Practical and Dynamic Attribute-Based Keyword Search Supporting Numeric Comparisons Over Encrypted Cloud Data

Hui Yin, Yangfan Li, Hua Deng, Wei Zhang, Zheng Qin, and Keqin Li, *Fellow, IEEE*

Abstract—The attribute-based keyword search (ABKS), which simultaneously achieves searching and fine-grained access control over encrypted data, is frequently applied in cloud computing environments characterized by data storage and sharing. Recently, inspired by attribute-based encryption (ABE) and searchable encryption (SE) primitives, several ABKS schemes have been presented. However, almost all existing ABKS schemes actually only provide an attribute-based keyword equality match function and do not have a structural index to support practical search efficiency and dynamic data updates in real-world applications. To the best of our knowledge, this study is the first to realize an attribute-based keyword search construction supporting numerical comparison expressions with the practical search efficient and dynamic data update capacity (ABKS-NICEST), based on our proposed attribute-based keyword secure search scheme supporting numerical comparison expressions (ABKS-NICE) and an *exclusive OR-chain*-based inverted index structure. To the best of our knowledge, ABKS-NICEST is the first attributed-based keyword search scheme with practical search efficiency and dynamic data update capacity. In addition, numerical values are an important and common attribute, so providing comparison expressions among numerical values can greatly enhance the expressivity of access policy. Therefore, we use the prefix membership verification technique to design a method to support any numeric comparison expression in a flexible and uniform manner. Through theoretical and experimental evaluations, we determine that ABKS-NICEST is the most efficient ABKS scheme.

Index Terms—Attribute-based keyword search, cloud computing, data security and access control, dynamic data update, practical search efficiency

1 Introduction

1.1 Motivation

The enormous advantages of cloud computing have driven numerous enterprises and individuals to outsource applications and data to cloud platforms. The end-users are able to access applications and data through mobile devices anytime and anywhere. However, the security of cloud computing is always a controversial issue as storing data to cloud platforms also means that data owners discard the control power of their data. Without a sturdy protection measure, the malicious attackers including even the cloud server self can easily obtain the outsourced data [1].

Encryption before uploading data to the cloud server is most effective way to guarantee the confidentiality of data [2]. A challenging problem is how to efficiently perform operations over ciphertext exactly like being in the plaintext domain without decrypting them. It is critical for cloud applications, as the cloud computing is characterized by not only massive data storage, but efficient data processing.

A number of studies has been conducted to achieve operations directly over ciphertext, among which searchable encryption is a recently vibrant research field, aiming at guaranteeing both confidentiality and searchability over encrypted data. Song et al. [3] presented the first practical searchable encryption construction. Subsequent researches [4], [5] devoted to persistently refining the search complexity and security, even explored the more practical dynamic schemes [6], [7], [8] supporting secure data addition and deletion. As all these constructions were built in the symmetric key system, they are referred to as searchable symmetrical encryption (SSE).

In practical, access control is a required mechanism to prevent data from being illegally accessed by unauthorized entities, especially in the data-sharing cloud environment. This practical requirement has broken new ground in designing searchable encryption with data access control, called *attribute-based keyword search* (ABKS) [9]. Such

- *Hui Yin, Hua Deng, and Wei Zhang are with the College of Computer Engineering and Applied Mathematics, Changsha University, Changsha, Hunan 410022, China. E-mail: yhui@ccsu.edu.cn, {hdeng, zweihnu}@hnu.edu.cn.*
- *Yangfan Li is with the School of Computer Science and Engineering, Central South University, Changsha, Hunan 410083, China. E-mail: yangfanli@hnu.edu.cn.*
- *Zheng Qin is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410008, China. E-mail: zqin@hnu.edu.cn.*
- *Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 31 May 2022; revised 10 October 2022; accepted 21 November 2022. Date of publication 5 December 2022; date of current version 8 August 2023.
The work was supported in part by the National Natural Science Foundation of China under Grants 61972058, 61902123, 62002106, and 62002112, in part by the Natural Science Foundation of Hunan Province under Grants 2021JJ30760 and 2021JJ40636, and in part by the Research Project of Provincial Education Department of Hunan under Grants 20A041, 20B064, and 21B0775.
(Corresponding author: Yangfan Li.)
This article has supplementary downloadable material available at https://doi.org/10.1109/TSC.2022.3225112, provided by the authors.
Digital Object Identifier no. 10.1109/TSC.2022.3225112

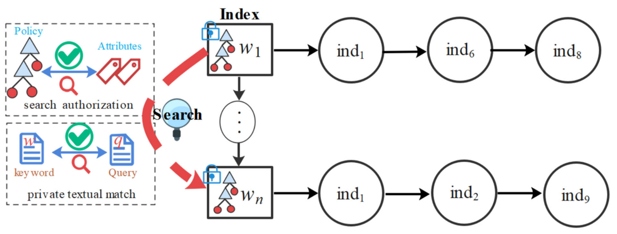


Fig. 1. Static and Linear Index in schemes [13] and [14].

solutions [10], [11], [12], [13], [14], [15] are able to achieve private search and fine-grained access control simultaneously in the ciphertext environment, inspired by attribute-based encryption [16] and searchable encryption. However, so far almost all existing ABKS schemes just provide an attribute-based keyword equality test functionality. How to employ them in real-world data set with practical search efficiency is still unknown. Meaningfully, these researches provide a theoretical conclusion for the technical feasibility of the attribute-based searchable encryption.

Inspired by searchable encryption, Work [13] and [14] equipped their ABKS with inverted index on the real-world data set, as shown in Fig. 1, where each encrypted keyword involves an access policy specifying its search permission and ind denotes the identifier of a data file containing keyword w. Though thus a construction can obtain search results from real world data set, it still suffers from an impractical search efficiency.

When a search query is issued, the search algorithm linearly scans encrypted keywords in such a way that for each keyword it has to first performs a search authorization as long as the search query has access to the keyword, followed by the private textual match between the search query and current keyword according to the intermediate result produced by the search authorization. Unfortunately, the search authorization based on ABE systems involves time-consuming public-key operations such as pairings and exponentiations. Though being able to obtain data files containing a certain search query, the simple index construction may lead to many redundant search authorizations that incur a long search process. For example, there is a super data user that has the search permissions of all index keywords. Some point in time, when she issues a search query with respect to the last keyword w_n in the index, $(n-1)$ search authorizations have to be conducted one by one, which introduces a large number of time-consuming pairing and exponentiation operations. More generally, if we refer to the index keywords that a search query can access as its the authorization keyword set (AKS), the average of needed pairing and exponentiation operations increases linearly with the size of AKS, which will become a performance bottleneck of this construction. Also, this is a simple static construction unsuitable for the real-world cloud application, without a mechanism to update data dynamically. Motivated by these limitations, in this paper, for the first time, we realize a practically efficient and dynamic ABKS construction based on our proposed *exclusive-OR chain*. The construction can ensure that the heavy search authorization is only performed one time at most regardless of the size of AKS, as well as the dynamic data and keyword search permission update. To the best of our knowledge, ABKS-NICEST is the first attributed-based keyword search scheme with practical search efficient and dynamic data update.

In addition, as few original ABE systems have focused on numerical value comparisons in access policies, almost all ABKS schemes do not consider the widely existing numeric comparison expression. Number is a kind of very important and common attribute in the practical application. For example, policy ((“DOCTOR” **OR** “PROFESSOR”) **AND** “COMPUTER SCIENCE” **AND** “$30 \leq age \leq 50$” **AND** “$level > 5$”) indicates that only person who is a doctor or a professor and works in the department of computer science and whose age is between 30 and 50 with the supervisory level greater than 5 can access the corresponding ciphertext. A naive approach to deal with the numeric comparison expressions “$30 \leq age \leq 50$” and “$level > 5$” is to enumerate all values within these two ranges and express them using **OR** gates, i.e., “*age*=30 **OR** 31...**OR** 50” and “*level*=6 **OR** 7... **OR** *max*,” where max may be a reasonably preset value to denote the maximum of the attribute *level*. Obviously, such an expression is impractical when the value space is large. In this paper, we propose to use the prefix membership verification technique to convert the numeric comparison expression into the equivalent **OR** gate policy and realize the match between a numerical attribute and the converted policy. Our approach can greatly reduce the number of **OR** gates and can be used by all ABE and ABKS schemes as an building-block to express numeric attribute comparisons.

1.2 Contributions

We make the following four contributions.

1. Based on the efficient and provably secure ABE scheme by Waters [17], we construct an Attribute-Based Keyword secure Search scheme supporting NumerIcal Comparison Expressions, named ABKS-NICE. Functionally, similar to existing ABKS schemes, ABKS-NICE is just an attribute-based keyword equality test scheme. However, except for supporting numerical comparison expressions, ABKS-NICE uses more general LSSS representation and has better efficiency due to the lesser pairing operations.
2. We design an inverted index construction based on encrypted XOR chain. Combining it and ABKS-NICE, for the first time, we propose the Attribute-Based Keyword Search construction supporting NumerIcal Comparison Expressions with the practical Search efficient and dynamic data updaTe capacity, ABKS-NICEST.
3. We use the prefix membership verification technique to realize an approach in ABE or ABKS systems that can support any numerical attribute comparison operation in a flexible and uniform manner.
4. We provide formal security proofs and extensive experiments for ABKS-NICE and ABKS-NICEST. The evaluation results show that ABKS-NICEST is a truly practical ABKS scheme over the real-world data set.

We believe that the technical contribution of ABKS-NICEST is to provide a feasible research line for the practical and dynamic attribute-based keyword search.

2 Related Work

2.1 Searchable Encryption

In [18], Chase and Kamara proposed the concept of the structured encryption. As its a particular case, searchable encryption has plentiful research results, recently. Song et al. [3] designed the first practical SSE and later Chang and Mitzenmacher [4] proposed a construction achieving forward privacy. The search time cost of both of them is linear in the size of the data files. It was not until 2006 when the SSE construction that achieves sub-linear search complexity was designed by Curtomla et al. [5]. They first proposed to use encrypted inverted index structure to speed up search process. Since then, the encrypted inverted index becomes a core data structure for designing efficient and practical SSE. In order to be applicable in real-word cloud storage systems, SSE was further extended to a dynamic environment, allowing for the dynamic and secure data addition and deletion [6], [7], [8]. Due to recently developed file injection attacks [19], the dynamic SEE constructions with forward privacy have been intensively researched [20], [21], [22], [23], [24]. The forward privacy guarantees that updating a document does not leak more information than what a predefined leakage function leaks and thus makes the file injection attack ineffective. Backward privacy is another stronger security notion. It requires that a search query does not reveal information from data that were deleted. Several SSE schemes achieving backward privacy are also proposed in [25], [26].

2.2 Attribute-Based Encryption and Attribute-Based Keyword Search

Sahai and Waters proposed the first attribute-based encryption scheme in [16]. Goyal et al. [27] and Bethencourt et al. [28] introduced more expressive attribute-based encryption by using tree based access policy. The former is a KP-ABE, which means the access policy is embedded into the key and the ciphertext is associated with a set of attributes. On the contrary, the later is that the access policy is in the ciphertext and the key is described by attributes, referred to as a CP-ABE. No matter what construction you chooses, a successful decryption has to require that the attributes satisfy the access policy. Many variants of ABE were proposed to acquire more excellent properties, such as multiple authorities [29] and hidden access structure [30].

Recently, in order to make the encrypted cloud data searchable with the flexible data access control, combining ABE and SE, a number of ABKS schemes are proposed in [10], [11], [12], [13], [14], [15]. However, these ABKS schemes just provide an attribute-based keyword equality test functionality and are short of effective index mechanism for a practical search efficiency in the real application. Though work [14] constructed a *raw* index by simply associating each encrypted keyword with data files containing the keyword to speed up the search process, without a full-fledged index structure, it still suffers from impractical search efficient in practice due to redundant and expensive search authorizations.

To enhance the expressivity of ABE scheme, authors [28] first considered to deal with numeric comparison expression in the access policy and transformed an integer comparisons into **AND** and **OR** gate policies over "bag of bits" divided by a numeric comparison expression. However, this transformation is relatively complex, resulting in the inefficiency. In order to degrade the complexity, Xue et al. [31] used only **OR** gates to convert access policies of numeric comparison expressions by using 0-encoding and 1-encoding technique. Though this scheme considers two inequality operations "$>$" and "$<$," it is defective to support other widely used relationships such as "$\neq$," "$\geq$," "$\leq$," etc.

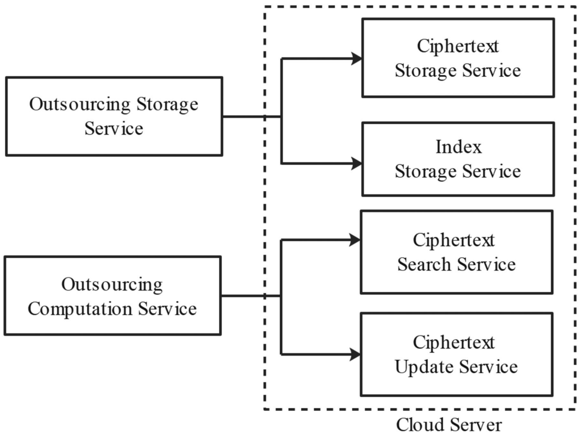

Outsourcing Storage Service
Outsourcing Computation Service
Ciphertext Storage Service
Index Storage Service
Ciphertext Search Service
Ciphertext Update Service
Cloud Server

Fig. 2. A service-oriented cloud computing outsourcing environment.

3 Preliminaries

In this section, we introduce several techniques used in our schemes, including Bilinear Pairing Map [16], Access Structure [28], Linear Secret Sharing Scheme (LSSS) [17], Prefix Membership Verification [32], Pseudo Random Function [5]. We present the detailed descriptions about these techniques in Appendix A.

4 Problem Formulations

Our attribute-based keyword search problem is based on a service-oriented cloud computing outsourcing environment, as shown in Fig. 2, where the cloud server provides powerful data storage and computation services. Roughly speaking, the cloud server provides a service to store outsourced encrypted data and secure index; if the cloud server receives an outsourced computation request, it will provide the corresponding computation services such as data searching or data updating based on ciphertexts. Our service-oriented computation model fully embodies the idea of "Service Computing," including "Storage-as-a-Service," "Software-as-a-Service," and "Security-as-a-Service". We will describe a more detailed system model in Section 4.1.

4.1 System Model

Our system model is illustrated in Fig. 3, in which three entities are involved. In order to make the outsourced data secure and searchable, before uploading the data, the data owner (DO) encrypts data using a symmetric encryption and construct searchable secure index using a CP-ABE scheme with access polices. Via secure communication channels, DO sends secret key with attributes to a data user

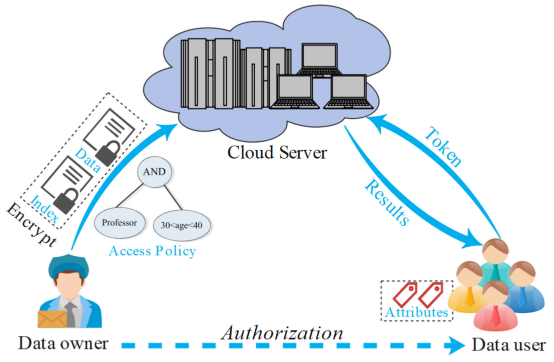


Fig. 3. The system model.

TABLE 1
Numeric Comparison Expressions and Corresponding Ranges

Comparison operation	Expression	Transformed range
Greater than	$attr > v$	$[v+1, max]$
Greater than or equal	$attr \geq v$	$[v, max]$
Less than	$attr < v$	$[min, v-1]$
Less than or equal	$attr \leq v$	$[min, v]$
Unequal	$attr! = v$	$[min, v-1] \bigvee [v+1, max]$
Equal	$attr = v$	$[v, v]$

(DU), by which DU can generate a token (encrypted keyword) he wants to search or update. The cloud server (CS) stores the encrypted data and index and, upon receiving a token, performs data searching if the token is a search token or data updating if it is a update token, on behalf of the data user. In ABKS-NICE and ABKS-NICEST, if and only if a data user has access to a certain index and his submitted search token matches the index, a successful search will complete.

4.2 Threat Model

We consider both DO and DU are fully trusted, but CS is an "honest-but-curious" passive adversary. In this threat model, the cloud server commits to obey with the security assurance protocol, but it may try to deduce as much information as possible from secure index, search query, and the encrypted data self. DO and DU do not leak their secret information to other parties such as unauthorized ones. In addition, in order to guarantee secure authorization to DU, we suppose that there exist secure communication channels between DO and DU.

5 Achieving Numeric Comparison Expressions in ABE

Simply converting a numeric comparison expression to the **OR** gate policy by enumerating all satisfied value may lead to impractical length of **OR** gates. To address this problem, we propose to use the prefix membership verification technique to reduce the number of OR gates and realize the match between a numerical attribute and the converted policy. The main idea is to convert a numeric comparison expression to a range $\mathcal{R}$ and compute the minimum set of prefixes $\mathcal{S}(\mathcal{R})$ of $\mathcal{R}$. For a numeric attribute n, it is denoted

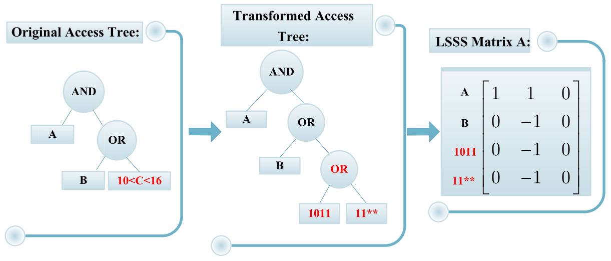


Fig. 4. The transformation of access policy.

as the prefix family $\mathcal{PF}(n)$. The prefix membership verification technique tells us that if n is in $\mathcal{R}$, there exists a common prefix in $\mathcal{S}(\mathcal{R})$ and $\mathcal{PF}(n)$. Thus, we convert the problem of whether a numerical attribute satisfies a numeric comparison policy to the verification of whether exists a common prefix in $\mathcal{S}(\mathcal{R})$ and $\mathcal{PF}(n)$.

Table 1 shows numeric comparison expressions and corresponding transformed ranges, where max and min are reasonable values to denote the maximum and the minimum of an attribute. For example, for attribute age, we can set max and min to be 0 and 100. Naturally, for expressions $v_1 < attr < v_2$ and $v_1 \leq attr \leq v_2$, the transformed ranges are $[v_1+1, v_2-1]$ and $[v_1, v_2]$, respectively.

Example. To help understand how our idea works, we give a concrete example. Consider an access policy "A AND (B OR 10 < C < 16) ". Our goal is to convert the numeric comparison expression "10 < C < 16" to OR gates of *non-numeric attributes*. First, "10 < C < 16" is converted to a range [11,15]. Then, we compute the minimum set of prefixes $\mathcal{S}([11,15]) = \{1011, 11^{**}\}$. Next, the transformed access policy can be represented as "A AND (B OR (1011 OR 11**))". Thus, expression "10 < C < 16" is converted to an *equivalent* policy of an OR gate with two prefix strings. Finally, we convert it to the corresponding LSSS matrix by a standard algorithm [29]. The access tree and the LSSS matrix are shown in Fig. 4.

Suppose that a decryptor has an attribute set {A, C=12}, which can be converted to the attribute set {A, 1100, 110*, 11**, 1***, ****} by solving the prefix family of 12. Due to having attributes "A" and "11**," the decryptor can decrypt a ciphertext encrypted under the policy "A AND (B OR (1011 OR 11**))".

Remark. Obviously, converting a comparison expression to the minimum set of prefixes actually increases the number of attributes, resulting in the extension of rows of the LSSS matrix. Theoretically, it will increase both storage and computation cost of the LSSS matrix. Fortunately, given a range $[v_1, v_2]$, its minimum set of prefixes $\mathcal{S}([v_1, v_2])$ contains at most $2h-2$ prefixes [33], where h denotes the bit length of v_1 and v_2, whose upper bound is a linear function of h. This is a very nice feature. Further, for all possible intervals $[v_1, v_2]$, if we assume all the intervals appear with the same probability, the average number of prefixes in $\mathcal{S}([v_1, v_2])$ is equal to $\frac{(h-2)2^{2h-1}+(h+1)2^h+1}{2^{2h-1}+2^{h-1}}$, which is approximately equal to $h-2$ [34]. For example, if we set h to be 8 bits, converting a numeric comparison expression only needs 14 prefixes (attributes) and 13 **OR** gates in the worst case. As a comparison, the naive approach based on numerical value enumeration produces 128 attributes and 127 **OR** gates in the worst case ($0 \leq attr \leq 127$).

6 ABKS-NICE

6.1 Construction

ABKS-NICE consists of SetUp, Enc, KeyGen, TokenGen, Search five algorithms. We describe the algorithm implementation in Fig. 5.

The work flow of ABKS-NICE incorporated into a service-oriented cloud computing environment is illustrated in Fig. 6. Though ABKS-NICE supports flexible numeric comparison expression policy, it only achieves the attribute-based textual matching between two encrypted keyword. Like existing ABKS schemes, it is still unknown to apply ABKS-NICE on real data set to facilitate practically efficient data searching and dynamic data updating. As the fundamental building block, ABKS-NICE will be integrated into ABKS-NICEST to achieve a truly practical, dynamic, and attribute-based searchable encryption.

6.2 Correctness

Theorem 1. *Given a ciphertext $\mathcal{I}_w$ of keyword w and LSSS matrix (A, ρ) and a search token $\mathcal{T}_{\mathcal{Q}}$ of query keyword $\mathcal{Q}$ and attribute set $\mathcal{U}$, if $\mathcal{U}$ satisfies (A, ρ) and $w = \mathcal{Q}$, the search algorithm **Search** will return true.*

Proof. First, if $\mathcal{U}$ satisfies (A, ρ), we can find a subset $I \subset \{1, 2, \ldots, |S|\}$ and a set of constants $\{\omega_i \in \mathbb{Z}_p^*\}_{i\in I}$ such that $\Sigma_{i\in I}\omega_i A_i = (1, 0, \ldots, 0)$, where I is defined as $I = \{i : \rho(i) \in \mathcal{U}\}$. Further, the algorithm computes

$$\begin{aligned} e((\mathcal{I}_w).\mathcal{I}_2, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_1) &= (g^s, g^{\alpha H_2(\mathcal{Q})} g^{xt\theta H_2(\mathcal{Q})}) \\ &= e(g^s, g^{\alpha H_2(\mathcal{Q})}) e(g^s, g^{xt\theta H_2(\mathcal{Q})}) \\ &= e(g, g)^{s\alpha H_2(\mathcal{Q})} e(g, g)^{sxt\theta H_2(\mathcal{Q})} \end{aligned}$$

and

$$\begin{aligned} &\prod_{i\in I} \left(e((\mathcal{I}_w).C_i, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_2) e\Big((\mathcal{I}_w).D_i, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_{\rho(i)}\Big) \right)^{\omega_i} \\ &= \prod_{i\in I} \left(e\Big(g^{x\lambda_i} H_1(\rho(i))^{-r_i}, g^{t\theta H_2(\mathcal{Q})}\Big) \cdot \right. \\ &\qquad \left. e\Big(g^{r_i}, H_1(\rho(i))^{t\theta H_2(\mathcal{Q})}\Big) \right)^{\omega_i} \\ &= \prod_{i\in I} \left(e\Big(g^{x\lambda_i}, g^{t\theta H_2(\mathcal{Q})}\Big) \cdot e\Big(H_1(\rho(i))^{-r_i}, g^{t\theta H_2(\mathcal{Q})}\Big) \cdot \right. \\ &\qquad \left. e\Big(g^{r_i}, H_1(\rho(i))^{t\theta H_2(\mathcal{Q})}\Big) \right)^{\omega_i} \\ &= \prod_{i\in I} \left(e\Big(g^{x\lambda_i}, g^{t\theta H_2(\mathcal{Q})}\Big) \cdot \right. \\ &\qquad \left. e(H_1(\rho(i)), g)^{-r_i t\theta H_2(\mathcal{Q}) + r_i t\theta H_2(\mathcal{Q})} \right)^{\omega_i} \\ &= \prod_{i\in I} \left(e\Big(g^{x\lambda_i}, g^{t\theta H_2(\mathcal{Q})}\Big) \right)^{\omega_i} \\ &= e\Big(g^x, g^{t\theta H_2(\mathcal{Q})}\Big)^{\sum_{i\in I} \lambda_i \omega_i} = e\Big(g^x, g^{t\theta H_2(\mathcal{Q})}\Big)^{\sum_{i\in I} \overrightarrow{v} A_i \omega_i} \\ &= e\Big(g^x, g^{t\theta H_2(\mathcal{Q})}\Big)^{(s, y_2, \ldots, y_n)(1, 0, \ldots, 0)} = e\Big(g^x, g^{t\theta H_2(\mathcal{Q})}\Big)^s. \end{aligned}$$

Therefore, we have

$$\begin{aligned} &\frac{e((\mathcal{I}_w).\mathcal{I}_2, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_1)}{\prod_{i\in I} \left(e((\mathcal{I}_w).C_i, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_2) e\Big((\mathcal{I}_w).D_i, (\mathcal{T}_{\mathcal{Q}}).\mathcal{T}_{\rho(i)}\Big) \right)^{\omega_i}} \\ &= \frac{e(g, g)^{sH_2(w)\alpha} e(g, g)^{sH_2(w)at\theta}}{e(g^a, g^{t\theta H_2(\mathcal{Q})})^s} \\ &\overset{w=\mathcal{Q}}{=\!=} e(g, g)^{\alpha s H_2(w)} \\ &= \mathcal{I}_w.\mathcal{I}_1. \end{aligned}$$

The proof is completed. □

6.3 Performance

Functionally, similar to the existing ABKS schemes [10], [11], [12], [13], [14], [15], (which were constructed over the access tree structure based ABE scheme of Bethencourt et. al. [28] and have asymptotically same computation complexity), ABKS-NICE is only an attribute-based keyword equality match scheme without providing the data index construction for real-world data set. However, ABKS-NICE has better match efficiency as the required pairing operations are less than that of the above mentioned schemes. Moreover, ABKS-NICE uses more general LSSS representation with better expressivity as well as supports numerical comparison expressions. We will conduct the theoretical and experimental evaluations of the algorithms in ABKS-NICE with the new scheme ABKS-NIEST together in Sections 7 and 8.

6.4 Security

Inspired by the security proof technique present in ABE scheme by Waters [17], we adopt the reduction idea to prove the security of ABKS-NICE. Loosely speaking, proving ABKS-NICE's security will be reduced to try to solve the decisional q-parallel Bilinear Diffie-Hellman Exponent problem [17]. However, the decisional q-parallel BDHE problem is acknowledgedly intractable in the polynomial time. Here, in brief, we review the decisional q-parallel BDHE problem/assumption as follows.

Let G, G_T be cyclic groups of prime order p and g be a generator of group G. The Decisional q-Parallel BDHE problem is defined that an adversary $\mathcal{A}$ has non-negligible advantage ϵ to solve decisional q-parallel BDHE problem if

$$\begin{aligned} \Big| \Pr[\mathcal{A}(\overrightarrow{\chi}, \mathcal{M} = e(g, g)^{x^{q+1}s}) = 0] \\ - \Pr[\mathcal{A}(\overrightarrow{\chi}, \mathcal{M} = R) = 0] \Big| \geq \epsilon, \end{aligned}$$

where $\overrightarrow{\chi} =$

$$\begin{aligned} &g, g^s, g^x, \ldots, g^{(x^q)}, g^{(x^{q+2}),\ldots,g^{(2q)}}, \\ &\forall_{1\leq j\leq q}\ g^{s\cdot y_j}, g^{x/y_i}, \ldots, g^{(x^q/y_j)}, g^{(x^{q+2}/y_j)}, \ldots, g^{(x^{2q}/y_j)} \\ &\forall_{1\leq j,k\leq q, k\neq j}\ g^{x\cdot s\cdot y_k/y_j}, \ldots, g^{(x^q\cdot s\ y_k/y_j)} \end{aligned}$$

and $x, s, y_1, \ldots, y_q \in \mathbb{Z}_p^*$ are chosen at random, R is randomly chosen from G_T. The decisional q-parallel BDHE assumption indicates that there is no adversary that can solve this problem within the polynomial time.

$\mathcal{P} \leftarrow \mathsf{SetUp}(1^l)$: When we input a sufficiently large security factor l, the algorithm chooses two multiplication cyclic groups $\mathbb{G}, \mathbb{G}_T$ of the order p. Choose a generator g of $\mathbb{G}$ and two random exponents $\alpha, x \in \mathbb{Z}_p^*$. Define a computationally efficient bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_p^*$. Finally, the algorithm outputs the public parameter $\mathcal{P} = (\mathbb{G}, \mathbb{G}_T, H_1, H_2, g, g^x, e(g,g)^\alpha)$ and master key $\mathcal{MK} = (x, \alpha)$.

$\mathcal{I}_w \leftarrow \mathsf{Enc}(\mathcal{P}, \tau, w)$: The DO runs the algorithm to encrypt a keyword. When we input a keyword w, an access policy τ with a set of attributes S, and the system parameter $\mathcal{P}$, the algorithm encrypts w as follows:

- First, if τ contains $l > 0$ comparison expression(s) with numerical attributes $\varsigma_1, ..., \varsigma_l$, it is converted into l ranges $\mathcal{R}_1, ..., \mathcal{R}_l$; the minimum set of prefixes $\{\mathcal{S}(\mathcal{R}_i)\}_{1\le i\le l}$ and $S := \cup_{1\le i\le l}((S - \varsigma_i) \cup \mathcal{S}(\mathcal{R}_i))$ is then computed; and τ is transformed by replacing each comparison expression with **OR** gates among prefixes in $\mathcal{S}(\mathcal{R}_i)_{1\le i\le l}$.
- Second, the algorithm generates the LSSS matrix of τ as (A, ρ), where A is a $|S| \times n$ matrix; extend s to $\overrightarrow{v} = (s, y_2, ..., y_n) \in \mathbb{Z}_p^n$ for random $y_2, ..., y_n$, and for $i = 1$ to $|S|$, calculate $\lambda_i = \overrightarrow{v} A_i$.
- Finally, it randomly chooses $r_1, ..., r_{|S|} \in \mathbb{Z}_p^*$ and encrypts w as

$$\mathcal{I}_w = \begin{cases} \mathcal{I}_1 = e(g,g)^{\alpha s H_2(w)}, \mathcal{I}_2 = g^s \\ \{C_i = g^{x\lambda_i} H_1(\rho(i))^{-r_i}, D_i = g^{r_i}\}_{1\le i\le |S|} \end{cases}$$

In this encryption, the access policy τ implicitly specifies who has the search permission to the keyword w.

$\mathcal{K} \leftarrow \mathsf{KeyGen}(\mathcal{U}, \mathcal{MK})$: The DO runs the algorithm to generate a private key for a specified attribute set $\mathcal{U}$. When we input a set of attributes $\mathcal{U}$ and $\mathcal{P}$, the algorithm chooses a random $t \in \mathbb{Z}_p^*$ and computes

$$\mathcal{K} = \begin{cases} k_1 = g^\alpha, k_2 = g^{xt}, k_3 = g^t \\ \forall u \in \mathcal{U} : k_u = H_1(u)^t \end{cases}$$

Data users who have the attribute set $\mathcal{U}$ can use $\mathcal{K}$ to encrypt a query keyword on which they want to perform search. Note that if $\mathcal{U}$ contains a numerical value v, the algorithm needs to calculate $\mathcal{PF}(v)$ and $\mathcal{U} \leftarrow (\mathcal{U} - v) \cup \mathcal{PF}(v)$.

$\mathcal{T}_\mathcal{Q} \leftarrow \mathsf{TokenGen}(\mathcal{K}, \mathcal{Q})$: The DU runs the algorithm to encrypt a query keyword $\mathcal{Q}$. When we input $\mathcal{K}$ and $\mathcal{Q}$, the algorithm chooses a random element $\theta \in \mathbb{Z}_p^*$ and encrypts $\mathcal{Q}$ as

$$\mathcal{T}_\mathcal{Q} = \begin{cases} \mathcal{T}_1 = (k_1(k_2)^\theta)^{H_2(\mathcal{Q})} = g^{\alpha H_2(\mathcal{Q})} g^{xt\theta H_2(\mathcal{Q})} \\ \mathcal{T}_2 = (k_3)^{\theta H_2(\mathcal{Q})} = g^{t\theta H_2(\mathcal{Q})} \\ \forall u \in \mathcal{U} : \mathcal{T}_u = (k_u)^{\theta H_2(\mathcal{Q})} = H_1(u)^{t\theta H_2(\mathcal{Q})} \end{cases}$$

An authorized DU with the attribute set $\mathcal{U}$ runs the algorithm to output the search token $\mathcal{T}_\mathcal{Q} = (\mathcal{T}_1, \mathcal{T}_2, \{\mathcal{T}_u\}_{u\in\mathcal{U}})$ of any query keyword $\mathcal{Q}$.

$\mathsf{true\ or\ false} \leftarrow \mathsf{Search}(\mathcal{I}_w, \mathcal{T}_\mathcal{Q})$: When we input a search token $\mathcal{T}_\mathcal{Q}$ and an encrypted keyword $\mathcal{I}_w$, if $\mathcal{U}$ hidden in $\mathcal{T}_\mathcal{Q}$ satisfies the $|S| \times n$ LSSS matrix (A, ρ) in $\mathcal{I}_w$, the algorithm can find a subset $I \subset \{1, 2, ..., |S|\}$ and a set of constants $\{\omega_i \in \mathbb{Z}_p^*\}_{i\in I}$ such that $\Sigma_{i\in I}\omega_i A_i = (1, 0, ..., 0)$, where I is defined as $I = \{i : \rho(i) \in \mathcal{U}\}$. The algorithm further computes

$$A = \frac{e((\mathcal{I}_w).\mathcal{I}_2, (\mathcal{T}_\mathcal{Q}).\mathcal{T}_1)}{\prod_{i\in I}\Big(e\Big((\mathcal{I}_w).C_i, (\mathcal{T}_\mathcal{Q}).\mathcal{T}_2\Big)e\Big((\mathcal{I}_w).D_i, (\mathcal{T}_\mathcal{Q}).\mathcal{T}_{\rho(i)}\Big)\Big)^{\omega_i}}$$

Finally, it verifies

$$A \overset{?}{=} \mathcal{I}_w.\mathcal{I}_1$$

When receiving token $\mathcal{T}_\mathcal{Q}$ from a DU, CS invokes the algorithm to perform an equality test between $\mathcal{T}_\mathcal{Q}$ and $\mathcal{I}_w$ without learning the underlying plaintexts of w and $\mathcal{Q}$.
If the equation above holds, this indicates a successful match ($\mathcal{U}$ satisfies (A, ρ) and $w = \mathcal{Q}$), and the algorithm returns true.
However, if the algorithm returns false, at least one of the following two conditions holds:

- $w \ne \mathcal{Q}$;
- There does not exist a subset $I \subset \{1, 2, ..., |S|\}$ and a set of constants $\{\omega_i \in \mathbb{Z}_p^*\}_{i\in I}$ such that $\Sigma_{i\in I}\omega_i A_i = (1, 0, ..., 0)$ (i.e., $\mathcal{U}$ does not satisfy access policy (A, ρ)).

The second condition also declares that the DU has no search permission for the keyword w.

Fig. 5. The construction of ABKS-NICE.

Also, we adopt the standard security model in the ABKS scheme, namely *Adaptively Chosen-Keyword Attack* model, which is formalized by the following game between a Challenger $\mathcal{B}$ and a PPT adversary $\mathcal{A}$.

Adaptively Chosen-Keyword Attack Game (ACKA):

Setup. $\mathcal{B}$ runs an algorithm Setup to establish the public parameter **P** and sends **P** to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ requests trapdoors $\mathcal{T}_{\mathcal{Q}_1}, \ldots, \mathcal{T}_{\mathcal{Q}_n}$ of keywords $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ corresponding to sets of attributes $S_1, \ldots, S_n$ by adaptively querying the following two oracles.

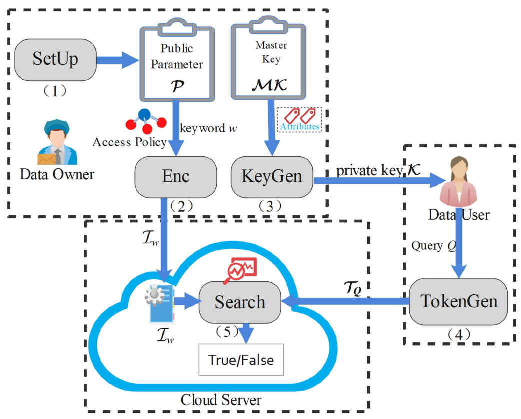


Fig. 6. The workflow of ABKS-NICE.

- $\mathcal{O}_{\mathsf{KeyGen}}(\mathbf{P},\ S_i)$: $\mathcal{B}$ runs an algorithm KeyGen to establish the private key $\mathcal{K}_i(1 \leq i \leq n)$ and sends it to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{TrapGen}}(\mathcal{K}_i,\ \mathcal{Q}_i)$: $\mathcal{B}$ runs an algorithm TrapGen to establish the query trapdoor $\mathcal{T}_{\mathcal{Q}_i}(1 \leq i \leq n)$ and sends it to $\mathcal{A}$.

Challenge. $\mathcal{A}$ chooses a challenging LSSS matrix A^* and two keywords w_0 and w_1, which are sent to $\mathcal{B}$. The restriction is that if $S_i(1 \leq i \leq n)$ satisfies A^*, then $w_0, w_1 \notin \{\mathcal{Q}_1, \ldots, \mathcal{Q}_n\}$. $\mathcal{B}$ randomly picks up a bit $b \in \{0, 1\}$ and encrypts w_b with A^*. $\mathcal{O}_{\mathsf{Enc}}$ responds the ciphertext $\mathcal{I}_{w_b}$ to $\mathcal{A}$.

Phase 2. $\mathcal{A}$ repeats *Phase 1*. There is a restriction that if an attribute set S_x corresponding to a requested trapdoor $\mathcal{T}_{q_x}$ satisfies A^*, then $w_0, w_1 \neq \mathcal{Q}_x$.

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

The advantage that a PPT adversary $\mathcal{A}$ wins the game is defined as $Adv = \mathbf{Pr}[b = b'] - \frac{1}{2}$.

Theorem 2. *Given the one-way secure hash H_2, when modeling H_1 as a random oracle, the proposed ABKS-NICE is secure against ACKA under the decisional q-parallel BDHE assumption.*

Refer to Appendix B to find the complete proof.

7 ABKS-NICEST

7.1 Main Idea

Our basic goal is to achieve attribute-based and dynamic searchable encryption with practical search complexity. In this section, we design ABKS-NICEST by employing the classic inverted index-based structural framework and ABKS-NICE scheme. In ABKS-NICEST construction, we propose the *exclusive-OR chain* to be able to dynamically form the inverted list of a keyword w with $\mathcal{O}(1)$ keyword authorization search. At a high level, when a new file (identifier) $\mathsf{ind}_{\mathsf{c}+1}$ containing w is about to be added in the corresponding list, the client first computes two secure update tokens T_c and $T_{\mathsf{c}+1}$ based on a locally maintained counter c, which counts the current number of files matching w before inserting $\mathsf{ind}_{\mathsf{c}+1}$; and then computes $T_\mathsf{c} \oplus T_{\mathsf{c}+1}$ and the encryption version $\widehat{\mathsf{ind}_{\mathsf{c}+1}}$ of $\mathsf{ind}_{\mathsf{c}+1}$. c is incremented and $(T_\mathsf{c}, T_{\mathsf{c}+1}, T_\mathsf{c} \oplus T_{\mathsf{c}+1}, \widehat{\mathsf{ind}_{\mathsf{c}+1}})$ is sent to the server. After receiving the update information, the server needs to perform two operations. One is to store $(T_\mathsf{c} \oplus T_{\mathsf{c}+1})$ along with $\widehat{\mathsf{ind}_{\mathsf{c}+1}}$ to a hash table at location $T_{\mathsf{c}+1}$. The second operation is that the server deletes w's search permission denoted by φ_w from entity at location T_c and re-attaches φ_w with entity $T_{\mathsf{c}+1}$, where φ_w implies an access policy determining who have permission to search with w.

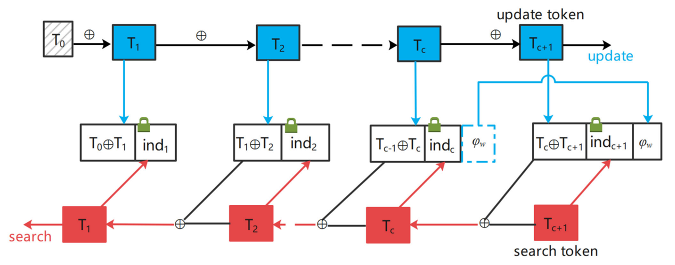


Fig. 7. Update and Search in ABKS-NICEST.

When a search query with respect to the search token $T_{\mathsf{c}+1}$ is issued, the server first performs a search permission verification between $T_{\mathsf{c}+1}$ and φ_w (search authorization). If $T_{\mathsf{c}+1}$ has the search permission to w, then it is arrowed to obtain $\mathsf{ind}_{\mathsf{c}+1}$ and $(T_\mathsf{c} \oplus T_{\mathsf{c}+1})$; by computing $(T_\mathsf{c} \oplus T_{\mathsf{c}+1} \oplus T_{\mathsf{c}+1})$, T_c is obtained subsequently leading to ind_c and $(T_{\mathsf{c}-1} \oplus T_\mathsf{c})$, and iteratively the server obtains all file identifiers until meets a sentinel T_0, which are sent to the client as search results; otherwise, the search returns nothing.

Fig. 7 shows the main idea of proposed ABKS-NICEST scheme. To place the ABKS scheme in the practice, the most essential goal is to let the computational complexity of the heavy attribute-based search authorization achieve $\mathcal{O}(1)$, regardless of the size of data files and keywords. We achieve this goal by designing the XOR chain construction and letting search permission of a keyword only associate with the latest entity to be added in the chain. Thus, in a search process, the search authorization can be performed at most one time. Meanwhile, thanks to XOR chain, the dynamic update of files and correct search can be achieved.

7.2 Construction

In essence, our proposed ABKS-NICEST is a combination of the ABKS scheme ABKS-NICE and a dynamic SSE scheme based on XOR chain index construction. We use the standard SSE protocol framework to describe ABKS-NICEST construction implementation. Formally, ABKS-NICEST consists of three polynomial-time protocols ABKS-NICEST=$(\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ between a client and a server.

Let $F: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ be a pseudo-random function, and $H_3: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ and $H_4: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^{3k}$ be two keyed hash functions, and $\psi: \{0,1\}^* \rightarrow \{0,1\}^{3k}$ be one-way hash function, where k is the system security parameter. Notation $a \overset{\$}{\leftarrow} X$ denotes that a is selected uniformly at random from a finite set X and notation $a \leftarrow X$ (or $X \rightarrow a$) denotes that a is the output of algorithm X, or X is assigned to a if X is only a value. Finally, we use notation Γ to denote the proposed ABKS-NICE scheme.

$\underline{\mathsf{Setup}(k; \perp)}$. The client takes a security parameter k as input, the algorithm generates a system master key K and invokes Γ.Setup and Γ.Keygen to obtain ABKS-NICE's public parameter, master key, and private key. Two empty hash tables $\mathbf{CT}$, $\mathbf{ST}$ are maintained by the client and the server, respectively. Algorithm 1 provides a pseudo-code description of setup implementation.

Algorithm 1. Setup$(k; \perp)$

```
Input: security parameter $k$
Output: system master key $\mathsf{K}$, ABKS-NICE's public parameter $\mathcal{P}$, master key $\mathcal{MK}$, private key $\mathcal{K}$, and two hash tables CT, ST
Client:
1:  $\mathsf{K} \xleftarrow{\$} \{0,1\}^k$
2:  $[\mathcal{P} = (\mathbb{G}, \mathbb{G}_T, H_1, H_2, g, g^x, e(g,g)^\alpha), \mathcal{MK} = (x, \alpha)] \leftarrow \Gamma.\mathsf{Setup}$
3:  $[\mathcal{K} = (k_1 \leftarrow g^\alpha, k_2 \leftarrow g^{at}, k_3 \leftarrow g^t, \{k_u \leftarrow H_1(u)^t\}_{u \in \mathcal{U}})] \leftarrow \Gamma.\mathsf{KeyGen}$
4:  CT $\leftarrow$ empty hash table
Server:
5:  ST $\leftarrow$ empty hash table
```

Update$(\mathsf{K}, \mathbf{CT}, \mathsf{ind}, w; \mathbf{ST})$. Update is a protocol between the sever and the client. Running this protocol, in the encryption way, a new file ind matching w is inserted into the inverted index with respect to keyword w, where ind denotes the file identifier with k bits length. Algorithm 2 provides a pseudo-code description of this protocol implementation.

Algorithm 2. Update$(\mathsf{K}, \mathbf{CT}, \mathsf{ind}, w; \mathbf{ST})$

```
Input: system key master $\mathsf{K}$, hash tables CT and ST, keyword $w$ and file identifier ind
Output: updated hash tables CT and ST
Client:
1:  $K_w \leftarrow F(\mathsf{K}, w||1), K'_w \leftarrow F(\mathsf{K}, w||2)$
2:  $(\mathsf{cnt}, \mathcal{I}_1) \leftarrow \mathbf{CT}[w]$.
3:  if $(\mathsf{cnt}, \mathcal{I}_1) = \perp$ then
4:  $T_0 \xleftarrow{\$} \{0,1\}^k$, $\mathsf{cnt} \leftarrow 0$
5:  $(\mathcal{I}_1, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|}) \leftarrow \Gamma.\mathsf{Enc}$
6:  else
7:  $R_{\mathsf{cnt}} \leftarrow F(K_w,' \mathsf{cnt})$
8:  $T_{\mathsf{cnt}} \leftarrow H_3(K_w, R_{\mathsf{cnt}})$
9:  end if
10: $R_{\mathsf{cnt}+1} \leftarrow F(K_w,' \mathsf{cnt}+1)$
11: $T_{\mathsf{cnt}+1} \leftarrow H_3(K_w, R_{\mathsf{cnt}+1})$
12: $U_{\mathsf{cnt}+1} \leftarrow T_{\mathsf{cnt}} \bigoplus T_{\mathsf{cnt}+1}$
13: $\widehat{\mathsf{ind}} \leftarrow (R_{\mathsf{cnt}}||U_{\mathsf{cnt}+1}||\mathsf{ind}) \bigoplus H_4(K_w, R_{\mathsf{cnt}+1})$
14:     Send $(\widehat{\mathsf{ind}}, T_{\mathsf{cnt}}, T_{\mathsf{cnt}+1}, \psi(\mathcal{I}_1), \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|})$ to the server.
15: $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$
16: $\mathbf{CT}[w] \leftarrow (\mathsf{cnt}, \mathcal{I}_1)$
Server:
17: $(\mu, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|}) \leftarrow \mathbf{ST}[T_{\mathsf{cnt}}]$
18: if $((\mu, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|}) \neq \perp)$ then
19: $\mathbf{ST}[T_{\mathsf{cnt}}] \leftarrow \mu \bigoplus \psi(\mathcal{I}_1)$
20: end if
21: $\mu \leftarrow \widehat{\mathsf{ind}} \bigoplus \psi(\mathcal{I}_1)$
22: $\mathbf{ST}[T_{\mathsf{cnt}+1}] \leftarrow (\mu, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|})$
```

The client takes the master key K, the hash table **CT**, the keyword w, and the file identifer to be inserted ind containing w as input, the protocol generates two secret keys K_w and K'_w using the pseudo-random function F under the master key K, and proceeds as follows. The client retrieves $\mathbf{CT}[w]$ to take the number cnt of files currently containing w and w's search permission component $\mathcal{I}_1$. If keyword w appears for the first time, the client invokes Γ.Enc to encrypt w as ciphertexts $\mathcal{I}_w$: $(\mathcal{I}_1, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|})$, sets a sentinel T_0 and initializes cnt to be 0 as well (lines 3-5). Recall that, in ABKS-NICE scheme, since the ciphertext $\mathcal{I}_w$ contains the access policy determining who have permission to search w, here, we call $\mathcal{I}_w$ the search permission of keyword w. Otherwise, it derives a pseudo-random value R_{cnt} using F from K'_w and cnt, and subsequently generates T_{cnt} using H_3 from K_w and R_{cnt}. The same operations are performed on $\mathsf{cnt}+1$ to generate $R_{\mathsf{cnt}+1}$ and $T_{\mathsf{cnt}+1}$. Here, the use of pseudo-random value R (line 7 and line 10) is to guarantee forward privacy, and we will discuss this security issue in next section. Next, a new encrypted entity is formed $\widehat{\mathsf{ind}}$ containing information R_{cnt}, $T_{\mathsf{cnt}} \oplus T_{\mathsf{cnt}+1}$, and identifier ind (lines 12 and 13). Finally, cnt is incremented and $(\mathsf{cnt}, \mathcal{I}_1)$ is stored in $\mathbf{CT}[w]$.

The server takes the hash table **ST** as input, it first tries to gain a value by retrieving $\mathbf{ST}[T_{\mathsf{cnt}}]$. If the value exists, this means ind is not the first file to be inserted containing w. The sever needs to *wipe* the search permission component $\mathcal{I}_w$ from current entity stored in $\mathbf{ST}[T_{\mathsf{cnt}}]$ (line 19). Finally, the server generates a new entity u by *associating* $\mathcal{I}_1$ with $\widehat{\mathsf{ind}}$ using a XOR operation (line 21). u along with the other two search permission components $\mathcal{I}_2$ and $\{C_i, D_i\}$ is stored in **ST** at locations $T_{\mathsf{cnt}+1}$ (line 22).

Algorithm 3. Search$(\mathsf{K}, \mathbf{CT}, w; \mathbf{ST})$

```
Input: system master key $\mathsf{K}$, hash tables CT and ST, query keyword $w$
Output: a set rest of search results
Client:
1:  $K_w \leftarrow F(\mathsf{K}, w||1), K'_w \leftarrow F(\mathsf{K}, w||2)$
2:  $(\mathsf{cnt}, \mathcal{I}_1) \leftarrow \mathbf{CT}[w]$.
3:  if $(\mathsf{cnt}, \mathcal{I}_1) = \perp$ then
4:  return $\emptyset$
5:  end if
6:  $R_{\mathsf{cnt}} \leftarrow F(K_w,' \mathsf{cnt})$
7:  $T_{\mathsf{cnt}} \leftarrow H_3(K_w, R_{\mathsf{cnt}})$
8:  $(\mathcal{T}_1, \mathcal{T}_2, \{\mathcal{T}_u\}_{u \in \mathcal{U}}) \leftarrow \Gamma.\mathsf{TrapGen}$
9:  Send $(K_w, R_{\mathsf{cnt}}, T_{\mathsf{cnt}}, \mathcal{T}_1, \mathcal{T}_2, \{\mathcal{T}_u\}, \mathsf{cnt})$ to the server.
    Server:
10: rest $\leftarrow \emptyset$
11: $(\mu, \mathcal{I}_2, \{C_i, D_i\}_{1 \le i \le |S|}) \leftarrow \mathbf{ST}[T_{\mathsf{cnt}}]$
12: if $(\exists \{\omega_i \in \mathbb{Z}_p^*\}_{i \in (I \subset \{1,2,\ldots,|S|\})})$ then
13: $\chi \leftarrow e(\mathcal{I}_2, \mathcal{T}_1) \Big/ \prod_{i \in I} \left( e(C_i, \mathcal{T}_2) e(D_i, \mathcal{T}_{\rho(i)}) \right)^{\omega_i}$
14: $\widehat{\mathsf{ind}} \leftarrow \mu \bigoplus \psi(\chi)$
15: $(\mathsf{ind}, U_{\mathsf{cnt}}, R_{\mathsf{cnt}-1}) \leftarrow \widehat{\mathsf{ind}} \bigoplus H_4(K_w, R_{\mathsf{cnt}})$
16: rest $\leftarrow$ rest $\bigcup \{\mathsf{ind}\}$
17: if $\mathsf{cnt} \ge 2$ then
18: for $c = \mathsf{cnt}$ to 2 do
19: $T_{c-1} \leftarrow U_c \bigoplus T_c$
20: $\widehat{\mathsf{ind}} \leftarrow \mathbf{ST}[T_{c-1}])$
21: $(\mathsf{ind}, U_{c-1}, R_{c-2}) \leftarrow \widehat{\mathsf{ind}} \bigoplus H_4(K_w, R_{c-1})$
22: rest $\leftarrow$ rest $\bigcup \{\mathsf{ind}\}$
23: end for
24: end if
25: end if
26: return rest
```

Search$(\mathsf{K}, \mathbf{CT}, w; \mathbf{ST})$. Search is a protocol between the client and the server. Running this protocol, the server will return to the client a set of file identifiers matching w or an empty set. Algorithm 3 provides a pseudo-code description of this protocol implementation.

TABLE 2
Notations for Evaluation

Notation	Description
Δ	keyword universe
Δ'	authorized keyword set of a given search query, $\Delta' \subseteq \Delta$
P	bilinear pairing operation
F	the set of data files
E, E_T	exponentiation operation in group G, G_T, respectively
H_1	hashing a string to an element in G
S	the set of attributes in an access policy
$\mathcal{U}$	the data user's attribute set
ξ	the least attribute set satisfying an access policy

The client takes the master key K, the table **CT**, and a query keyword w as input, two secret keys K_w and K'_w are derived from $w||1$ and $w||2$, respectively. The protocol proceeds as follows. The client retrieves $(\mathsf{cnt}, \mathcal{I}_w)$ from $\mathbf{CT}[w]$ (If cnt does not exist, the protocol will terminate in advance) and generates R_{cnt} and T_{cnt}. Also, the client invokes algorithm TrapGen in ABKS-NICE to generate a search trapdoor $\mathcal{T}_w$ used to verify whether this search has permission to obtain the results containing w.

The server takes the table **ST** as input and retrieves entity u matching w and w's search permission components $\mathcal{I}_2$ and $\{C_i, D_i\}$. If the attribute set embedded in $\mathcal{T}_w$ satisfies the access policy associated with $\mathcal{I}_2$ and $\{C_i, D_i\}$, the server can obtain the cnt^{th} file identifier $\mathsf{ind}_{\mathsf{cnt}}$, since (see Γ.Match algorithm)

$$e(\mathcal{I}_2, \mathcal{T}_1) \Big/ \prod_{i \in I} \left(e(C_i, \mathcal{T}_2) e(D_i, \mathcal{T}_{\rho(i)}) \right)^{\omega_i} = \mathcal{I}_1 = \chi,$$
$$\mu \bigoplus \psi(\chi) = \widehat{\mathsf{ind}} \bigoplus \psi(\mathcal{I}_1) \bigoplus \psi(\chi) = \widehat{\mathsf{ind}}.$$

Decrypt $\widehat{\mathsf{ind}}$ to obtain $(\mathsf{ind}_{\mathsf{cnt}}, U_{\mathsf{cnt}}, R_{\mathsf{cnt}-1})$ and add $\mathsf{ind}_{\mathsf{cnt}}$ into the result set rest (lines 15 and 16). Having $U_{\mathsf{cnt}} = T_{\mathsf{cnt}-1} \oplus T_{\mathsf{cnt}}$, $R_{\mathsf{cnt}-1}$, and T_{cnt}, by the XOR chain, the server iteratively obtains search results $\mathsf{ind}_{\mathsf{cnt}-1}, \mathsf{ind}_{\mathsf{cnt}-2}, \ldots, \mathsf{ind}_1$ (lines 18-22).

7.3 Performance

In order to highlight the practicality of ABKS schemes in the real scenario, we theoretically evaluate the computational performance trends of ABKS algorithms (especially the search efficiency) caused by the size of the index keyword universe. In the evaluation, for simplicity, we just consider heavy group operations in ABE environment (public-key setting), that is, the pairing operation, the exponentiation operation, and the hash operation H_1, while ignoring lightweight computations in symmetric cryptographic setting. As a comparison, we evaluate the similar work proposed in [14], the only ABKS scheme so far that the authors use real-world data set to evaluate its search efficiency via a simple inverted index. Recall that ABKS-NICE is also an attribute-based keyword equality match scheme, and we generate a same index construction as scheme [14] in a real data set to evaluate its performance. Several notations are shown in Table 2. We describe the computational cost of each algorithm for different schemes in Table 3.

Table 3 demonstrates that ABKS-NICE, ABKS-NICEST, and scheme [14] have the asymptotically same computational complexity in terms of KeyGen and Enc algorithms, and the TrapGen algorithm in scheme [14] keeps the constant exponentiation operations and is relatively efficient compared with ABKS-NICE and ABKS-NICEST. Note that, in ABKS-NICEST, the algorithm KeyGen, Enc, TrapGen is invoked by Setup, Update, and Search, respectively. We can observe that the time-consuming exponentiation operation E and hash operation H_1 in Enc are linear with the size of the keyword universe (the asymptotic complexity is $\mathcal{O}(|\Delta|)$, which will lead to a high computational overhead especially when the size of the keyword universe is large. However, to achieve retrieve on ciphertexts, the encrypted index construction may be a high price that any searchable encryption must pay for.

The search efficiency is of paramount importance in determining availability for a practical search system. Similar to scheme [14], given a search query, the expensive pairing and exponentiation operations in ABKS-NICE scale with the size of AKS of the search query. This is because those two schemes need to linearly scan the encrypted index keywords to obtain matched data files, which may lead to redundant search authorizations. As a result, the higher the search permission of a data user is, the higher the average search complexity is. An extreme case is that if a data user has the search permission of the whole index keyword universe, the average search complexity is $\mathcal{O}(|\Delta|)$. Note that ABKS-NICE has slightly better search efficient than scheme [14], as the pairing operations in ABKS-NICE are linear in $|\xi|$, but $|S|$ in Scheme [14], usually $|S| > |\xi|$.

ABKS-NICEST has the optimal search efficient with complexity $\mathcal{O}(1)$ on both search query and search authorization, regardless of the size of the keyword universe.

7.4 Security

As an application case of structured encryption, ABKS-NICEST is a searchable encryption by structurally indexing encrypted data. Therefore, in the real-world service, ABKS-NICEST unavoidably leaks information to the cloud server (adversary) for pursuing a practical search efficiency. This

TABLE 3
The Computational Cost of Each Algorithm in Different Schemes

Algorithms	ABKS-NICE scheme	ABKS-NICEST scheme	Scheme [14]	ABKS-HD [13]
KeyGen	$(3+\lvert\mathcal{U}\rvert)E + \lvert\mathcal{U}\rvert H_1$	$(3+\lvert\mathcal{U}\rvert)E + \lvert\mathcal{U}\rvert H_1$	$(3\lvert\mathcal{U}\rvert+2)E + \lvert\mathcal{U}\rvert H_1$	$(2\lvert\mathcal{U}\rvert+3)E + \lvert\mathcal{U}\rvert H_1$
Enc	$\lvert\Delta\rvert(E_T + \lvert S\rvert H_1 + (3\lvert S\rvert+1)E)$	$\lvert\Delta\rvert(E_T + \lvert S\rvert H_1 + (3\lvert S\rvert+1)E)$	$\lvert\Delta\rvert(E_T + \lvert S\rvert H_1 + (2\lvert S\rvert+1)E)$	$\lvert\Delta\rvert(E_T + \lvert S\rvert H_1 + (2\lvert F\rvert + 2\lvert S\rvert+1)E)$
TokenGen	$(3+\lvert\mathcal{U}\rvert)E$	$(3+\lvert\mathcal{U}\rvert)E$	$2E$	$(4+2\lvert\mathcal{U}\rvert)E$
Search	$\frac{1+\lvert\Delta'\rvert}{2}((1+2\lvert\xi\rvert)P + \lvert\xi\rvert E_T)$	$(1+2\lvert\xi\rvert)P + \lvert\xi\rvert E_T$	$\frac{1+\lvert\Delta'\rvert}{2}((1+2\lvert S\rvert)P + \lvert\xi\rvert E_T)$	$\frac{1+\lvert\Delta'\rvert}{2}((3+2\lvert S\rvert)P + \lvert\xi\rvert E_T)$

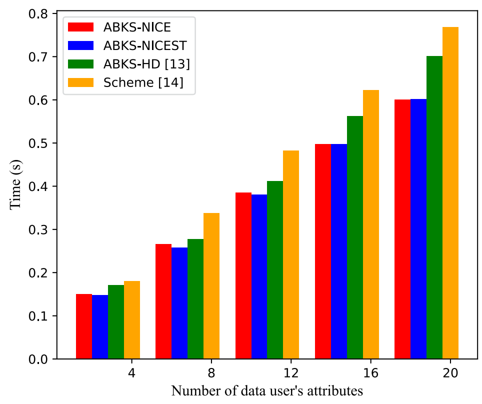


Fig. 8. Time cost of key generation.

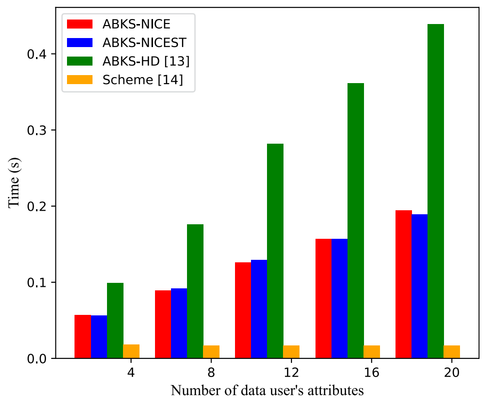


Fig. 9. Time cost of token generation.

is also a generally existing tradeoff between security and efficiency for searchable encryption. Our goal is to *minimize* the leakages. Typically, this is captured via the standard security model of a real-world experiment RealExp versus a ideal-world experiment IdeaExp. The security model is formalized by a group of *leakage function* $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Update}}, \mathcal{L}_{\mathsf{Search}})$, which specifies the upper limit of information that a dynamic searchable encryption scheme can reveal to adversary when running algorithm Setup and protocols Update and Search, correspondingly. We describe IdeaExp and RealExp experiments as follows.

- $\mathsf{RealExp}_{\mathcal{A}}(k)$: A PPT adversary $\mathcal{A}$ chooses a set of data files DB, the experiments first runs algorithm $\mathsf{Setup}(k;\perp)$ and returns **ST** to $\mathcal{A}$. Then, $\mathcal{A}$ performs search and update queries for polynomially-bounded times and receives a series of transcripts generated from Search and Update protocols. $\mathcal{A}$ observes these results and outputs a bit $b \in \{0,1\}$.
- $\mathsf{IdeaExp}_{\mathcal{A},\mathcal{S}}(k)$: A PPT adversary $\mathcal{A}$ chooses a set of data files DB, there exists a simulator $\mathcal{S}$ that runs the leakage function $\mathcal{L}_{\mathsf{Setup}}$ and returns $\mathcal{S}(\mathcal{L}_{\mathsf{Setup}}(k,\perp))$ to $\mathcal{A}$. Then, $\mathcal{A}$ performs search and update queries for polynomially-bounded times and receives a series of transcripts generated from the simulator $\mathcal{S}(\mathcal{L}_{\mathsf{Search}}(w))$ and $\mathcal{S}(\mathcal{L}_{\mathsf{Update}}(\mathsf{ind}, w))$. $\mathcal{A}$ observes these results and outputs a bit $b \in \{0,1\}$.

We say a searchable encryption is $\mathcal{L}$-adaptively-secure searchable encryption if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that:

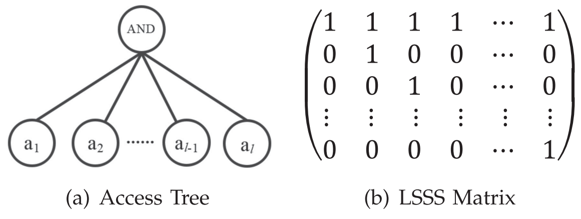


Fig. 10. Access policy used in our evaluation.

$$\left|\Pr(\mathsf{RealExp}_{\mathcal{A}}(k)=1) - \Pr\big(\mathsf{IdeaExp}_{\mathcal{A},\mathcal{S}}(k)=1\big)\right| \le \mathsf{negl}(k),$$

where k is a security parameter and $\mathsf{negl}(k)$ is a negligible function in k.

Theorem 3. *Given the pseudo-random function F, when modeling H_3 and H_4 as a random oracles, if q-parallel BDHE assumption holds, our proposed ABKS-NICEST is a dynamic $\mathcal{L}$-adaptively-secure searchable encryption scheme.*

Refer to Appendix C to find the complete proof.

Forward Privacy: Forward privacy claims that all search queries that have been issued before cannot be used to gain the newly added data files that are being updated. This property guarantees a update query cannot leak any keyword information in the newly added data, radically resists the file injection attack. Our proposed ABKS-NICEST achieves forward security since in Update protocol (Algorithm 2) we designedly introduce the pseudo-random value R with secret key K'. Without K', the server cannot generate up-to-date $R_{\mathsf{cnt}+1}$ and $T_{\mathsf{cnt}+1}$ from pervious counters $1,\ldots,\mathsf{cnt}$ (K' has been never given to the server). As a result, the newly updated data files cannot be searched using pervious search queries.

8 Experimental Evaluation

We experimentally evaluate ABKS-NICE and ABKS-NICEST in a real data set Enron Email Dataset.[1] As a comparison, we also implement the similar work ABKS-HD [13] and scheme [14] that the authors use inverted index construction to evaluate performance on real data set. All programmes are run at Java platform with JPBC[2] library and a Windows 7 with 3.30 GHZ Inter Core i7-11370H CPU, 16 GB memory. Our implementation uses the symmetric bilinear map Type A over a 160-bit elliptic curve group.

8.1 Efficiency for Key and Token Generation

In this section, we evaluate the efficiencies of algorithms KeyGen and TokenGen with varying number of data user's attributes. Fig. 8 shows a performance comparison among ABKS-NICE, ABKS-NICEST, ABKS-HD and Scheme [14]. We can observe that the time cost of all of the four schemes increases with the increasing number of attributes. ABKS-NICE and ABKS-NICEST have approximate computational cost to generate a private key, while ABKS-HD and Scheme [14] need slightly more time. For example, when setting $|\mathcal{U}| = 20$, ABKS-NICE, ABKS-NICEST, ABKS-HD, and Scheme [14] spend about 0.6 s, 0.601 s, 0.7 s, and 0.768 s on the private key generation, respectively. These results

1. Enron dataset. http://nlp.cs.aueb.gr/software
2. http://gas.dia.unisa.it/projects/jpbc/index.html

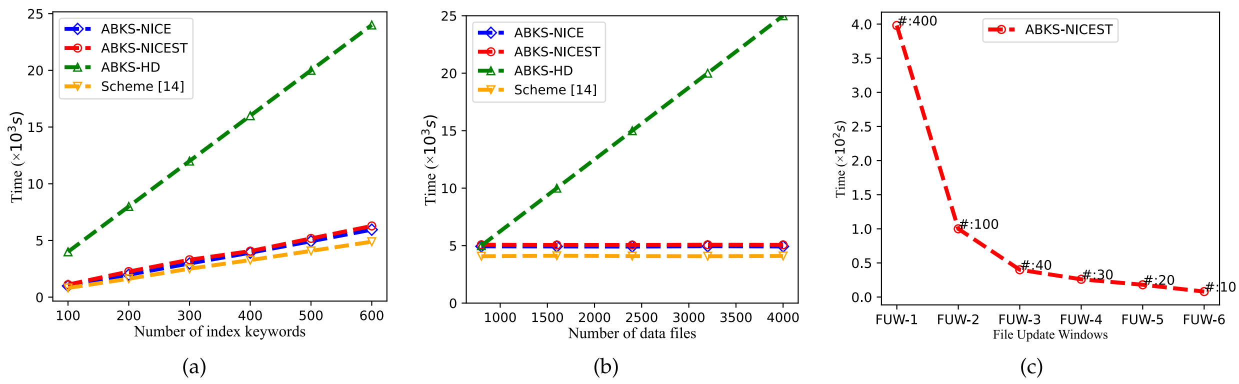


Fig. 11. Time cost of secure index construction. (a) For different number of index keywords with fixed number of data files ($|F| = 4000$) and attributes in the access policy ($|S| = 20$). (b) For different number of data files with fixed number of keywords ($|\Delta| = 500$) and attributes in the access policy ($|S| = 20$). (c) For different file update windows with fixed number of attributes in the access policy ($|S| = 20$), where $\#$ denotes the number of newly extracted keywords in current FUW.

are also in accordance with the theoretical evaluation in Table 3.

Fig. 9 shows the time cost of encrypting a search keyword when varying the number of data user's attributes. The experimental results show that Scheme [14] requires expending least cost to generate search trapdoor, and is moreover not affected by the number of attributes, while ABKS-NICE ABKS-NICEST, and ABKS-HD grow linearly. These experimental results conform with the theoretical analysis for the TokenGen algorithm in Table 3. For example, $\mathcal{U}| = 20$, ABKS-NICE, ABKS-NICEST, ABKS-HD, and Scheme [14] consume about 0.193 s, 0.189 s, 0.451 s, and 0.018 s, respectively.

8.2 Evaluation for Secure Index Construction

We extract 600 index keywords from 4000 data files in Enron Email Dataset. In practice, different index keywords should be specified personalized access policies to indicate their search permissions. However, a variety of structures of access policy raise the difficulty for our evaluations, especially for the search performance evaluations. Therefore, in our experiments, we use for all keywords unified access policy structure as "a_1 AND a_2 AND... a_l," where a denotes an attribute and $l = 20$. Its corresponding access tree (used in ABKS-HD and Scheme [14]) and LSSS matrix (used in ABKS-NICE and ABKS-NICEST) are shown in Fig. 10.

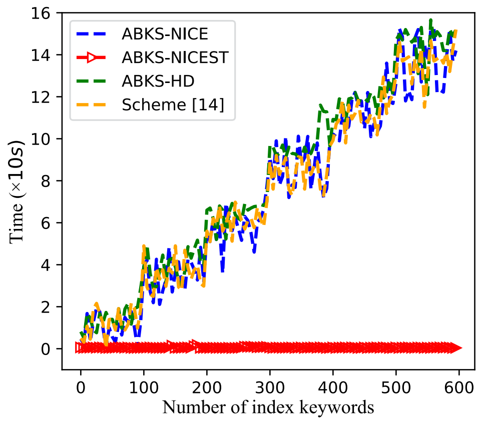
Fig. 11a shows the time cost of encrypting index keywords when varying the size of index keywords with fixed number of data files ($|F| = 4000$). Our experimental results show that the time cost of constructing secure index of all of the four schemes linearly increases with the number of index keywords. ABKS-HD needs much more time overhead than other three schemes, and ABKS-NICE and ABKS-NICEST have approximate computational overhead and are more than that of Scheme [14]. This is because ABKS-HD includes $2|\Delta||S| + 2|F||S|$ exponentiations, and ABKS-NICE and ABKS-NICEST include $3|\Delta||S|$, but $2|\Delta||S|$ in Scheme [14]. Compared to ABKS-NICE, ABKS-NICEST needs a little more time due to a number of light-weight computations such as XOR and hash operations. The second group experiment is to vary the number of data files and fix the size of index keywords ($|\Delta| = 500$). Fig. 11b demonstrates that ABKS-NICE, ABKS-NICEST, and Scheme [14] are not affected by the size of data files, but the time cost in ABKS-HD linearly grows. The results show that secure index construction is an extremely expensive process.

Figs. 11a and 11b demonstrate the one-time cost on index construction over a static data set. However, in the real-world application, for ABKS-NICEST scheme, the *every-time* cost required to expend on index keyword encryption will far less the above experimental results, as ABKS-NICEST is *dynamic*. Practically, the index keywords are encrypted in batches in the different update time point (File Update Window, FUW). Fig. 11c shows a group of experimental results for the different file update windows. In the current FUW, only these index keywords that have not appeared in the previous FUW need to be encrypted (See Algorithm 2, lines 2-5). Generally speaking, in the first update (FUW-1), the time cost of index construction achieves maximum, as all keywords appear for the first time. In the subsequent FUW, many keywords have been encrypted in the previous FUW, the time cost is drastically reduced.

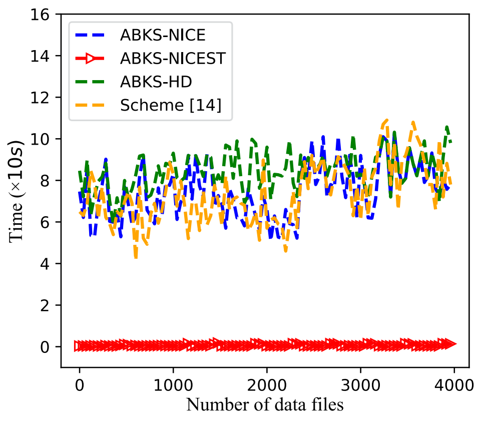
8.3 Evaluation for Data Search

For ease of evaluation and comparison, we set $|\xi|$ to be identical to $|S|$ and let any search query have the search permissions of all index keywords, i.e., set $|\xi| = |S| = |\mathcal{U}| = 20$. This is equivalent to a simulation that a super data user conducts data searching.

Fig. 12a demonstrates when varying the size of index keywords and fixing the number of data files, the time cost distributions of the search algorithms in ABKS-NICE, ABKS-NICEST, ABKS-HD, and Scheme [14]. In this evaluation, we perform 6 groups experiments with an incremental size of index keyword set. we repeat to perform 30 times search in each experiment, and choose query keywords from the currently incremental index keywords. For example, in the first group, the 30 query keywords are randomly chosen from range [0,100] (i.e., set $\{w_0, \ldots, w_{100}\}$), and second group is [101,200], and so on. The intention of choosing query keywords in this way is to observe the impact of the size of index keywords set on search efficiency. We can see that the time cost of ABKS-NICE, ABKS-HD, and Scheme [14] emerge an explicitly linear increase with the size of the index keyword set, while ABKS-NICEST is not affected by the number of index keywords. Fig. 12b shows that, when varying the size of



(a) For a different number of index keywords with fixed number of data files (n=4000).



(b) For a different number of data files with fixed number of index keywords (|Δ|=400).

Fig. 12. Time cost of data search.

data files and fixing the number of keywords, the time cost distributions of the search algorithms in ABKS-NICE, ABKS-NICEST, ABKS-HD, and Scheme [14]. In order to focus on the impact the size of data file set on search efficiency, we select query keywords from range [200,400]. This can effectively circumvent possible linear distribution of experimental results due to linearly choosing query keywords, which has been evaluated in Fig. 12a.

The remarkable conclusion is that, for ABKS-NICEST, the sizes of both data file set and index keyword set have very little influence on the search overhead; more importantly, the search time near to 0.06 s is extremely efficient, while in ABKS-NICE, ABKS-HD, and Scheme [14] the search complexities are too high to be accepted in practice. For example, in the worst-case, running a search needs to spend about 160 s in ABKS-NICE, ABKS-HD, and Scheme [14]. Practical search efficiency is crucial for the availability of a real-world search system.

9 Conclusion

In this paper, we investigate the attribute-based keyword search over encrypted cloud data. First, we construct an ABKS scheme supporting numeric attribute comparison policy, namely ABKS-NICE. Similar to the existing schemes, ABKS-NICE is *static* and with an impractical search complexity. Based on ABKS-NICE and our proposed encrypted XOR chain, we design the first truly practical and dynamic ABKS scheme, ABKS-NICEST. By the theoretical and experimental performance evaluations, ABKS-NICEST is most efficient ABKS scheme with dynamic data update ability as far as we know. Also, we provide formal security proofs for ABKS-NICE and ABKS-NICEST. As our future work, we will research practical and dynamic multi-keyword ABKS scheme, ABMKS-NICEST.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have helped to improve the quality of the manuscript.

References

[1] K. Ren, C. Wang, and Q. Wang, “Security challenges for the public cloud,” *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan./Feb. 2012.
[2] S. Kamara and K. Lauter, “Cryptographic cloud storage,” in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2010, pp. 136–149.
[3] D. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.
[4] Y. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2005, pp. 442–455.
[5] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved deinitions and efficient constructions,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.
[6] S. Kamara, C. Papamanthou, and T. Roeder, “Dynamic searchable symmeteric encryption,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
[7] S. Kamara and C. Papamanthou, “Parallel and dynamic searchable symmetric encryption,” in *Financial Cryptography and Data Security*, Berlin, Germany: Springer, 2013, pp. 258–274.
[8] F. Hahn and F. Kerschbaum, “Searchabel encryption with secure and efficient updates,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2014, pp. 310–320.
[9] Q. Zheng, S. Xu, and G. Ateniese, “VABKS: Verifiable attribute-based keyword search over outsourced encrypted data,” in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 522–530.
[10] J. Li, X. Lin, Y. Zhang, and J. Han, “KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage,” *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 715–725, Sep./Oct. 2016.
[11] Y. Miao, J. Ma, X. Liu, X. Li, Z. Liu, and H. Li, “Practical attribute-based multi-keyword search scheme in mobile crowdsourcing,” *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3008–3018, Aug. 2018.
[12] H. Wang, X. Dong, and Z. Cao, “Multi-value-independent ciphertext-policy attribute based encryption with fast keyword search,” *IEEE Trans. Serv. Comput.*, vol. 13, pp. 1142–1151, Nov./Dec. 2020.
[13] Y. Miao, J. Ma, X. Liu, X. Li, Q. Jiang, and J. Zhang, “Attribute-based keyword search over hierarchical data in cloud computing,” *IEEE Trans. Serv. Comput.*, vol. 13, no. 6, pp. 985–998, Nov./Dec. 2020.
[14] H. Yin, Z. Qin, J. Zhang, H. Deng, F. Li, and K. Li, “A fine-grained authorized keyword secure search scheme with efficient search permission update in cloud computing,” *J. Parallel Distrib. Comput.*, vol. 135, pp. 56–69, 2020.
[15] H. Yin et al., “CP-ABSE: A ciphertext-policy attribute-based searchable encryption scheme,” *IEEE Access*, vol. 7, pp. 5682–5694, 2019.
[16] A. Sahai and B. Waters, “Fuzzy identity-base encryption,” in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2005, pp. 457–473.
[17] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” in *Proc. Int. Workshop Public Key Cryptogr.*, 2011, pp. 53–70.
[18] M. Chase and S. Kamara, “Structured encryption and controlled disclosure,” in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Secur.*, 2010, pp. 577–594.
[19] Y. Zhang, J. Katz, and C. Papamanthou, “All your queries are belong to us: The power of file-injection attacks on searchable encryption,” in *Proc. USENIX Secur. Symp.*, 2016, pp. 707–720.

[20] R. Bost, "Σοφος: Forward secure searchable encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[21] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 912–927, Sep./Oct. 2020.

[22] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1538–1549, 2020.

[23] Y. Wang and D. Papadopoulos, "Multi-user collusion-resistant searchable encryption with optimal search time," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2021, pp. 252–264.

[24] Y. Watanabe, K. Ohara, M. Lwamoto, and K. Ohta, "Efficient dynamic searchable encryption with forward privacy under the decent leakage," in *Proc. ACM Conf. Data Appl. Secur. Privacy*, 2022, pp. 312–323.

[25] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou, "Dynamic searchable encryption withsmall client storage," in *Proc. Int. Conf. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[26] S. Sun et al., "Practical non-interactive searchable encryption with forward and backward privacy," in *Proc. Int. Conf. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[27] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encryption data," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.

[28] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, 2007, pp. 321–334.

[29] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2011, pp. 547–567.

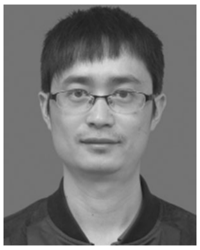
[30] J. Lai, Robert H. Deng, and Y. Li, "Expressive CP-ABE with partially hidden access structures," in *Proc. ACM Symp. Inf. Comput. Commun. Secur.*, 2012, pp. 18–19.

[31] K. Xue, J. Hong, Y. Xue, D. Wei, N. Yu, and P. Hong, "CABE: A new comparable attribute-based encryption construction with 0-encoding and 1-encoding," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1491–1503, Sep. 2017.

[32] J. Cheng, H. Yang, S. H. Wong, and S. Lu, "Design and implementation of cross-domain cooperative firewall," in *Proc. IEEE Int. Conf. Netw. Protoc.*, 2007, pp. 284–293.

[33] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.

[34] J. Li and E. R. Omiecinski, "Efficiency and security trade-off in supporting range queries on encrypted databases," in *Proc. Conf. Data Appl. Secur.*, 2005, pp. 69–83.



Hui Yin received the MS degree in computer software and theory from Central South University, China, in 2008, and the PhD degree from the College of Information Science and Engineering, Hunan University, China, in 2018. He is currently an associate professor with the College of Computer Science and Engineering, Changsha University, China. His interests include data security, privacy protection, and applied cryptography.

Yangfan Li received the bachelor's degree in engineering from the School of Automation, Huazhong University of Science and Technology, China, in 2015, and the PhD degree from the College of Computer Science and Electronic Engineering, Hunan University, China, in 2022. He is currently a lecturer with the School of Computer Science and Engineering, Central South University, China. His research interest includes, information security, parallel and distributed computing, machine learning, and deep learning.

Hua Deng received the MS degree in cryptography from Southwest Jiaotong University, China, in 2010, and the PhD degree in information security from Wuhan University, China, in 2015. Now he is an associate professor with the College of Computer Engineering and Applied Mathematics, Changsha University, China. His research interests include applied cryptography, data security and privacy, and cloud security.

Wei Zhang received the MS and PhD degrees from the College of Computer Science and Electronic Engineering, Hunan University. Now he is an Assistant Professor with the College of Computer Engineering and Applied Mathematics, Changsha University, China. His research interests include privacy protection, wireless networks, signal processing, and machine learning.

Zheng Qin received the PhD degree in computer software and theory from Chongqing University, China, in 2001. From 2010 to 2011, he served as a visiting scholar with the Department of Computer Science, Michigan University. He is currently a professor with the College of Computer Science and Electronic Engineering, Hunan University. His main interests include network and data security, privacy, machine learning, and applied cryptography.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a National Distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the worlds top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is an AAIA fellow. He is also a Member of Academia Europaea (Academician of the Academy of Europe).