheme, in which $xtoken[c_1,i] = ((\hat{g}^{(\alpha_8 H_3(w_i)+\alpha_9)/((\alpha_7 H_3(w_1||c_1)+\alpha_6)(\delta_i^{c_1}/\delta^{c*}))})^r, \hat{g}^r)$ and $i \in [2,n]$.

• Challenge Phase. $\mathcal{A}$ commits to m_0, m_1, w^* and ID^*. $\mathcal{B}$ chooses $\xi, t \in_R \mathbb{F}_q$, and computes the ciphertext as $C_1 = e(g,\hat{g})^{\xi}$, $C_2 = g^t$, $C_3 = (h_1^{ID^*}g_3)^t$, $C_4 = (g_5 g_4^{-H_2(m_b)})^t$, $C_5 = e(g_4,\hat{g}_4)^t$, $C_6 = H_1(e(h_2,\hat{g}_4)^t)$, $C_7 = K^t$, $y = (g_7^{*H_3(w^*||c_1)}g_6^*)^t$, $xtag = (g_8^{*H_3(w^*)}g_9^*)^t$, where $g_6^*, g_7^*, g_8^*, g_9^*$ are the public key elements of ID^* generated by $\mathcal{B}$ as in the real scheme.

• Phase 2. Same as Phase 1 but with the restrictions.

• Guess. $\mathcal{A}$ outputs a guess bit b'.

We assume $\mathcal{A}$ can query the group oracles by using its responses from the simulations and some intermediate values obtains from the oracles; there are q distinct values in the ranges of $\delta_{1,2,T}$ with probability $1 - O(1/q)$. We seen an oracle query as a rational function $f = x/y$ in the variables ξ, β_z, α_l, θ, t, ν, the random factors δ, σ and r, where $z \in \{1,2\}$ and $l \in [1,9]$. We here consider a collision event where two queries for two distinct rational functions $f = x/y$ and $f' = x'/y'$ with two sets of random choices of variables that yields the same output. For any query pair (in $\mathbb{G}_1$, $\mathbb{G}_2$, or $\mathbb{G}_T$) corresponding to two distinct f and f', the collision will happen only if the non-zero polynomial $xy' - x'y$ leads to zero, where the total degree of the equation is at most 7. By the Schwartz-Zippel lemma [27], [35], we have that the probability of the collision is at most $O(1/q)$. By a union bound, we have $O(Q_1^2/q)$. The simulations do not have collision event with probability $1 - O(Q_1^2/q)$.

We here consider the view of $\mathcal{A}$ in the case where $\xi = \alpha_1\alpha_2 t$. Since there is no any collision for queries to oracles (with overwhelming probability) and each group element (responded by $\mathcal{B}$) is uniformly chosen, the view of $\mathcal{A}$ should be identically distributed. However, one remaining possibility that $\mathcal{A}$'s view is distinct in the above case is that there are two distinct queries f and f' to $\mathbb{G}_T$ but yielding the same output. Since the ξ is an exponent of the element in $\mathbb{G}_T$, we can have some additive computation to output an exponent $\gamma\xi$ with a non-zero γ. Similarly, we have $\gamma'\alpha_1\alpha_2 t$ as well. Accordingly, we have $f - f' = \gamma\xi - \gamma'\alpha_1\alpha_2 t$, and then $f - f' - \gamma\xi = \gamma'\alpha_1\alpha_2 t$. To hold the equation, $\mathcal{A}$ has to obtain the element with exponent $\gamma'\alpha_1\alpha_2 t$ from queries to $\mathbb{G}_T$. Namely, if $\mathcal{A}$ can achieve the element, it can tell the difference to win the game.

But $\mathcal{A}$ cannot construct a query for $e(g,\hat{g})^{\gamma'\alpha_1\alpha_2 t}$ for some constant γ'. Since our system is built on top of asymmetric pairing groups, it is much easier to make observation on oracle queries. By observation, only group $\mathbb{G}_1$ provides elements with exponent t, namely, g^t, $g^{(\beta_1 ID+\alpha_3)t}$, $g^{(\alpha_5-\alpha_4 H_2(m_b))t}$, $g^{\theta t}$, $g^{(\alpha_7 H_3(w||c_1)+\alpha_6)t}$ and $g^{(\alpha_8 H_3(w)+\alpha_9)t}$. Since there are no factors $1/\theta$, $1/\beta_1$ and $1/\alpha_3$ existing in group $\mathbb{G}_2$ for the elimination of the corresponding exponents, $\mathcal{A}$ may consider the rest of the elements. Recall that $\alpha_6, \alpha_7, \alpha_8, \alpha_9 \in \mathbb{F}_q$ are designed for keyword field that is unrelated to the target component. Moreover, α_4 and α_5 have no direct computation relationship with α_1 and α_2. $\mathcal{A}$ can only focus on g^t. $\mathcal{A}$ is given an element $e(g^{\alpha_4},\hat{g}^{\alpha_4})^t$ in $\mathbb{G}_T$. However, it cannot help $\mathcal{A}$ break the game as it is only used in additive operations.

Given g^t, $\mathcal{A}$ needs the elements with exponent $k\alpha_1\alpha_2$ in group $\mathbb{G}_2$. From the simulations, we can see that there is only the response of secret key query, a $\hat{g}^{\alpha_1\alpha_2+(ID\beta_1+\alpha_3)r}$, satisfying the requirement. $\mathcal{A}$ accordingly has a $K_1 = t(\alpha_1\alpha_2 + (ID_i\beta_1+\alpha_3)r_{ID_i}) = \alpha_1\alpha_2 t + ID_i\beta_1 r_{ID_i} t + \alpha_3 r_{ID_i} t$, where i is the index for i-th query. To cancel out the part $ID_i\beta_1 r_{ID_i} t + \alpha_3 r_{ID_i} t$, $\mathcal{A}$ needs to find a g^t and the elements with exponents $\beta_1 r_{ID_i}$ and $\alpha_3 r_{ID_i}$ in $\mathbb{G}_2$. By observation, $\mathcal{A}$ has $K_2 = (\beta_1 ID_i + \alpha_3)r_{ID_i} - (\beta_1 ID_j + \alpha_3)r_{ID_j} + \theta\nu$ (via re-encryption key queries), such that it can create a query $\alpha_1\alpha_2 t + t\beta_1 ID_j r_{ID_j} + t\alpha_3 r_{ID_j} - \theta\nu t$ by subtracting K_1 with tK_2. We can see that the computation indicates that the given ciphertext under ID_i is re-encrypted to ID_j. We set $K_3 = \alpha_1\alpha_2 t + \Delta - \theta\delta t$. Since $\mathcal{A}$ is given $g^{\theta t}$ and $\hat{g}^{\nu}$, it can cancel out the last part of K_3 to have $K_4 = \alpha_1\alpha_2 t + \Delta$.

If $\mathcal{A}$ finds a way to eliminate Δ, it can recover $\alpha_1\alpha_2 t$ from

K_4. $\mathcal{A}$ only needs to require a query between $g^{(\beta_1 ID_j+\alpha_3)t}$ and a "special" $\hat{g}^{r_{ID_j}}$. But a pair of identities ID_i, ID_j in a re-encryption key cannot be corrupted in the security game if one of them has a re-encryption path with ID^*. Besdies, K_4 is computed from the elements of the challenge ciphertext. Therefore, the special element $\hat{g}^{r_{ID_j}}$, a part of the secret key sk_{ID_j}, will not be given to $\mathcal{A}$. There is no other term $\mathcal{A}$ gains access to that can cancel out the part Δ of K_4. $\mathcal{A}$ cannot construct a query for $e(g,\hat{g})^{\gamma' \alpha_1 \alpha_2 t}$ with some constant γ'.

Theorem 2. Let Q_2 be a bound on the total number of group elements an adversary $\mathcal{A}$ receives from queries of hash function, groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and the bilinear map e, and from interactions with the keyword privacy game. We have that the advantage of $\mathcal{A}$ in winning the game is $O(Q_2^2/q)$.

Proof 2. Due to the similarities in the proofs between Theorem 4.1 and Theorem 4.1, we here only present a proof stretch, and state that the proof can be easily completed by following the roadmap of the proof of Theorem 4.1. The game challenger here sets up system and responds the queries of random oracles, public and secret key, re-encryption key, keyword update token and search token as in the proof of Theorem 4.1 but being limited to the restrictions listed in the Definition 3. In the challenge phase, the challenger constructs the challenge ciphertext to $\mathcal{A}$ by intaking (m, w_0^*, w_1^*, ID^*). Following the restrictions given in the Definition 3, $\mathcal{A}$ attempts to find either $\hat{g}^{\alpha_7 H_3(w_b||c_1)+\alpha_6}$ or $\hat{g}^{\alpha_8 H_3(w_b)+\alpha_9}$ to match pairings $e(g^{(\alpha_7 H_3(w_b||c_1)+\alpha_6)t}, \hat{g})$ or $e(g^{(\alpha_8 H_3(w_b)+\alpha_9)t}, \hat{g})$. Since $\hat{g}_6$ and $\hat{g}_9$ are not given as the public key of ID^* and meanwhile, no $\hat{g}^{K\alpha_6}$ or $\hat{g}^{K\alpha_9}$ can be achieved via oracle queries (where K is a constant), the only way $\mathcal{A}$ can reach the goal is to corrupt the secret key of ID^* or to obtain some "sensitive" search tokens. But those queries are forbidden (Definition 3). $\mathcal{A}$ fails to win the game.

Theorem 3. Assume the decisional Diffie-Hellman assumption [7] holds in $\mathbb{G}_2$, all the pseudorandom functions and hash functions are secure and target collision resistant, our scheme achieves the weak search token privacy.

Proof 3. Given two distinct search tokens TK_1 and TK_2, the PPT adversary $\mathcal{A}$ is target to tell the difference. Assume $TK_1 = ((\hat{g}^{\alpha_8 H_3(w_2^{(R)})/\alpha_7 H_3(w_1^{(R)}||c_1)\beta_1})^{r_1}, \hat{g}^{r_1})$ and $TK_2 = ((\hat{g}^{\alpha_8 H_3(w_2^{(b)})/\alpha_7 H_3(w_1^{(b)}||c_1)\beta_2})^{r_2}, \hat{g}^{r_2})$, where r_1 and r_2 are distinct random factors in $\mathbb{Z}_q$, β_1 and β_2 are the respective δ computation, (R) and (b) represent the sign of random and bit b, respectively. Since our bilinear pairings are designed as $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with no easy homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$, $\mathcal{A}$ cannot use the parings computation directly. Furthermore, r_1 and r_2 are two fresh and distinct random factors, $\mathcal{A}$ cannot find the corresponding elements in $\mathbb{G}_1$ to compute correct pairings to verify the equality of TK_1 and TK_2. Similarly, $\mathcal{A}$ cannot generate correct elements in $\mathbb{G}_1$ without knowing α_6, α_7, α_8, α_9, the parts of secret key of the target ID. But if $\mathcal{A}$ can output $\hat{g}^{r_1 r_2 \Delta}$ and $\hat{g}^{r_1 r_2 \Omega}$ by using TK_1 and TK_2, it can definitely tell the difference between the search tokens. If so, it can break the DDH problem. That contradicts to our assumption. This weak

4.2 Efficiency Analysis

To present a fair computational cost analysis, we denote the exponent cost in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ as exp_1, exp_2, exp_3, respectively. For the pairing cost in $\mathbb{G}_T$, we denote it as p. We will consider the cost w.r.t. a trusted system setup party, system user, a PKG and a server. We note that the computational cost of the algorithm Enc shown in Table 3 is for the case where a file is only tagged with a single keyword. From the Table 3, it can be seen that a system user is only required to take less than 50% of the system total cost, $13exp_1+3exp_2+5p$, to achieve flexible data query, keyword update and data share, while the trusted party, PKG and server are responsible for the rest of the cost, which is more than 50% of the total cost.

In the following communication analysis, we use mpk, msk, pk, sk, CT, $uptk$, rk, TK to denote master secret key, master public key, public key, secret key, ciphertext, keyword update token and search token, respectively. We note that the communication cost of TK and CT normally have to respectively multiply factors $|SF|$ and $|\Delta|$, where $|SF|$ is the size of the search formula, $|\Delta| = \sum_{w \in W} |DB(w)|$. Below we only set $|SF|$ and $|\Delta|$ to be 1, namely, the following communication cost is for the case where a file is tagged with a single keyword.

TABLE 4: Theoretical Communication Cost

Groups	Communication Cost							
	mpk	msk	pk	sk	TK	rk	uptk	CT
$\mathbb{Z}_q$	0	1	0	4	0	2	3	0
$\mathbb{G}_1$	8	0	4	0	0	0	0	7
$\mathbb{G}_2$	6	3	2	2	2	3	1	0
$\mathbb{G}_T$	0	0	0	0	0	0	0	2
TP $\rightarrow$ PKG	$\mathbb{Z}_q + 3\mathbb{G}_2$							
PKG $\rightarrow$ User	$5\mathbb{Z}_q + 4\mathbb{G}_1 + 5\mathbb{G}_2$							
PKG $\rightarrow$ Server	$2\mathbb{Z}_q + 3\mathbb{G}_2$							
User $\rightarrow$ Server	$3\mathbb{Z}_q + 7\mathbb{G}_1 + 3\mathbb{G}_2 + 2\mathbb{G}_T$							
Total	$10\mathbb{Z}_q + 19\mathbb{G}_1 + 19\mathbb{G}_2 + 2\mathbb{G}_T$							

In Table 4, we can see that a system user only needs to spend $3\mathbb{Z}_q + 7\mathbb{G}_1 + 3\mathbb{G}_2 + 2\mathbb{G}_T$ in communication cost with a server during the execution of the system. The $\mathbb{G}_T$ part is due to the pairings (C_1, C_5) used in the ciphertext. It is not difficult to see that the cost of the user is approximately 45% of the total system cost.

4.3 Practical Simulation

We further implement our scheme using PBC library [23] which is one of the most widely used library for pairing computation. We choose the asymmetric pairing which is constructed on ordinary curves with embedding degree 6, and its orders are prime or a prime multiplied by a small constant. It is first discovered by Miyaji, Nakabayashi and Takano [25], and it is usually more efficient compared with other curves. The simulation is performed on a mac pro with 2.2GHz Intel Core i7 and 16GB 1600 MHz DDR3 memory. Similar to the theoretical analysis, we demonstrate our practical simulation results in Table 5 and Table 7.

We verify our theoretical analysis through the simulation that the workload of a system user indeed is lightened significantly. From Table 5 (computational cost), we can see that the computational time for Trust Party, Server and PKG are 0.0738, 0.0409 and 0.1510 seconds, while the user side

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3004988, IEEE Transactions on Services Computing

TABLE 3: Theoretical Computation Cost

Cost	Computation Cost									
	Setup	KeyGen	UpTKGen	ReKeyGen	TKGen	Enc	Update. (2)	Update. (1)	Search	Dec
epx_1	8	4	0	0	0	13	2	2	0	0
epx_2	8	8	2	7	3	0	0	0	0	0
epx_3	0	0	0	0	0	0	0	1	0	0
p	0	0	0	0	0	3	3	1	2	2
trusted party (for setup)	$8exp_1 + 8exp_2$									
Server	$4exp_1 + exp_3 + 6p$									
PKG	$4exp_1 + 17exp_2$									
User	$13exp_1 + 3exp_2 + 5p$									
Total	$29exp_1 + 28exp_2 + exp_3 + 11p$									

TABLE 5: Practical Computational Cost (second)

Trust Party	0.0738
Server	0.0409
PKG	0.1510
User Side	0.0682
Total	0.3267

takes only 0.0682 seconds. As a result, the user side takes only 20% of the total computational cost.

We further compare our design with the recent and lightweight PKES scheme [13]. Since [13] do not consider keyword update (which is mainly on server side) and decryption functionalities, we only compare the computational cost in terms of $KeyGen$, Enc, $TKGen$ and $Search$ to achieve comparison fairness. Based on the test bar set in [13], we can have our $KeyGen$, Enc, $TKGen$ and $Search$ require 0.02598s, 0.046789s, 0.006495s and 0.010854s, respectively. From Table 6, it can be seen that our scheme outperforms [13] in trapdoor generation and a single keyword search, while the rest of our functions require more run time than [13]. But, in general, our run time cost in the four functions is acceptable in practice.

TABLE 6: Computational Cost Comparison (second)

Function/Scheme	Ours	[13]
KeyGen	0.02598	0.008674
Enc	0.046789	0.016378
TKGen	0.006495	0.017462
Search	0.010854	0.015225

As for the communication cost, we need to find out the length of elements in groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and $\mathbb{Z}_n$. According to the implementation of the MNT curve, the group elements have $40, 120, 120$ and 20 bytes in length accordingly. As a result, from the trusted party to PKG, 380 bytes data are sent. From PKG, 860 bytes and 400 bytes data are sent to User and Server respectively. The data communication cost from user to server comes from the ciphertext and token delivery as well as the search query costing about 940 bytes. In total the practical communication cost is about 2580 bytes.

TABLE 7: Practical Communication Cost (byte)

Trusted Party→ PKG	380
PKG→ User	860
PKG→ Server	400
User → Server	940
Total	2580

We also compare our scheme with [13] w.r.t. communication cost in Table 8. But we only consider the related cost over the $Keys$, $Ciphertext$ and $Trapdoor$. We note that for $Keys$ we consider the size of both public key and secret key. As for the size of $Keys$, our scheme requires 4 more $\mathbb{G}_2$ than [13]; and our ciphertext size is larger than that of [13] due to the support of our decryption function (which needs extra $5\mathbb{G}_1 + 2\mathbb{G}_T$ as compared to [13]). But we state that the above cost is acceptable in the viewpoint of practical user since the cost is < 1 MB which is bearable for common network/communication device, e.g., smartphone.

TABLE 8: Communication Cost Comparison (byte)

Item/Scheme	Ours	[13]
Keys	720	240
Ciphertext	520	80
Trapdoor	240	120

The "least frequent keyword" technique proposed in this paper can help to improve the keyword searching speed dramatically. Now we would like to set up an experiment to simulate this event. Assume that there are in total r files, and the keyword set A belongs to m files, and keyword set B belongs to n files, where $m = 10 \times n$, $m = r \times 0.9$, and $m, n < r$, such that the number of files contain A dominants the whole set. B is defined to be the "least frequent keyword" according to the previous definition. To search the files that contain both A and B, traditionally, we would first search r files to locate files contain either A or B. And then search m or n times depending on which keyword is searched first. So the time complexity is $T_{search} \times r + T_{search} \times m$ or $T_{search} \times r + T_{search} \times n$. However, if we assume that B is the "least frequent keyword" that has been prepared, then the searching strategy becomes that we search the files contain B first, and then search the A. In this case, the time complexity becomes $T_{search} \times r + T_{search} \times n$, which is obviously better than the previous searching strategy if we are sure $n < m$. In the first case, we do not know which keyword will be searched first, thus each time the user will have 50% chance to pick A or B. In detail, the average time follows the binomial distribution with $p = 1/2$, but for the simplicity we omit the probability discussion here. We further state that the bitmap index (via BigInteger) is used to denote file identifier, and our experiment tests the search and update time for one keyword in which each keyword has 20 entries.

As shown in Figure 5, the number of total files (r) is from 100 to 2000, and m,n can be easily computed accordingly. By "traditional searching time" we mean that is the non-oblivious-cross-search approach with encryption. It is clear that our proposed keyword searching strategy can greatly

improve over the previous designed searching strategy as the number of files grows large.

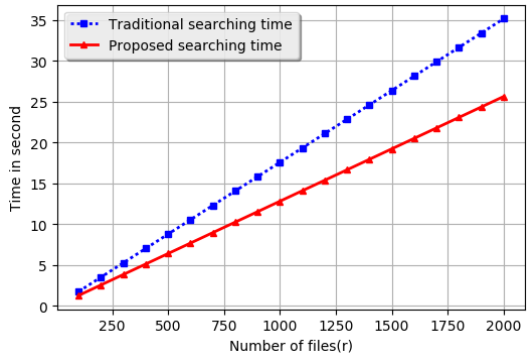


Fig. 5: Time complexity for large scale searching

5 Conclusion

We have revised encrypted cloud-based data share and search (with keyword update) framework as well as its security notion. We have further proposed an enhanced system satisfying the notion by leveraging identity-based encryption, asymmetric pairing group conversion, identity-based proxy re-encryption and "least frequent keyword" searchable technique. We have proved the security of the system in the generic bilinear group model. Our system is cost-effective as compared to its conference version, and has great potential in being deployed in large scale database.

This paper also leaves some interesting open problems. In the context of PKSE, it may be desirable to consider the forward and backward security, if the system provides file add and deletion. One may consider to use simulation-based or universal composability model define the security for forward/backward PKSE. Another research direction may be to consider the use of the technique proposed in [33] to hold against active online attacks.

ACKNOWLEDGEMENTS. This work is supported in part by the National Natural Science Foundation of China (Grant No. 61972094, 61822202 and 61702212), in part by the Singapore National Research Foundation under NCR Award Number NRF2018NCR-NSOE004-0001 and the AXA Research Fund, and in part by the Fundamental Research Funds for the Central Universities under Grand No. CCNU19TS017. Jiageng Chen is the corresponding author.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841 of *LNCS*, pages 111–131. Springer, 2011.

[3] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT '98*, pages 127–144. Springer, 1998.

[4] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *LNCS*, pages 506–522. Springer, 2004.

[5] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, 2007.

[6] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 185–194. ACM, 2007.

[7] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer, 2013.

[8] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *LNCS*, pages 577–594. Springer, 2010.

[9] Ronald Cramer and Victor 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.

[10] Liming Fang, Willy Susilo, Chunpeng Ge, and Jiandong Wang. Chosen-ciphertext secure anonymous conditional proxy re-encryption with keyword search. *Theor. Comput. Sci.*, 462:39–58, 2012.

[11] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security, Second International Conference, ACNS 2004*, volume 3089 of *LNCS*, pages 31–45. Springer, 2004.

[12] Matthew Green and Giusepagese Ateniese. Identity-based proxy re-encryption. In *ACNS '07*, volume 4512 of *LNCS*, pages 288–306. Springer, 2007.

[13] Debiao He, Mimi Ma, Sherali Zeadally, Neeraj Kumar, and Kaitai Liang. Certificateless public key authenticated encryption with keyword search for industrial internet of things. *IEEE Trans. Industrial Informatics*, 14(8):3618–3627, 2018.

[14] YongHo Hwang and PilJoong Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *Pairing-Based Cryptography Pairing 2007*, volume 4575 of *LNCS*, pages 2–22. Springer, 2007.

[15] Peng Jiang, Fuchun Guo, Kaitai Liang, Jianchang Lai, and Qiaoyan Wen. Searchain: Blockchain-based private keyword search in decentralized storage. *Future Gener. Comput. Syst.*, 107:781–792, 2020.

[16] Kaitai Liang, Man Ho Au, Joseph K. Liu, Willy Susilo, Duncan S. Wong, Guomin Yang, Tran Viet Xuan Phuong, and Qi Xie. A DFA-based functional proxy re-encryption scheme for secure public cloud data sharing. *IEEE Transactions on Information Forensics and Security*, 9(10):1667–1680, 2014.

[17] Kaitai Liang, Cheng-Kang Chu, Xiao Tan, Duncan S. Wong, Chunming Tang, and Jianying Zhou. Chosen-ciphertext secure multi-hop identity-based conditional proxy re-encryption with constant-size ciphertexts. *Theor. Comput. Sci.*, 539:87–105, 2014.

[18] Kaitai Liang, Joseph K. Liu, Duncan S. Wong, and Willy Susilo. An efficient cloud-based revocable identity-based proxy re-encryption scheme for public clouds data sharing. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security*, volume 8712 of *LNCS*, pages 257–272. Springer, 2014.

[19] Kaitai Liang, Chunhua Su, Jiageng Chen, and Joseph K. Liu. Efficient multi-function data sharing and searching mechanism for cloud-based encrypted data. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 83–94. ACM, 2016.

[20] Kaitai Liang and Willy Susilo. Searchable attribute-based mechanism with efficient data sharing for secure cloud storage. *IEEE Transactions on Information Forensics and Security*, 10(9):1981–1992, 2015.

[21] Kaitai Liang, Willy Susilo, Joseph K. Liu, and Duncan S. Wong. Efficient and fully CCA secure conditional proxy re-encryption from

hierarchical identity-based encryption. *Comput. J.*, 58(10):2778–2792, 2015.

[22] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *PKC'08*, volume 4939 of *PKC'08*, pages 360–379. Springer, 2008.

[23] PBC Library. http://crypto.stanford.edu/pbc, 2006. Online; accessed 18-Sep-2015.

[24] Joseph K. Liu, Man Ho Au, Willy Susilo, Kaitai Liang, Rongxing Lu, and Bala Srinivasan. Secure sharing and searching for real-time video data in mobile cloud. *IEEE Network*, 29(2):46–50, 2015.

[25] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 84(5):1234–1243, 2001.

[26] Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. Passive attacks against searchable encryption. *IEEE Trans. Information Forensics and Security*, 14(3):789–802, 2019.

[27] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[28] Jun Shao, Zhenfu Cao, Xiaohui Liang, and Huang Lin. Proxy re-encryption with keyword search. *Inf. Sci.*, 180(13):2576–2587, 2010.

[29] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (S&P 2007)*, pages 350–364. IEEE Computer Society, 2007.

[30] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[31] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 763–780. ACM, 2018.

[32] Peng Xu, Hai Jin, Qianhong Wu, and Wei Wang. Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. *IEEE Trans. Computers*, 62(11):2266–2277, 2013.

[33] Yi Zhao, Jianting Ning, Kaitai Liang, Yanqi Zhao, Liqun Chen, and Bo Yang. Privacy preserving search services against online attack. *Computers and Security*, 2020.

[34] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 522–530. IEEE, 2014.

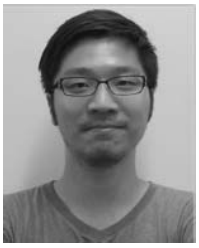
[35] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, pages 216–226, 1979.

[36] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018*, pages 228–246. Springer, 2018.

[37] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption with forward and stronger backward privacy. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security*, pages 283–303. Springer, 2019.

Jianting Ning received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University in 2016. He is currently a Professor with the Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Computer Science, Fujian Normal University, China. He is also a research fellow at School of Information Systems, Singapore Management University. His research interests include applied cryptography and information security, in particular, public key encryption, secure and privacy-preserving computation.

Jiageng Chen received PhD degree in computer science from the School of Information Science, Japan Advanced Institute of Science and Technology (JAIST) in 2012. He was working as an assistant professor in the School of Information Science, Japan Advanced Institute of Science and Technology from 2012 to 2015. He is currently an associate professor in the Computer School of Central China Normal University. His research areas include cryptography, especially in the areas of algorithms and cryptanalysis.



Kaitai Liang (M15) received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, Hong Kong, in 2014. He is currently an Assistant Professor with the Department of Computer Science, University of Surrey, Guildford, U.K. His current research interests include applied cryptography and information security in particular, encryption, blockchain, post-quantum crypto, privacy enhancing technology, and security in cloud computing.

Joseph K. Liu (Member, IEEE) received the Ph.D. degree from The Chinese University of Hong Kong in 2004. He is currently an Associate Professor with the Faculty of Information Technology, Monash University. He is currently the Lead of the Monash Cyber Security Group. He has established the Monash Blockchain Technology Centre in 2019, where he also serves as the Founding Director. His research areas include cyber security, blockchain, IoT security, applied cryptography, and privacy enhanced technology. He has received more than 5700 citations and his H-index is 43, with more than 170 publications in top venues, such as CRYPTO and ACM CCS.

Chunhua Su received the BS degree for Beijing Electronic and Science Institute, in 2003 and recieved the MS and PhD degree in computer science from Faculty of Engineering, Kyushu University, in 2006 and 2009, respectively. He is currently working as an assistant professor in School of Information Science, Japan Advanced Institute of Science and Technology. He has worked as a Scientist in Cryptography & Security Department of the Institute for Infocomm Research, Singapore from 2011-2013. His research areas include algorithm, cryptography, data mining and RFID security & privacy.

Qianhong Wu (Member, IEEE) received the M.Sc. degree in applied mathematics from Sichuan University, Sichuan, China, in 2001, and the Ph.D. degree in cryptography from Xidian University, Xian, China, in 2004. Since then, he has been an Associate Research Fellow with the University of Wollongong, Wollongong, Australia, an Associate Professor with Wuhan University, Wuhan, China, and a Senior Researcher with the Universitat Rovira i Virgili, Tarragona, Catalonia. His research interests include cryptography, information security and privacy, and ad hoc network security.