Multi-Client Boolean File Retrieval with Adaptable Authorization Switching for Secure Cloud Search Services

Kai Zhang, Xiwen Wang, Jianting Ning, Mi Wen, *Member, IEEE,* and Rongxing Lu, *Fellow, IEEE*

Abstract—Secure cloud search services provide a cost-effective way for resource-constrained clients to search encrypted files in the cloud, where data owners can customize search authorization. Despite providing fine-grained authorization, traditional attribute-based keyword search (ABKS) solutions generally support single keyword search. Towards expressive queries over encrypted data, multi-client searchable symmetric encryption (MC-SSE) was introduced. However, current search authorizations of existing MC-SSEs: (i) cannot support dynamic updating; (ii) are (semi-)black-box implementations of attribute-based encryption; (iii) incur significant cost during system initialization and file encryption. To address these limitations, we present AasBirch, an MC-SSE system with fast fine-grained authorization that supports adaptable authorization switching from one policy to any other one. AasBirch achieves constant-size storage and lightweight time cost for system initialization, file encryption and file searching. We conduct extensive experiments based on Enron dataset in real cloud environment. Compared to state-of-the-art MC-SSE with fine-grained authorization, AasBirch achieves 30$\sim$200$\times$ smaller public parameter and secret key size, with the assumed least frequent keyword in a query (s-term) as 21. Moreover, it runs 10$\sim$20$\times$ faster for file encryption and $>$20$\times$ faster for file searching. In addition, AasBirch outperforms 80,000$\times$ (resp. 7,850$\times$) faster with s-term=1 (resp. =21), as compared to classic dynamic ABKS system.

Index Terms—Cloud Storage, Searchable Encryption, Keyword Search, Boolean Query, Access Control.

1 Introduction

CURRENTLY, an increasing number of resource-restrained users have moved private files to the cloud. The adoption of cloud-based corporate data storage has increased from 30% in 2015 to 50% in 2021, according to a recent report released in Statista [1]. To effectively retrieve multi-client shared files in cloud, cloud search services have been widely adopted in practice (such as Google, Amazon). Consequently, configuring flexible search permissions on whether a client satisfies search authorization policy is naturally raised. In particular, the changes of search permissions may cause the authorization policies to be updated accordingly. Due to security and privacy concerns, files are usually first encrypted before being outsourced to the cloud [2]. Fig. 1 illustrates a conceptual example of secure multi-client cloud search services, where data owners can customize search authorization policy over different data users. Generally, there are two *design goals* for secure multi-client cloud search services [3]:

1) *Non-interactive, flexible and fine-grained authorization.* Data owners can non-interactively configure and update flexible fine-grained authorizations, for whether a client has files retrieval and access permissions.

- *Kai Zhang, Xiwen Wang and Mi Wen are with College of Computer Science and Technology, Shanghai University of Electric Power, Shanghai, China.*
- *Jianting Ning is with the Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, Fuzhou, China.*
- *Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada.*

Manuscript received April 19, 2005; revised August 26, 2015.

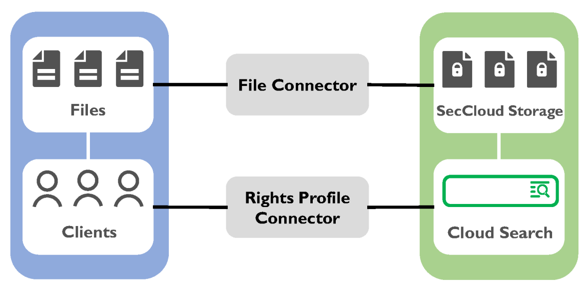


Fig. 1. Secure Multi-Client Cloud Search Services

2) *Effective, efficient and expressive file retrieval.* Data users can process expressive keyword search over data owners' encrypted files and efficiently receive corresponding searching results from the cloud.

Nevertheless, simple data encryption may introduce challenges of securely sharing and searching over encrypted data in cloud. To address the challenges, the concept of *searchable encryption (SE)* [4] was introduced and rapidly developed in the public-key setting [5], particularly for achieving fine-grained access control [6] towards a set of clients. To date, there are two main approaches proposed for constructing multi-client SE schemes with search authorization: *attribute-based keyword search (ABKS)* and *multi-client searchable symmetric encryption (MC-SSE).* In general, ABKS systems [7], [8], [9], [10] provide fine-grained search authorization in a variety of practical applications, but rarely support conjunctive even boolean queries for scalable storage. On the other hand, MC-SSE systems [11], [12], [13], [14], [15], enable multi-client boolean queries over encrypted data with sub-linear complexity, while have not formally considered dynamic updating for search authorization. Technically, ABKS combines public-key encryption with keyword search (PEKS) and attribute-based encryption (ABE). Therefore, ABKS mainly focuses on access control for keyword search while MC-SSE primarily studies on data sharing in multi-client setting.

Attribute-Based Keyword Search. An effective ABKS system [7] can be seen as a non-trivial framework that combines attribute-based encryption (ABE) [16] and somewhat key delegating or search token extraction techniques, such as proxy re-encryption. Based on inherent characters of fine-grained access control over encrypted data provided by ABE, a number of ABKS works realize owner-enforced authorization towards multiple clients with enhanced functionalities in various applications [7], [8], [9], [10], [17], [18], [19], such as data verifiable, auditable in E-health or cloud storage services. In particular, a dynamic ABKS system was recently proposed to capture dynamic authorization updating, however, it only supports single keyword search over encrypted data [17].

In fact, most of existing ABKS schemes only support single keyword search (or less support conjunctive keyword search), where the searching cost increases in linear with the number of stored files. Especially, the cost may be even more expensive for processing boolean queries in highly-scalable cloud storage services. Therefore, efficiently realizing expressive query models for multi-client SE is desirable.

Multi-client SSE. To enable efficient conjunctive/boolean queries over encrypted data, Cash et al. [20] presented a searchable symmetric encryption (SSE) scheme with sub-linear searching cost, which is followed by [21], [22]. Nevertheless, the schemes in [20], [21], [22] only work in the symmetric setting that outsourced data can only be written and read by a data owner. Later, [23] extended SSE into a multi-client situation, but has not realized search authorization for multiple clients. To achieve fine-grained search authorization, Sun et al. [11] presented a non-interactive MC-SSE system that employs an ABE module in a black-box manner. Very recently, Zhang et al. [15] have attempted an MC-SSE scheme with owner-enforced attribute-based authorization based on [20], which is partially regarded as a semi-black-box implementation of ABE for search authorization. Unfortunately, the attribute universe that describes clients is statically fixed in system initialization. A new dynamic MC-SSE scheme was proposed in [14], however, it cannot support non-interactive and fine-grained authorizations.

To sum up, state-of-the-art MC-SSEs with fine-grained authorization are generally modular frameworks that combines SSE and ABE, which loses high efficiency for system initialization, file encryption and file searching. Besides, these solutions have not formally considered authorization dynamic updating, that is, the systems need to be costly reset once the data owner changes authorization policies. Such consumption of resource led by repetitive encryption also indicated the importance of policy switching.

▷ **Motivation.** According to claimed design goals, the limitations in all known non-interactive MC-SSE schemes for secure cloud search services are concluded as:

i) no support for flexible, fine-grained authorization switching;
ii) (semi-) black-box implementations for search authorization;
iii) small attribute universe for static descriptions of clients.

Hence, this work is motivated to introduce an effective and efficient non-interactive boolean MC-SSE scheme with fine-grained authorization configuration and updating.

1.1 Our Results

In this paper, we propose a dynamic boolean file retrieval system for secure cloud search services that supports adaptable fine-grained authorization switching, termed *AasBirch*. In AasBirch, data users are allowed to search over data owner's encrypted files in cloud, under a non-interactive owner-enforced search authorization. In addition, AasBirch supports authorization dynamic updating for data owners, in which authorization configuration and switching are directly achieved rather than (semi-)black-box implementations of strong cryptographic primitives like ABE.

Besides enabling *multi-client boolean keyword searching for scalable cloud search services with sub-linear searching cost*, the main features of AasBirch are as follows:

1) **Direct, fast implementation for search authorization.** We give a direct approach for realizing fine-grained "AND"-gate authorization in MC-SSE, rather than (semi-)black-box implementations of ABE as existing works [11], [13], [15]. For any attribute of a client's attribute set ($\mathsf{att}_i \in \Sigma$) and any attribute in an authorization ($\mathsf{att}'_i \in \Lambda$), we consider $\{(x_i, y_i)\}$ and $\{(x'_i, y'_i)\}$ as two sets of points of a Lagrange interpolation polynomial and compute a product of coefficients Δ and Δ'. Hence, only data users with the same attributes that required in Λ can recover master secret key α from its secret key $(*, g^{\alpha r}, \Delta^{\alpha})$, where α have been shared for each att_i and att'_i. To prevent data users with the same attributes set (as a data owner) from forging an authorization updating request, we attach the data owner's transformation key tk with a proof π generated by a message authentication code scheme.
2) **Fine-grained authorization with dynamic updating.** Inspired by [24], [25], we introduce a new *adaptable authorization switching* module for AasBirch: TKGen and AuzAdp algorithms. This module allows a data owner to switch authorization policy *from one* "$\Lambda = \mathsf{att}_1 \wedge \mathsf{att}_2 \wedge \cdots$" *to any other* "$\Lambda' = \mathsf{att}'_1 \wedge \mathsf{att}'_2 \wedge \cdots$", for non-interactively updating search authorizations over different data users, while not generating corresponding transformation keys for each user as [9], [17]. By generating a transformation key tk of $\Lambda \Rightarrow \Lambda'$ that includes two respective set of points $(\{x_i, y_i\}, \{x'_i, y'_i\})$ of $\Psi_k(x), \Psi'_k(x)$ and partial conversion tuples, a data owner can thus allow the cloud to transform encrypted files under from Λ switching to any Λ'. In particular, the cloud obtains no knowledge of authorizations Λ, Λ' and file plaintext information.
3) **Efficient system initialization and file encryption.** To reduce system initialization cost, we consider the description of clients as a dynamic large-universe $\mathcal{U} = \{0, 1\}^*$, which allows Setup algorithm to avoid preparing $2n$ variables for each att_i of a small-universe description $\mathcal{U} = \{0, 1\}^n$ as [15]. Moreover, we introduce a

TABLE 1
Non-Interactive Multi-client Searchable Encryption with fine-grained authorization

Work	MC	BQ	Authorization			ABE	Universe	Time Cost			Storage Cost		
			NI	FG	SW			Setup	Encryption	Searching	PP	sk	EDB
[7], [9]	✔	✗	✔	✔	✗	●	◪	-	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert)$	-	-	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$
[17]	✔	✗	✔	✔	✔	●	□	$\mathcal{O}(1)$	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$
[14]	✔	✔	✗	✗	✔	NA	NA	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathsf{DB}\rvert)$	$\mathcal{O}(q \cdot c_{w_1})$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(\lvert\mathsf{DB}\rvert)$
[11], [13]	✔	✔	✔	✔	✗	●	◪	$\mathcal{O}(n)$	$\mathcal{O}(\lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(q \cdot c_{w_1})$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$
[15]	✔	✔	✔	✔	✗	◐	■	$\mathcal{O}(n)$	$\mathcal{O}(\lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(q \cdot c_{w_1})$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$
AasBirch	✔	✔	✔	✔	✔	○	□	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathsf{DB}\rvert)$	$\mathcal{O}(q \cdot c_{w_1})$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\mathsf{DB}\rvert)$

[1] Let "MC" denote "Multi-Client", "BQ" denote "Boolean Query", "NI" denote "Non-Interactive", "FG" denote "Fine-Grained", "SW" denote "Policy Switching", "ABE" denote "ABE Usage", "Universe" denote "Attribute Universe" and "PP, sk, EDB" denote public parameter, secret key and encrypted files index.
[2] Let "●, ◐, ○, NA" denote a "Black-Box, Semi-Black-Box, No Use, Not Applicable" implementation of ABE for authorization; and "■, ◪, □, NA" denote a "Static, Static/Dynamic, Dynamic, Not Applicable" attribute-based universe description for clients.
[3] Let "n" denote the base of considered attribute universe, "n_k" denote the number of keywords associated with a file, "q" denote the number of keywords in a query, "c_{w_1}" denote the number of files related to s-term, "$\lvert\mathsf{DB}\rvert$" denote the number of files in DB and "$\lvert\Sigma\rvert$" denote the number of attributes of an encrypted file.

dummy authorization $\Lambda_0 = \mathsf{att}_0 \wedge \mathsf{att}'_0$ for a data owner first-time encrypting files that outsourced to cloud, later allow it to generate a transformation key for cloud transforming encrypted files under $\Lambda_0 \Rightarrow \Lambda$ without confidential information leakage. As a result, AasBirch achieves constant-size public parameter, secret key and encrypted files, which are independent of an actual owner-enforced authorization Λ.

The high-level construction idea and formal description of our AasBirch system are concluded in Section 4. In addition, we give a formal security analysis of AasBirch under the threats from adversarial server and clients. Table. 1 shows a general feature and efficiency comparison between AasBirch and state-of-the-art multi-client SE solutions. As shown in Table. 1, AasBirch achieves more desirable functionalities, in particular, *non-interactive fine-grained authorization with adaptable switching*. Besides, the authorization is effectively, efficiently and directly realized over a dynamic large-universe description of clients, while does not rely on (semi-)black-box implementations of ABE to achieve fine-grained search authorizations as all existing solutions. Furthermore, the storage and time cost of AasBirch are highly efficient.

To illustrate practical performance, we implement state-of-the-art non-interactive fine-grained MC-SSE [15], dynamic ABKS [17] and AasBirch based on Enron dataset [26] in real HUAWEI Cloud environment [27]. For consumed time cost on the sides of client, client-to-cloud communication and cloud, AasBirch outperforms [15], [17] for system initialization, file encryption and searching. In particular, AasBirch achieves 0.49 KB constant-size PP and 0.36 KB constant-size sk, which is respectively roughly 30∼250 times smaller and 30∼200 times smaller than state-of-the-art MC-SSE with fine-grained authorization [15]. In addition, the authorization switching cost of a data owner is only 0.2% of file encryption under a dummy authorization, where #attributes in the authorization switches from 90 to 100. Moreover, AasBirch runs 20× faster than existing solutions for file encryption and more than 10,000× faster than traditional DABKS [17] for file searching.

Organization. Section 2 reviews preliminaries and Section 3 defines problem formulation. We present AasBirch system and analyze its security in Section 4 and Section 5. The experiment and performance analysis are given in Section 6, and Section 7 shows some discussions and comparisons with related work. Section 8 concludes this work.

2 Background Knowledge

The notations through this work are listed in Table 2.

TABLE 2
Notations

Notation	Meaning
att	An attribute.
att_0	A dummy attribute.
Σ	An attribute set $\Sigma = (\mathsf{att}_1, \mathsf{att}_2, \cdots)$.
$\mathcal{U}$	A universe description for clients.
Λ	An authorization policy.
Λ_0	A dummy authorization policy.
ind	The indice of a document.
w	A keyword.
W	A set of keywords $W = (w_1, w_2, \cdots, w_n)$.
Doc	A document Doc is labeled with (ind, W_{ind}).
DB	An outsourced database.
$\mathsf{DB}[w]$	The indices of documents laberled with keyword w.
s-term	The least frequent keyword in a query.
xterm	Any queried keyword in a query.

Definition 1. A keyword dictionary δ manages a set of tuples (w, c), in which w is a keyword and c is a counter. Generally, there are two functions in δ:

- $c \leftarrow \mathsf{Get}(\delta, w)$: For any keyword w in δ, the function outputs the counter of w; else directly returns 0.
- $\mathsf{Update}(\delta, w, c)$: For any keyword w in δ, the function updates the counter of a keyword w to c. Otherwise, it inserts the tuple (w, c) into δ.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3227650

TABLE 3
The workflow of AasBirch

Data Owners $(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}})$		Cloud $(\mathsf{PP}, \mathsf{pk}_{\mathsf{Ser}}, \mathsf{sk}_{\mathsf{Ser}})$		Data Users $(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}})$
$\mathsf{Encrypt}(\mathsf{PP}, \mathsf{Doc}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda_0)$	$\xrightarrow{\mathsf{EDoc}}$			
$\mathsf{TKGen}(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda, \Lambda')$	$\xrightarrow{(\mathsf{tk},\pi)}$			
		$\mathsf{AuzAdp}(\mathsf{PP}, \mathsf{EDoc}, \mathsf{sk}_{\mathsf{Ser}}, \mathsf{tk}, \pi) \hookrightarrow \mathsf{EDoc}$		
			$\xleftarrow{\mathsf{Token}}$	$\mathsf{TrapGen}(\mathsf{sk}_{\mathsf{Clnt}}, Q)$
		$\mathsf{Search}(\mathsf{Token})$	$\xrightarrow{R}$	
				$\mathsf{Retrieve}(\mathsf{sk}_{\mathsf{Clnt}}, R) \hookrightarrow \mathsf{DOC}$

Definition 2. Let $\Psi(x) = \sum_{i=0}^{k} y_i p_i(x) = \sum_{i=0}^{k} \Delta_i x^i$ be a Lagrange interpolation polynomial of degree k passing $k+1$ distinct points $\{(x_i, y_i) = (x_i, \Psi(x_i))\}_{i=0}^{k}$, where

$$p_i(x) = \prod_{0\le j\ne i\le k} \frac{x - x_j}{x_i - x_j} = \begin{cases} 1, & x = x_i \\ 0, & x \in \{x_0, \cdots, x_k\}\backslash\{x_i\}. \end{cases}$$

Definition 3. A message authentication code (MAC) is a primitive that used to authenticate a message with generating a tag, which has the following algorithms:

- $\mathsf{Gen}(\kappa) \to K'$: The key generation algorithm inputs a security parameter κ and produces a random key K';
- $\mathsf{Mac}(K', m) \to \pi$: The tag generation algorithm inputs a message m and outputs a valid tag π;
- $\mathsf{Veri}(K', \pi, m) \to \{0, 1\}$: The verification algorithm inputs π and m, and returns 1 if π is a valid tag for m; otherwise, it outputs 0.

For any PPT adversary $\mathcal{A}$, a secure unforgeable MAC scheme implies that the $\mathcal{A}$'s winning advantage $\mathsf{Adv}_{\mathcal{A},\mathsf{MAC}}^{\mathsf{Unforgeable}}(\kappa)$ of forging a tag π' for m (passes Veri algorithm) is negligible.

Definition 4. A pseudo-random function (PRF) F is an efficiently computable function that simulates a random oracle, where no probabilistic polynomial time (PPT) algorithms can distinguish between F and a random function F'. For any PPT adversary $\mathcal{A}$, the F is said to be a secure PRF if $\mathcal{A}$'s winning advantage $\mathsf{Adv}_{\mathcal{A},F}^{\mathsf{PRF}}(\kappa) = |\Pr[\mathcal{A}^{F(K,\cdot)}(1^\kappa)] - \Pr[\mathcal{A}^{F'(\cdot)}(1^\kappa)]| \le \mathsf{negl}(\kappa)$ holds, where $K \xleftarrow{\$} \{0,1\}^\kappa$.

Definition 5. Consider a cyclic group $\mathbb{G}$ of prime order p, any positive integer a, and g is a generator of $\mathbb{G}$ and h is an element of $\mathbb{G}$. For any PPT adversary $\mathcal{A}$, the Discrete Logarithm (DL) assumption implies that the $\mathcal{A}$'s winning advantage $\mathsf{Adv}_{\mathcal{A},\mathbb{G}}^{\mathsf{DL}}(\kappa) = |\Pr[\mathcal{A}(g, g^a)] - \Pr[\mathcal{A}(g, h)]|$ of distinguishing g^a from h is negligible.

Definition 6. Consider a cyclic group $\mathbb{G}$ of prime order p, and g is a randomly chosen element from $\mathbb{G}$ and a, b, r are randomly chosen from $\mathbb{Z}_p$. For any PPT adversary $\mathcal{A}$, the Decisional Diffie-Hellman (DDH) assumption implies that $\mathcal{A}$'s winning advantage $\mathsf{Adv}_{\mathcal{A},\mathbb{G}}^{\mathsf{DDH}}(\kappa) = |\Pr[\mathcal{A}(g, g^a, g^b, g^{ab})] - \Pr[\mathcal{A}(g, g^a, g^b, g^r)]|$ of distinguishing (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^r) is negligible.

3 Problem Formulation

To address the lack of flexible fine-grained authorization switching, we formalize system model, function definition, design goals and security guarantee model for AasBirch.

3.1 System Model

There are three different entities in the AasBirch system:

- **Authority:** It is a trusted entity that initializes a system with publishing public parameters. Moreover, it distributes public key and private key pair for cloud servers, and secret key for clients.
- **Cloud:** The cloud is a *semi-honest* server that honestly runs algorithm and provides encrypted files searching and authorization switching services for clients.
- **Clients:** The clients include multiple data owners and multiple data users. A data owner stores files on the cloud with enforcing an authorization policy, while data users who satisfy the policy could search the files.

Function Definition. The AasBirth system includes the following functions (as depicted in Table 3).

- $\mathsf{Setup}(1^\kappa, \mathcal{U}) \to (\mathsf{PP}, \mathsf{MK})$: Input a security parameter κ and attribute universe $\mathcal{U}$, the authority runs the setup algorithm to generate a public parameter PP and a master key MK.
- $\mathsf{KeyGen}_{\mathsf{Ser}}(\mathsf{PP}, \mathsf{MK}) \to (\mathsf{pk}_{\mathsf{Ser}}, \mathsf{sk}_{\mathsf{Ser}})$: Input PP and MK, the authority runs the server key generation algorithm to generate a pair of public key and secret key $(\mathsf{pk}_{\mathsf{Ser}}, \mathsf{sk}_{\mathsf{Ser}})$ for the server.
- $\mathsf{KeyGen}_{\mathsf{Clnt}}(\mathsf{MK}, \Sigma) \to \mathsf{sk}_{\mathsf{Clnt}}$: Input MK and an attribute set Σ, the authority runs the client key generation algorithm to generate a secret key $\mathsf{sk}_{\mathsf{Clnt}}$ for a client.
- $\mathsf{Encrypt}(\mathsf{PP}, \mathsf{Doc}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda_0) \to \mathsf{EDoc}$: Input PP, a secret key $\mathsf{sk}_{\mathsf{Clnt}}$, a document Doc and a dummy authorization policy Λ_0, the client runs the encryption algorithm to generate encrypted documents EDoc.
- $\mathsf{TKGen}(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda, \Lambda') \to (\mathsf{tk}, \pi)$: Input PP, a secret key $\mathsf{sk}_{\mathsf{Clnt}}$, an authorization Λ and an updated authorization Λ', the client runs the transformation key generation algorithm, and generates a transformation key tk and a proof π.
- $\mathsf{AuzAdp}(\mathsf{PP}, \mathsf{EDoc}, \mathsf{sk}_{\mathsf{Ser}}, \mathsf{tk}, \pi) \to \mathsf{EDoc}'$: Input PP, encrypted documents EDoc, a server secret key $\mathsf{sk}_{\mathsf{Ser}}$ and a transformation key tk and a proof π, the server runs the policy adaptable switching algorithm to generate an updated encrypted documents EDoc' under other policy.

- $\mathsf{TrapGen}(\mathsf{sk}_{\mathsf{Clnt}}, Q) \rightarrow \mathsf{Token}$: Input $\mathsf{sk}_{\mathsf{Clnt}}$ and a query Q, a client runs the trapdoor generation algorithm, and generates a search token Token.
- $\mathsf{Search}(\mathsf{Token}) \rightarrow R$: Input a search token Token from a client, the server runs the search algorithm, and returns the corresponding search result R.
- $\mathsf{Retrieve}(\mathsf{sk}_{\mathsf{Clnt}}, R) \rightarrow \mathsf{Doc}$: Input a client secret key $\mathsf{sk}_{\mathsf{Clnt}}$ and the search result R, the client runs the file retrieval algorithm and gets corresponding files Doc.

3.2 Design Goals

The design goals of AasBirch are formalized as follows.

- **Non-interactive fine-grained authorization.** Data owners can non-interactively enforce fine-grained search authorizations for multiple data users, where satisfied users can search over the data owners' encrypted files.
- **Dynamic authorization updating.** The system supports adaptable authorization switching from one authorization to any other one for data owners.
- **Efficient, expressive searching query.** For a data user's boolean queries, the cloud can return corresponding results with sub-linear complexity searching cost.
- **High running efficiency.** The system achieves fast algorithm-running efficiency and low entity-communication overhead.

3.3 Security Guarantee Model

The security threats of AasBirch system comes from both adversarial server and adversarial clients, that is:

3.3.1 *Security against adversarial server*

Based on the defined security model of SSE [11], [20], this security implies that the view of the cloud can be simulated given only the output of a leakage function $\mathcal{L}$ for (non-)adaptive attacks.

Definition 7. Let Π be a AasBirch scheme that presented in Section 4, we define the security via the following two experiments by two efficient algorithms $\mathcal{A}$ and $\mathcal{S}$:

$\mathsf{Real}^{\Pi}_{\mathcal{A}}(\kappa)$: $\mathcal{A}(\kappa)$ continually chooses an encryption tuple $(\mathbf{D}, \mathsf{sk}_{\mathsf{OW}}, \Lambda_0)$ or a query tuple $(Q, \mathsf{sk}_{\mathsf{USR}})$, where $\mathsf{sk}_{\mathsf{OW}}, \mathsf{sk}_{\mathsf{USR}}$ respectively denotes the secret key of a data owner or a data user, and Λ_0 is a dummy authorization policy. The algorithm returns (EDB,XSet) to $\mathcal{A}$ via running $\mathsf{Encrypt}(\mathsf{PP}, \mathbf{D}, \mathsf{sk}_{\mathsf{OW}}, \Lambda_0)$ and $\mathsf{TKGen}(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda, \Lambda')$ for a chosen encryption tuple; otherwise returns R to $\mathcal{A}$ via running $\mathsf{TrapGen}(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}}, Q)$ and $\mathsf{Search}(\mathsf{Token})$. Finally, the experiment outputs a bit from $\{0, 1\}$.

$\mathsf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa)$: This experiment initializes two empty lists $\mathbf{d}$ and $\mathbf{q}$, with setting two counters $i = 1$ and $j = 1$. $\mathcal{A}(1^\kappa)$ continually picks an encryption tuple $(\mathbf{D}, \mathsf{sk}_{\mathsf{OW}}, \Lambda_0)$ or a query tuple $(Q, \mathsf{sk}_{\mathsf{USR}})$. For a chosen encryption tuple, $\mathcal{A}$ records it as $\mathbf{d}[i]$ with increasing i, and the experiment returns $(\mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathcal{S}(\mathcal{L}(\mathbf{d}, \mathbf{q}))$ to $\mathcal{A}$. Otherwise, the experiment records it as $\mathbf{q}[j]$ with increasing j, and outputs a transcript to $\mathcal{A}$ by $\mathcal{S}(\mathcal{L}(\mathbf{d}, \mathbf{q}))$. Finally, the experiment outputs a bit from $\{0, 1\}$.

▷ **Formalized Leakage function $\mathcal{L}$.** In the two algorithms $\mathcal{A}$ and $\mathcal{S}$, the leakage function $\mathcal{L}(\mathbf{d}, \mathbf{q})$ is formally defined as $\mathcal{L}(\mathbf{d}, \mathbf{q}) = \{\mathsf{op}, N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{SRP}, \mathsf{dRP}, \mathsf{IP}, \mathsf{xt}\}$, whose outputs are as follows (the leakage information in RP, SRP and SRP is overstated):

- op is an array that records an "encrypt" or a "search" type for each operation, whose length is $|\mathsf{op}| = |\mathbf{d}|+|\mathbf{q}|$. In particular, the knowledge of each operation $\mathsf{op}[i]$ is directly known (leaked) to the cloud.
- N is an array that records the number of keywords XSet in each encrypted file EDB.
- $\bar{\mathbf{s}}$ denotes the *equality pattern* of a set of s-terms. For example, we set $\bar{\mathbf{s}} = (1, 2, 1, 3, 2)$ for $\mathbf{s} = (a, b, a, c, b)$.
- $\mathsf{RP}[i, \alpha] = \mathsf{DB}[\mathbf{s}[i]] \cap \mathsf{DB}[\mathbf{x}[i, \alpha]]$ records the revealed indices from the intersection of s-term and any xterm in a query. Let $\mathsf{RP}[i, \alpha, d]$ be the ind of $\mathsf{RP}[i, \alpha]$ in $\mathbf{d}[d]$.
- $\mathsf{SRP}[i] = \mathsf{DB}[\mathbf{s}[i]]$ denotes the matching results of s-terms in the i-th query.
- IP records some partial results between the intersection of two s-terms $(\mathbf{s}[i_1]]$ and $\mathbf{s}[i_2]])$. That is, $\mathsf{IP}[i_1, i_2, \alpha, \beta] = \mathsf{DB}[\mathbf{s}[i_1]] \cap \mathsf{DB}[\mathbf{s}[i_2]]$ if $\mathbf{s}[i_1] \neq \mathbf{s}[i_2], \mathbf{x}[i_1, \alpha] = \mathbf{x}[i_2, \beta]$. Otherwise, it is an empty set.
- $\mathsf{dRP}[i][j] == 1$ indicates that the s-term $\mathbf{s}[j]$ is used to retrieve $\mathsf{EDB}[l]$ generated by $\mathbf{d}[i]$; otherwise it is 0.
- $\mathsf{xt[i]} = |\mathbf{x}[i, \cdot]|$ is the number of xterms in the i-th query.

Assume an efficient algorithm $\mathcal{S}$ exits, we say that Π is $\mathcal{L}$-semantically secure against an adversarial adversary if

$$\Pr[\mathsf{Real}^{\Pi}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathsf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

3.3.2 *Security against adversarial clients*

The security implies that the colluded clients cannot forge search tokens of data users and switching trapdoor proofs of data owners.

Definition 8. Let Π be a AasBirch scheme that presented in Section 4, we define the security via the following game $\mathsf{CollUFExp}^{\Pi}_{\mathcal{A}}(\kappa)$ that played by a challenger and an adversary $\mathcal{A}$:

Initialization. The challenger runs the setup algorithm and returns public parameters $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{U})$ to $\mathcal{A}$.

Key Extraction. $\mathcal{A}$ adaptively issues any secret key query under an attribute set Σ, and the challenger runs the key generation algorithm and returns corresponding secret keys $\mathsf{sk}_{\mathsf{Clnt}} \leftarrow \mathsf{KeyGen}_{\mathsf{Clnt}}(\mathsf{MK}, \Sigma)$ to $\mathcal{A}$.

Challenge. $\mathcal{A}$ chooses a challenge attribute set Σ^* that not queried before and sends it to the challenger. Finally, the challenger returns a query keyword W to $\mathcal{A}$.

Output. For a challenge attribute set Σ^*, $\mathcal{A}$ outputs *a search token* related with W and *a transformation key*. It indicates that $\mathcal{A}$ wins the game if its output is valid.

For any PPT adversary $\mathcal{A}$, we say that Π is secure against adversarial colluded clients if

$$\Pr[\mathcal{A} \text{ wins in } \mathsf{CollUFExp}^{\Pi}_{\mathcal{A}}(\kappa)] \leq \mathsf{negl}(\kappa).$$

4 AasBirch: System Description

In this section, we start from give a high-level description and technical overview of our AasBirch. In the following, we propose an effective conjunctive AasBirch system that supports "AND"-gate authorization. An enhanced boolean AasBirch is discussed in Section 4.5.

4.1 High-level Description

In our AasBirch, the authority initializes the system via running $\mathsf{Setup}(1^\kappa, \mathcal{U}) \to (\mathsf{PP}, \mathsf{MK})$, and produces a public key and private key pair to the cloud and secret keys to clients via running $\mathsf{KeyGen}_{\mathsf{Ser}}(\mathsf{PP}, \mathsf{MK})$ and $\mathsf{KeyGen}_{\mathsf{Clnt}}(\Sigma, \mathsf{MK})$. A data owner encrypts files with a dummy policy Λ_0 via running $\mathsf{Encrypt}(\mathsf{PP}, \mathsf{Doc}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda_0)$ and later generates an actual authorization policy (or switch an authorization policy) via running $\mathsf{TKGen}(\mathsf{PP}, \mathsf{sk}_{\mathsf{Clnt}}, \Lambda, \Lambda') \to (\mathsf{tk}, \pi)$. The cloud transforms encrypted files into that encrypted under a newly updated authorization, via running $\mathsf{AuzAdp}(\mathsf{PP}, \mathsf{EDoc}, \mathsf{sk}_{\mathsf{Ser}}, \mathsf{tk}, \pi)$. To search the data owner's encrypted documents in the cloud, a data user issues a file query request Q via running $\mathsf{TrapGen}(\mathsf{sk}_{\mathsf{Clnt}}, Q) \to \mathsf{Token}$ and sends Token to the cloud. The cloud search files via running $\mathsf{Search}(\mathsf{Token}) \to R$ according to Token. Finally, the data user retrieves the encrypted files with $\{ind||K_{ind}\}$ decrypted from R.

4.2 Technique Overview

1) **Achieving direct implementation of fine-grained search authorization.** We consider each attribute att_i of a client as a point $(x_i, y_i) = (g^{\gamma H_2(\mathsf{att}_i)}, g^{\gamma H_3(\mathsf{att}_i)})$ of a Lagrange interpolation polynomial $\Psi_k(x)$ (c.f. Definition 2) and compute a product of coefficients $\Delta = \prod_{i=0}^{k-1} \Delta_i$; and thus produce a secret key $\mathsf{sk}_{\mathsf{Clnt}} = (\gamma, g^{\alpha r}, \Delta^\alpha)$, where α, γ is the master secret key and r is a randomness. For any attribute att'_i that required in an "AND"-gate authorization Λ, we re-compute a product of coefficients $\Delta'_i = \prod_{i=0}^{k-1} \Delta'_i$ of $\Psi_k(x)$. Hence, the client with the same attributes as required in Λ can share a same product of coefficients ($\Delta = \Delta'$), and finally recover the master secret key α.
2) **Enabling multi-client conjunctive/boolean queries under owner-enforced authorization.** We revisit Cash et al.'s SSE [20] and introduce an owner-enforced authorization $\Lambda = (\mathsf{att}_1 \wedge \mathsf{att}_2 \wedge \cdots)$ into file encryption algorithm, where a new key generation algorithm $\mathsf{KeyGen}_{\mathsf{Clnt}}$ is introduced for producing secret keys $\mathsf{sk}_{\mathsf{Clnt}} = (\gamma, g^{\alpha r}, \Delta^\alpha)$ for each client. For a data user's conjunctive query, its search token Token relates to attributes $\Sigma = (\mathsf{att}_1, \mathsf{att}_2, \cdots)$ and issued keywords $Q = (w_1 \wedge w_2 \wedge \cdots w_q)$. Thus, we have every $\mathsf{Trap}[c][j] = \Delta^{\alpha\gamma H(w_j)/(F(K_z, c||w_1))}$ in Token where w_1 is assumed as the s-term, thus the searching cost is still a sub-linear complexity $\mathcal{O}(c_{w_1})$ as [13], [15], [20]. Extending conjunctive AasBirch to deal with boolean queries is straightforwardly obtained and formally discussed in Section 7.
3) **Supporting search authorization with adaptable switching.** To enable adaptable authorization switching from Λ to any other Λ', we introduce two new algorithms inspired by [24], [25]: transformation key generation TKGen and authorization adaptable AuzAdp. By generating a transformation key tk that includes two sets of points $\{(x_i, y_i)\}, \{(x'_i, y'_i)\}$, a data owner delegates cloud to compute two products of coefficients Δ and Δ'. Thus, the cloud can transform encrypted files from $(ind||K_{ind} \cdot g^\gamma \Delta^\alpha, e_1, e_2, g^{\beta t} g^{F_p(K_y, H_4(\Lambda)||H_5(v)}, e_4)$ to $(ind||K_{ind} \cdot g^\gamma \Delta'^\alpha, e_1, e_2, g^{\beta t} g^{F_p(K_y, H_4(\Lambda')||H_5(v)}, e_4)$, while learns no privacy information about Λ, Λ' and file plaintext. In addition, we let data owners encrypt files under a dummy authorization policy $\Lambda_0 = \mathsf{att}_0 \wedge \mathsf{att}'_0$, and later switch it to an actual authorization $\Lambda_0 \Rightarrow \Lambda$ for further reducing file encryption cost. Besides, we introduce a message authentication code MAC in case a data user may forge a transformation key tk who has the same attributes set as a data owner. Concretely, we generate a tag $\pi = \mathsf{MAC.Mac}(K', \mathsf{tk})$ as a proof to be verified by running $\mathsf{MAC.Veri}(K', \mathsf{tk}, \pi)$.

4.3 AasBirch system with conjunctive queries

Assume the dynamic attribute universe that describe clients is $\mathcal{U} = \{0,1\}^*$. Let $\mathbb{G}, \mathbb{G}_T$ be groups of a prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where g is a generator of $\mathbb{G}$. Let F be a PRF with range in $\{0,1\}^*$, F_p be a PRF with range in $\mathbb{Z}_p$, $H_1, H_2, H_3 : \{0,1\}^* \to \mathbb{Z}_p$, $H_4 : \{0,1\}^* \to \{0,1\}^*$ and $H_5 : \mathbb{G} \to \{0,1\}^*$ are collision-resistant hash functions. Define a Lagrange interpolation polynomial function $\Psi_k(x) = \Delta_0 + \Delta_1 x + \cdots + \Delta_{k-1} x^{k-1} = \sum_{i=0}^{k-1} (\Delta_i x^i)$ and a secure message authentication code scheme $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Veri})$ that instantiated with an AES algorithm. Moreover, we employ a symmetric key encryption algorithm (e.g., AES) with a key K_{ind} to encrypt a document $\{(ind, W_{ind})\}$, and use a keyword dictionary δ defined in Definition 1 to manage a set of (w, c) by $c \leftarrow \mathsf{Get}(\delta, w)$ and $\mathsf{Update}(\delta, w, c)$. Concretely, AasBirch is formally described in Fig. 2.

4.4 Correctness Guarantee

4.4.1 The correctness of authorization switching

1) Switching a dummy policy Λ_0 to an actual policy Λ: Given an encryption tuple $\mathsf{EDB}[l_c] = (e_0, e_1, e_2, e_3, e_4)$ encrypted under Λ_0 and a transformation key

$$\mathsf{tk} = (\overbrace{\{(\perp, \perp)\}}^{\mathsf{tk}_0}, \overbrace{\{(x'_i, y'_i)\}_{i=0}^{k'-1}}^{\mathsf{tk}_1}, \overbrace{g^{\beta r} g^{\eta}}^{\mathsf{tk}_2}, \overbrace{g^{\beta r} g^{\eta'}}^{\mathsf{tk}_3}, \overbrace{g^r}^{\mathsf{tk}_4}, \overbrace{0}^{st}),$$

the cloud first verifies whether

$$\mathsf{tk}_2 / \mathsf{tk}_4^\beta \stackrel{?}{=} e_3 / e_4^\beta,$$

and then uses tk_0 and tk_1 to compute a set of b'_i and $B' = \prod_{i=0}^{k'-1} b'_i$ for Λ' according to Lagrange interpolation polynomial as follows:

$$\Psi'(x) = \sum_{i=0}^{k'-1} (y'_i \cdot \prod_{0 \le j \ne i \le k'-1} \frac{x - x'_j}{x'_i - x'_j}) = \sum_{i=0}^{k'-1} (b'_i x^i),$$

finally, it uses tk_3 and tk_4 to update $\mathsf{EDB}[l_c]$ and xtag' as follows:

$$\begin{aligned} e'_0 &= e_0 / e_4^\beta \cdot B'^\alpha \\ &= ind||K_{ind} \cdot g^\gamma g^{\beta r} / g^{r\beta} \cdot B'^\alpha \\ &= ind||K_{ind} \cdot g^\gamma B'^\alpha \\ e'_3 &= e_4^\beta \cdot \mathsf{tk}_3 / \mathsf{tk}_4^\beta = g^{r\beta} \cdot g^{\beta r} g^{\eta'} / g^{r\beta} = g^{\beta r} g^{\eta'} \\ \mathsf{xtag}' &= e(B'^\alpha, e_2) = e(B'^\alpha, g^{\gamma H(w) \cdot \mathsf{xind}}) \end{aligned}$$

- $\mathsf{Setup}(1^\kappa,\mathcal{U}) \rightarrow (\mathsf{PP},\mathsf{MK})$: The authority inputs a secure parameter κ and an attribute universe $\mathcal{U} = \{0,1\}^*$. It randomly selects three keys $K_x, K_z, K_y \leftarrow \{0,1\}^n$ for F_p, a key $K_l \leftarrow \{0,1\}^n$ for F and $\gamma, \alpha \leftarrow \mathbb{Z}_p$. Finally, it outputs a public parameter $\mathsf{PP} = \{e, g, g^\alpha, H, H_1, H_2, H_3, H_4, H_5, F, F_p\}$ and a master key $\mathsf{MK} = \{K_x, K_z, K_l, K_y, \gamma, \alpha\}$.
- $\mathsf{KeyGen_{Ser}}(\mathsf{PP},\mathsf{MK}) \rightarrow (\mathsf{pk_{Ser}},\mathsf{sk_{Ser}})$: The authority inputs $(\mathsf{PP},\mathsf{MK})$, and randomly selects $\beta \leftarrow \mathbb{Z}_p$. Finally, it outputs a secret key $\mathsf{sk_{Ser}} = (\alpha, \beta)$ and a public key $\mathsf{pk_{Ser}} = g^\beta$ for the cloud server.
- $\mathsf{KeyGen_{Clnt}}(\Sigma,\mathsf{MK}) \rightarrow \mathsf{sk_{Clnt}}$: The authority inputs MK and an attribute set $\Sigma \subseteq \mathcal{U}$ (a non-empty subset of $\mathcal{U}$) of a client. For any attribute $\mathsf{att}_i \in \Sigma$, it computes $x_i = g^{\gamma H_2(\mathsf{att}_i)}, y_i = g^{\gamma H_3(\mathsf{att}_i)}$, and computes $A = \prod_{i=0}^{k-1} a_i$ according to a Lagrange interpolation polynomial $\Psi_k(x) = \sum_{i=0}^{k-1} a_i x^i$, where k is the number of client's attributes. It randomly selects $r \leftarrow \mathbb{Z}_p$ and compute $v = g^{\alpha r}$, and runs $\mathsf{MAC.Gen}(\kappa)$ to produce a key K'. Finally, it outputs a secret key $\mathsf{sk_{Clnt}} = \{K_x, K_z, K_l, K_y, \gamma, v, A^\alpha, K'\}$ for a client.
- $\mathsf{Encrypt}(\mathsf{PP},\mathsf{Doc},\mathsf{sk_{Clnt}},\Lambda_0) \rightarrow \mathsf{EDoc}$: A client inputs PP, a secret key $\mathsf{sk_{Clnt}} = \{K_x, K_z, K_l, K_y, \gamma, v, A^\alpha, K'\}$, a document $\mathsf{Doc} = (ind, W_{ind})$ and a dummy authorization policy $\Lambda_0 = \{\mathsf{att}_0 \wedge \mathsf{att}_0'\}$, encrypts the original document by using a symmetric key algorithm (e.g., AES) with a secret key K_{ind}, and does the following:
  1) Compute an internal counter $c \leftarrow \mathsf{Get}(\delta, w), c \leftarrow c+1$ and $l_c \leftarrow F(K_l, c||w)$, $z \leftarrow F_p(K_z, c||w)$, $\eta \leftarrow F_p(K_y, H_4(\Lambda_0)||H_5(v))$, $\mathsf{xind} \leftarrow F_p(K_x, ind)$, and run $\mathsf{Update}(\delta, w, c)$ to update the counter of each keyword w to c.
  2) Randomly select $t \leftarrow \mathbb{Z}_p$, and compute encrypted tuples $e_0 \leftarrow ind||K_{ind} \cdot g^\gamma g^{\beta t}$, $e_1 \leftarrow g^{z \cdot \mathsf{xind}}$, $e_2 \leftarrow g^{\gamma H_1(w) \cdot \mathsf{xind}}$, $e_3 \leftarrow g^{\beta t} \cdot g^\eta$, $e_4 \leftarrow g^t$. Set each item in $\mathsf{EDB}[l_c] = (e_0, e_1, e_2, e_3, e_4)$, and append $\mathsf{xtag} \leftarrow e(g^\eta, g^{\gamma H_1(w) \cdot \mathsf{xind}})$ to XSet.
  3) Compute $l_d = H_4(g^\eta)$ and set $\mathsf{EDoc}[l_d] = (\mathsf{EDB}, \mathsf{XSet})$.

  Here, we remark that the xtag encrypted under a dummy policy Λ_0 is later transformed to xtag' under a new actual policy Λ' in $\mathsf{AuzAdp}(\cdot)$.
- $\mathsf{TKGen}(\mathsf{PP},\mathsf{sk_{Clnt}},\Lambda,\Lambda') \rightarrow (\mathsf{tk},\pi)$: A data owner inputs PP, secret key $\mathsf{sk_{Clnt}}$, an authorization Λ and an updated authorization policy $\Lambda'(\neq \Lambda)$, and does the following:
  1) Compute each $(x_i, y_i) = (g^{\gamma H_2(\mathsf{att}_i)}, g^{\gamma H_3(\mathsf{att}_i)})$ for any attribute $\mathsf{att}_i \in \Lambda$, and compute each $(x_i', y_i') = (g^{\gamma H_2(\mathsf{att}_i')}, g^{\gamma H_3(\mathsf{att}_i')})$ for any attribute $\mathsf{att}_i' \in \Lambda'$. And compute $\eta \leftarrow F_p(K_y, H_4(\Lambda)||H_5(v))$ and $\eta' \leftarrow F_p(K_y, H_4(\Lambda')||H_5(v))$.
  2) Randomly select $r \leftarrow \mathbb{Z}_p$, and set a transformation key $\mathsf{tk} = (\{(x_i, y_i)\}_{i=0}^{k-1}, \{(x_i', y_i')\}_{i=0}^{k'-1}, g^{\beta r} g^\eta, g^{\beta r} g^{\eta'}, g^r, \mathsf{st})$, where $\mathsf{st} = 0$ if $\Lambda = \Lambda_0$ (i.e., the old policy Λ is a dummy policy Λ_0), else set $\mathsf{st} = 1$. And it generates a proof $\pi = \mathsf{MAC.Mac}(K', \mathsf{tk})$ for the transformation key tk and sends (tk, π) to cloud.
- $\mathsf{AuzAdp}(\mathsf{PP},\mathsf{sk_{Ser}},\mathsf{tk},\pi) \rightarrow \mathsf{EDoc}'$: The cloud inputs $\mathsf{PP}, \mathsf{sk_{Ser}}$ and a transformation key tk and a proof π. By verifying the validity of (tk, π) by $\mathsf{MAC.Veri}(K', \mathsf{tk}, \pi)$, it does the following
  1) Compute $B = \prod_{i=0}^{k-1} b_i$ according to $\Psi_k(x) = \sum_{i=0}^{k-1} b_i x^i$ based on $\{(x_i, y_i)\}_{i=1}^{k-1}$, and compute $B' = \prod_{i=0}^{k'-1} b_i'$ according to $\Psi_{k'}(x) = \sum_{i=0}^{k'-1} b_i' x^i$ based on $\{(x_i', y_i')\}_{i=1}^{k'-1}$. And compute $l_d = H_4(g^\eta)$ if $\mathsf{st} = 0$, else $l_d = H_4(B^\alpha)$ and $l_d' = H_4(B'^\alpha)$ according to B and B', and later locate $\mathsf{EDoc}[l_d] = (\mathsf{EDB}, \mathsf{XSet})$.
  2) For each tuple $(e_0, e_1, e_2, e_3, e_4) \in \mathsf{EDB}$, if g^η not equals to e_3/e_4^β, it terminates. Otherwise, it does the following:
     a) and if $\mathsf{st} = 0$, convert $e_0 = ind||K_{ind} \cdot g^\gamma g^{\beta t}$ to $e_0 = ind||K_{ind} \cdot g^\gamma B'^\alpha$.
     b) and if $\mathsf{st} = 1$, convert $e_0 = ind||K_{ind} \cdot g^\gamma B^\alpha$ to $e_0 = ind||K_{ind} \cdot g^\gamma B'^\alpha$.
  3) For each tuple $\mathsf{xtag} \in \mathsf{XSet}$, convert xtag to $\mathsf{xtag}' = e(B'^\alpha, g^{\gamma H_1(w) \cdot \mathsf{xind}})$ (where $g^{\gamma H_1(w) \cdot \mathsf{xind}}$ comes from e_2).
  4) Replace a modified tuple $(e_0', e_1, e_2, e_3', e_4)$ and xtag' to $\mathsf{EDoc}[l_d']$, where $e_3' = g^{\beta t} \cdot g^{\eta'}$.
- $\mathsf{TrapGen}(\mathsf{sk_{Clnt}}, Q) \rightarrow \mathsf{Token}$: For a conjunctive query $Q = (w_1 \wedge w_2 \wedge \cdots \wedge w_q)$ where w_1 is the least frequent term (s-term) in Q, a client inputs $\mathsf{sk_{Clnt}}$ and computes $l_d \leftarrow H_4(A^\alpha)$, $l_c \leftarrow F(K_l, c||w_1)$, $z_c \leftarrow F(K_z, c||w_1)$, $\mathsf{Trap}[c][j] = A^{\alpha\gamma H(w_j)/z_c}$, for $j \in [q], c = 1, 2, \cdots$. Eventually, it sends $\mathsf{Token}[c] = (l_d, l_c, \mathsf{Trap})$ where $\mathsf{Trap}[c] = \{\mathsf{Trap}[c][j]\}_{j \in [q]}$ to the cloud for $c = 1, 2, \cdots$.
- $\mathsf{Search}(\mathsf{Token}) \rightarrow R$ With a search token Token from a client, the cloud initializes an empty set R as a searching result for $c = 1, 2, \cdots$, and retrieves $(e_0, e_1, e_2, e_3, e_4) \leftarrow \mathsf{EDB}[l_c]$, where $\mathsf{EDB} \leftarrow \mathsf{EDoc}[l_d]$. By checking if $e(A^{\alpha\gamma H(w_j)/z_c}, e_1) \in$ XSet for all $j \in [q]$ (where $A^{\alpha\gamma H(w_j)/z_c}$ comes from $\mathsf{Trap}[c][j]$), then it adds e_0 to the set R for all $j \in [q]$.
- $\mathsf{Retrieve}(R) \rightarrow Doc$ A data user decrypts each encrypted ind from $e_0 = ind||K_{ind} \cdot g^\gamma A^\alpha$ from the received R, and thus gets the encrypted files with inds. Finally, it decrypts the encrypted files with K_{ind}.

Fig. 2. Formal description of the AasBirch system that deals with conjunctive queries

2) Switching an old policy Λ to a new policy Λ': Given an encryption tuple $\mathsf{EDB}[l_c] = (e_0, e_1, e_2, e_3, e_4)$ encrypted under Λ, a transformation key

$$\mathsf{tk} = (\overbrace{\{(x_i, y_i)\}}^{\mathsf{tk}_0}, \overbrace{\{(x'_i, y'_i)\}_{i=0}^{k'-1}}^{\mathsf{tk}_1}, \overbrace{g^{\beta r} g^{\eta}}^{\mathsf{tk}_2}, \overbrace{g^{\beta r} g^{\eta'}}^{\mathsf{tk}_3}, \overbrace{g^{r}}^{\mathsf{tk}_4}, \overbrace{1}^{st}),$$

and a new policy Λ', the cloud first verifies whether

$$\mathsf{tk}_2/\mathsf{tk}_4^{\beta} \stackrel{?}{=} e_3/e_4^{\beta},$$

and then uses tk_0 and tk_1 to compute a set of b_i, b'_i and $B = \prod_{i=0}^{k'-1} b_i, B' = \prod_{i=0}^{k'-1} b'_i$ for Λ, Λ' according to Lagrange interpolation polynomial, finally, it uses tk_3 and tk_4 to update $\mathsf{EDB}[l_c]$ and xtag' as follows:

$$\begin{aligned} e'_0 &= e_0/B^{\alpha} \cdot (B')^{\alpha} \\ &= ind||K_{ind} \cdot g^{\gamma} B^{\alpha}/B^{\alpha} \cdot B'^{\alpha} \\ &= ind||K_{ind} \cdot g^{\gamma} B'^{\alpha} \\ e'_3 &= e_4^{\beta} \cdot \mathsf{tk}_3/\mathsf{tk}_4^{\beta} = g^{t\beta} \cdot g^{\beta r} g^{\eta'}/g^{r\beta} = g^{\beta t} g^{\eta'} \\ \mathsf{xtag}' &= (B'^{\alpha}, e_2) = e(B'^{\alpha}, g^{\gamma H(w) \cdot \mathsf{xind}}). \end{aligned}$$

4.4.2 The correctness of file searching

Given a search token

$$\mathsf{Token}[c] = (\overbrace{H_4(A^{\alpha})}^{l_d}, \overbrace{F(K_l, c||w)}^{l_c}, \overbrace{\{A^{\alpha\gamma H(w_j)/z_c}\}_{j=1}^{q}}^{\mathsf{Trap}})$$

and an encrypted document

$$\mathsf{EDoc}[l_d] := (\overbrace{\{e_0, e_1, e_2, e_3, e_4\}}^{\mathsf{EDB}}, \overbrace{\{\mathsf{xtag} = e(A^{\alpha}, e_2)\}}^{\mathsf{XSet}}),$$

the cloud first uses l_d to locate $\mathsf{EDoc}[l_d]$ and get $(\mathsf{EDB}, \mathsf{XSet})$, and also uses l_c to locate the encryption tuple $(e_0, e_1, e_2, e_3, e_4)$ from $\mathsf{EDB}[l_c]$, and finally uses $\mathsf{Token}[c]$ to search over $\mathsf{EDoc}[l_d]$ as follows:

$$\begin{aligned} e(\mathsf{Trap}[c][[j], e_1) &= e(A^{\alpha\gamma H(w_j)/z_c}, g^{z_c \cdot \mathsf{xind}}) \\ &= e(A^{\alpha\gamma H(w_j)}, g^{\mathsf{xind}}) \\ &= e(A^{\alpha}, g^{\gamma H(w_j) \cdot \mathsf{xind}}) = \mathsf{xtag}. \end{aligned}$$

4.5 Discussion and Extension

4.5.1 Fine-grained Authorization towards Multiple Clients

In AasBirth, every document is associated with a set of keywords, and the data owner and client are described by a set of attributes. We note that the fine-grained attribute-based authorization configuration and switching are designed for limiting the access and search permission of multiple clients. Nevertheless, the fine-gained authorization switching is not applicable for the situation of document $\mathsf{Doc} = (ind, W_{ind})$. Since the authorization switching for the document implies the authorization switching for the underlying encrypted keywords (i.e., $(\{\mathsf{EDB}[l]\}, \{\mathsf{xtag}\})$) in the document, this may unfortunately leak the connection privacy between $(\{\mathsf{EDB}[l]\}, \{\mathsf{xtag}\})$ and (ind, W_{ind}) to the cloud.

Technically speaking, a document $\mathsf{Doc} = (ind, W_{ind})$ is encrypted as $\mathsf{EDoc}[l_d] = (\{\mathsf{EDB}[l_i]\}_{i=1}^{|W_{ind}|}, \{\mathsf{xtag}_i\}_{i=1}^{|W_{ind}|})$, where $\{\mathsf{EDB}[l_i]\}$ and $\{\mathsf{xtag}_i\}$ are separately linked with a document. In this way, a data owner can switch all encryption tuples (i.e., $\mathsf{EDoc}[l_d] = (\{\mathsf{EDB}[l_i]\}_{i=1}^{|W_{ind}|}, \{\mathsf{xtag}_i\}_{i=1}^{|W_{ind}|})$) under from old authorization policy Λ to new Λ' by just inputting several keywords (i.e., a subset of W_{ind}) and a policy switching pair (Λ, Λ'). Hence, the cloud obtains the knowledge that which $\mathsf{EDB}[l]$ or xtag associates with a document.

4.5.2 Enhanced AasBirch system with boolean queries

Similar to [20], we show how to extend conjunctive queries "$w_1 \wedge w_2 \wedge \cdots \wedge w_q$" to boolean queries "$w_1 \wedge \psi(w_2, \cdots, w_q)$" for a set of keywords $(w_1, w_2, \cdots, w_q)$. A data user computes l_c and $\mathsf{Trap}[c]$ and sends them with a boolean expression $\bar{\psi}$ to the cloud, where $\bar{\psi}$ is a copy of ψ except that the keywords are replaced by $(v_2, \cdots, v_q)$. Later, the cloud uses l_c to retrieve tuples $(e_0, e_1, e_2, e_3, e_4)$ using s-term keyword w_1, where the difference with conjunctive queries is the way to determine which tuples match $\bar{\psi}$. For each $(e_0, e_1, e_2, e_3, e_4) \leftarrow \mathsf{EDB}[l_c]$, the cloud sets $(v_2, \cdots, v_q)$ as

$$v_j = \begin{cases} 1 & \text{if } e(A^{\alpha\gamma H(w_j)/z_c}, e_1) \in \mathsf{XSet} \\ 0 & \text{otherwise} \end{cases}$$

where $j = 2, \cdots, q$. If $e(A^{\alpha\gamma H(w_j)/z_c}, e_1) \in \mathsf{XSet}$ and $\bar{\psi}$ holds, this implies that the tuple matches the query, then the cloud appends e_0 to the result set R. We remark that the searching cost for processing such boolean query is $\mathcal{O}(c_{w_1})$ where w_1 is the s-term in a query. The leakage profile description and analysis is consistent with that of conjunctive AasBirch, except for $\bar{\psi}$ is obtained by the cloud.

5 AasBirch: Security Analysis

Based on the introduced security model, we give a formal security analysis of (non-)adaptive adversarial cloud server and colluded adversarial clients for the AasBirch system.

5.1 Security Analysis

We present the following three theorems to sketch a security analysis for the AasBirch system.

Theorem 1. Assume the employed PRFs and hash functions and MAC in AasBirch are secure, and DL and DDH assumptions hold in $\mathbb{G}$ and $\mathbb{G}_T$, hence *our scheme is $\mathcal{L}$-semantically secure* against non-adaptive attacks under the introduced security model in Section 3.3.

Proof 1. The non-adaptive attack indicates that an adversary submits two completed lists $\mathbf{d}$ and $\mathbf{q}$ as inputs of a leakage function $\mathcal{L}$, thus we construct a simulator $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\kappa)$ (formally presented in Algorithm. 1) that has the same distribution as the real game $\mathsf{Real}_{\mathcal{A}}^{\Pi}(\kappa)$. Formally speaking, given the leakage function $\mathcal{L}(\mathbf{d}, \mathbf{q}) = (\mathsf{op}, N, \bar{\mathsf{s}}, \mathsf{RP}, \mathsf{SRP}, \mathsf{IP}, \mathsf{dRP}, \mathsf{xt})$, the simulator firstly computes a restricted equality pattern $\bar{x}$ to describe which xterms are "known" to be equal by the cloud. For two queries

$$\mathbf{q}[t_1] = ((\mathbf{s}[t_1], \mathbf{x}[t_1, \cdot]), \mathbf{id}[t_1])$$

and

$$\mathbf{q}[t_2] = ((\mathbf{s}[t_2], \mathbf{x}[t_2, \cdot]), \mathbf{id}[t_2]),$$

there exists a ind such that $ind \in \mathsf{DB}[\mathbf{s}[t_1]] \cup \mathsf{DB}[\mathbf{s}[t_2]]$. The adversary is able to know the xterms $\mathbf{x}[t_1, \alpha]$ and $\mathbf{x}[t_2, \beta]$ that are equal from "$e(A^{\alpha\gamma H(w_j)/z_c}, e_1)$".

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3227650

Algorithm 1 Security Proof: Simulator Description

function Initialize($\mathcal{L}(\mathbf{d},\mathbf{q})$)
$\gamma \xleftarrow{\$} \mathbb{Z}_p$
for each $h \in \mathbf{s}'$ **do**
c_h=0
end for
for each $w \in \mathbf{x}', ind \in$ RP $\cup$ IP **do**
$H_2[w, ind] = y \xleftarrow{\$} \mathbb{Z}_p$
$H_3[w, ind] = e(g,g)^{H_2[w,ind]}$
end for
for each $w \in \mathbf{x}', ind \in$ RP $\cup$ IP,$\Lambda_{ind} \in \mathbf{\Lambda}$ **do**
if $\Lambda_{ind} = \bigwedge_{\mathsf{att}_i \in I} \mathsf{att}_i$ **then**
if $H_7[\mathsf{att}_i]$ exists **then**
$x_i, y_i \leftarrow H_7[\mathsf{att}_i]$
else $x_i \leftarrow \mathbb{G}$, $y_i \leftarrow \mathbb{G}$, $H_7[\mathsf{att}_i] = x_i, y_i$
end if
$H_5[w, ind, \Lambda_{ind}] = A$ calculated by Lagrange
end if
end for
$d = q = a = 1$
for $h = 1$ to $|\mathsf{op}|$ **do**
switch ($\mathsf{op}[h]$):
case encrypt:
$\mathbf{t}[h] \leftarrow$ Encrypt($\mathcal{L}(\mathbf{d},\mathbf{q})), d++$; break
case search:
$\mathbf{t}[h] =$ TranGen($\mathcal{L}(\mathbf{d},\mathbf{q})), q++$; break
end for
return t
end function
function Encrypt($\mathcal{L}(\mathbf{d},\mathbf{q})$)
$h = 0$, Dup $\leftarrow \{\}$
for $\mathbf{s}'[q'] \in \{\mathbf{s}'[q] \cdots \mathbf{s}'[|\bar{\mathbf{s}}|]\}$, $\mathbf{s}'[q'] \notin$ Dup and dRP$[d][q'] == 1$ **do**
$c_{\mathbf{s}'[q']}++$
$l \xleftarrow{\$} \{0,1\}^*$, $l[\mathbf{s}[q'], c_{\mathbf{s}'[q']}] = l, e_0 = g^{\gamma} H_5[\Lambda[d]])$
$y' \xleftarrow{\$} \mathbb{Z}_p, H_1[\mathbf{s}'[q'], c_{\mathbf{s}'[q']}] = y'$
$e_1 \leftarrow g_2^{H_1\left[\mathbf{s}'[q'], c_{\mathbf{s}'[q']}\right]}$, $e_2 \xleftarrow{\$} \mathbb{G}, e_3 \xleftarrow{\$} \mathbb{G}, e_4 \xleftarrow{\$} \mathbb{G}$
EDB$[l] = (e_0, e_1, e_2, e_3, e_4)$
XSet $\leftarrow$ XSetSetup($\mathcal{L}(\mathbf{d},\mathbf{q})$)
EDoc$[H_8[\Lambda[d]]] \leftarrow$ EDoc$[H_8[\Lambda[d]]] \cup$ (EDB, XSet)
Dup $\leftarrow$ Dup $\cup\, \mathbf{s}'[q'], h++$
end for
for h to $N[d]$ **do**
$l \xleftarrow{\$} \{0,1\}^*$, $y \xleftarrow{\$} \mathbb{G}, e_0 = yH_5[\Lambda[d]])$
$e_1 \xleftarrow{\$} \mathbb{G}, e_2 \xleftarrow{\$} \mathbb{G}, e_3 \xleftarrow{\$} \mathbb{G}, e_4 \xleftarrow{\$} \mathbb{G}$
EDB$[l] = (e_0, e_1, e_2, e_3, e_4)$
XSet $\leftarrow$ XSetSetup($\mathcal{L}(\mathbf{d},\mathbf{q})$)
EDoc$[H_8[\Lambda[d]]] \leftarrow$ EDoc$[H_8[\Lambda[d]]] \cup$ (EDB, XSet)
end for
return EDoc
end function
function XSetSetup($\mathcal{L}(\mathbf{d},\mathbf{q})$)
XSet $\leftarrow \{\}, h = 0$
for $w = \mathbf{x}'[t \geq q, \alpha]$ and RP$[t, \alpha, d] \neq \emptyset$ **do**
$ind \leftarrow$ RP$[t, \alpha, d]$, xtag $\leftarrow H_3[w, ind]$
XSet $\leftarrow$ XSet $\cup$ xtag, $h++$
end for
for j to $N[d]$ **do**
xtag $\xleftarrow{\$} \mathbb{G}_T$, XSet $\leftarrow$ XSet $\cup$ xtag
end for
return XSet
end function
function TKGen($\mathcal{L}(\mathbf{d},\mathbf{q})$)
for $\mathsf{att}_i \in \Lambda$ **do**
if $H_7[\mathsf{att}_i]$ exists **then** $x_i, y_i \leftarrow H_7[\mathsf{att}_i]$
else $x_i \leftarrow \mathbb{G}$, $y_i \leftarrow \mathbb{G}$, $H_7[\mathsf{att}_i] = x_i, y_i$
end if
end for
for $\mathsf{att}'_i \in \Lambda'$ **do**
if $H_7[\mathsf{att}_i]$ exists **then** $x'_i, y'_i \leftarrow H_7[\mathsf{att}'_i]$
else $x'_i \leftarrow \mathbb{G}$, $y'_i \leftarrow \mathbb{G}$, $H_7[\mathsf{att}'_i] = x'_i, y'_i$
end if
end for
if $H_8[\Lambda]$ exists **then** $\eta = H_8[\Lambda]$
else $\eta \xleftarrow{\$} \mathbb{Z}_p, H_8[\Lambda] = \eta$
end if
if $H_8[\Lambda']$ exists **then** $\eta' = H_8[\Lambda']$
else $\eta' \xleftarrow{\$} \mathbb{Z}_p, H_8[\Lambda'] = \eta'$
end if
$h++$, $t \leftarrow \mathbb{Z}_p$
if $\Lambda = \Lambda_0$ **then** $\mathsf{st} = 0$
else $\mathsf{st} = 1$
end if
return $\mathsf{tk} = (\{(x_i, y_i)\}, \{(x'_i, y'_i)\}, g^{\beta' t} g^{\eta}, g^{\beta' t} g^{\eta'}, g^t, \mathsf{st})$
end function
function TranGen($\mathcal{L}(\mathbf{d},\mathbf{q})$)
$\mathbf{l} = \{l[\mathbf{s}'[q], h]\}_{h=1}^{c_{\mathbf{s}'[q]}}$, $(ind_1, \cdots, ind_{c_{\mathbf{s}'[q]}}) \leftarrow$ SRP[q]
for $\alpha \in [\mathsf{xt}[q]]$ **do**
$R \leftarrow$ RP$[q, \alpha] \cup_{q' \in [|\mathbf{s}'|], \beta \in [\mathsf{xt}[q']]}$ IP$[q, q', \alpha, \beta]$
for $c \in \left[c_{\mathbf{s}'[q]}\right]$ **do**
if $ind_c \in R$ **then**
$y \xleftarrow{\$} H_2[\mathbf{s}'[q'], c_{\mathbf{s}'[q']}]$
$A \leftarrow H_5[\mathbf{x}'[q, \alpha], ind_c, \mathbf{\Lambda}[q]]$
Trap$[c][\alpha] = A^{\frac{y}{H_1\left[\mathbf{s}'[q'], c_{\mathbf{s}'[q']}\right]}}$
else
if $\exists\, H_6[\mathbf{s}'[q], \mathbf{x}'[q, \alpha], c, \mathbf{id}[q]]$ **then**
Trap$[c][\alpha] = H_6[\mathbf{s}'[q], \mathbf{x}'[q, \alpha], c, \mathbf{\Lambda}[q]]$
else
Trap$[c][\alpha] \xleftarrow{\$} \mathbb{G}$,
$H_6[\mathbf{s}'[q], \mathbf{x}'[q, \alpha], c, \mathbf{\Lambda}[q]] =$ Trap$[c][\alpha]$
end if
end if
end for
end for
Token $\leftarrow$ ($\mathbf{l}$, Trap), Res $\leftarrow$ Search(Token)
ResInds $\leftarrow \cap$ RP$[q, \alpha]$ for $\alpha \in [\mathsf{xt}[q]]$
return (Token, Res, ResInds)
end function

Similarly, the leakage IP is also formulated. Therefore, we can define $\bar{\mathbf{x}}[t, \alpha]$ to record $\bar{\mathbf{x}}[t_1, \alpha] = \bar{\mathbf{x}}[t_2, \beta]$ if $\mathsf{IP}[t_1, t_2, \alpha, \beta] \neq \varnothing$, and thus have

$$\bar{\mathbf{x}}[t_1, \alpha] = \bar{\mathbf{x}}[t_2, \beta] \Rightarrow \mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]$$

and

$$(\mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta]) \wedge (\mathsf{DB}[s[t_1]] \cap \mathsf{DB}[s[t_2]] \neq \varnothing) \\ \wedge (\mathbf{id}[t_1] = \mathbf{id}[t_2]) \Rightarrow \bar{\mathbf{x}}[t_1, \alpha] = \bar{\mathbf{x}}[t_2, \beta].$$

In the simulator, we introduce hash tables H_2 and H_3, where H_3 is used to generate xtag and H_2 is used to generate Token. In addition, we also introduce hash table H_7 and H_8 to record the attribute points and the location that will be access later.

When it computes the search results ResInds, the simulator directly pulls the values from RP, and thus makes the final output of queries the same as that in real game. Since, we have $\Pr[\mathsf{Real}^{\Pi}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathsf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\kappa) = 1] \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A},\mathbb{G}}(\kappa) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},F_p}(\kappa) + \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{A},\mathbb{G}}(\kappa) + \mathsf{Adv}^{\mathsf{Unforgeable}}_{\mathcal{A},\mathsf{MAC}}(\kappa)$.

Theorem 2. Assume the employed PRFs, hash functions and MAC in AasBirch are secure, and DL and DDH assumptions hold in $\mathbb{G}$ and $\mathbb{G}_T$, hence *our scheme is secure against collusion attacks launched by adversarial clients* under the introduced security model in Section 3.3.

Proof 2. Firstly, we define the game sequence for unforgeability of search token, and give an adversary's winning advantage analysis between neighbor games:

- Game_0 : The game is exactly the same as the real scheme that defined in Fig. 2. Thus, we have
$$\Pr[\mathsf{CollUFExp}^{\Pi}_{\mathcal{A}} = 1] = \Pr[\mathsf{Game}_0 = 1].$$
- Game_1 : In the game, we randomly choose r from $\mathbb{Z}_p$, and replace a secret key $A^{\alpha} = g^a$ with g^r. If $\mathcal{A}$ can distinguish Game_1 from Game_0, then we can build a simulator $\mathcal{B}_1$ to break the DL assumption. Thus, we have
$$\Pr[\mathsf{Game}_0 = 1] - \Pr[\mathsf{Game}_1 = 1] \leq \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}_1}(\kappa).$$
- Game_2 : In the game, we randomly choose r from $\mathbb{Z}_p$, set $A^{\gamma} = g^a$, $A^{\alpha H(w_j)/z_c} = g^b$ and replace a trapdoor $A^{\alpha\gamma H(w_j)/z_c} = g^{ab}$ with g^r. If $\mathcal{A}$ can distinguish Game_2 from Game_1, then we can build a simulator $\mathcal{B}_2$ to break the DDH assumption. Thus, we have
$$\Pr[\mathsf{Game}_1 = 1] - \Pr[\mathsf{Game}_2 = 1] \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}_2}(\kappa).$$
- Game_3 : In the game, we replace the keyed PRFs (i.e., F_p with K_x, K_l, K_z) with a truly random function. If $\mathcal{A}$ can distinguish Game_3 from Game_2, then we can build a simulator $\mathcal{B}_3$ to distinguish the keyed PRFs from a truly random function. Thus, we have
$$\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1] \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}_3,F_p}(\kappa).$$
- Game_4 : In the game, we generate proof π for every transformation key tk, i.e., $\pi = \mathsf{MAC.Sign}(K', \mathsf{tk})$. If $\mathcal{A}$ can distinguish Game_4 from Game_3, then we can build a simulator $\mathcal{B}_4$ to break the IND-CPA security of MAC. Thus, we have
$$\Pr[\mathsf{Game}_3 = 1] - \Pr[\mathsf{Game}_4 = 1] \leq \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathcal{B}_4,\mathsf{MAC}}(\kappa).$$

Finally, we may conclude that the advantage of any adversary forging search token and transformation key is negligible, since we have $\Pr[\mathsf{CollUFExp}^{\Pi}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathsf{Game}_4 = 1] \leq \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}_1}(\kappa) + \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}_2}(\kappa) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}_3,F_p}(\kappa) + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\mathcal{B}_4,\mathsf{MAC}}(\kappa)$.

6 AasBirch: Experiment and Analysis

To illustrate practical performance, we conduct extensive experiments for state-of-the-art solutions and AasBirch in real cloud environment, and show performance analysis.

6.1 Theoretical Analysis

TABLE 4
Size of Public Parameter, secret key and encrypted files index.

	[17]	[15]	AasBirch
PP	$5\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert$	$2\lvert\mathbb{G}\rvert + (2n+3)\lvert\mathbb{Z}_p\rvert + \mathsf{MPK}_{\mathsf{ABE}}$	$2\lvert\mathbb{G}\rvert + 6\lvert\mathbb{Z}_p\rvert$
sk	$(2n+1)\lvert\mathbb{G}\rvert$	$(n+1)\lvert\mathbb{G}\rvert + 3\lvert\mathbb{Z}_p\rvert + \mathsf{sk}_{\mathsf{ABE}}$	$3\lvert\mathbb{G}\rvert + 4\lvert\mathbb{Z}_p\rvert$
EDB	$\mathcal{O}(n_k \cdot \lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(\lvert\Sigma\rvert \cdot \lvert\mathsf{DB}\rvert)$	$\mathcal{O}(\lvert\mathsf{DB}\rvert)$

In the table, we let "$|\mathbb{G}|$" denote an element of $\mathbb{G}$, "$|\mathbb{Z}_p|$" denote an element of $\mathbb{Z}_p$, "n" denote the base of attribute universe, "n_k" denote the number of keywords of a file and "$|\Sigma|$" denote the number of attributes of an encrypted file, "$|\mathsf{DB}|$" denote the number of files in DB ; "$\mathsf{MPK}_{\mathsf{ABE}}$, $\mathsf{sk}_{\mathsf{ABE}}$" denote "master public key, secret key" size of ABE.

In Table 4, we show a more detailed size of public parameter PP, secret key sk and encrypted files index EDB to clarify the cost comparison listed in Table 1, where our AasBirch achieves constant-size PP, sk and EDB while that of state-of-the-art work [15], [17] are unfortunately related to a parameterized $n_k, |\Sigma|$ and underlying employed ABE schemes. Note that [15] should prepare $2n$ variables for n attributes, which is considered to be a static small attribute-universe for user description. Similarly, there are k variables produced for secret key of a client that described with k attributes in [15], [17]. However, AasBirch employs a large-universe description for each client and inherently saves large parameter size instead of preparing a set of public tuples.

Furthermore, AasBirch allows data owners to switch authorization from one attribute-based level policy to another one, while [17] only supports a specified authorization updating from one user-level to another one. As shown in Table 5, the cost for switching authorization in AasBirch is comparable with 5 despite more flexible authorization is achieved.

TABLE 5
Performance analysis of authorization updating.

Measurement	[15]	AasBirch
Time Cost	$\mathcal{O}(\lvert\Sigma\rvert)$	$\mathcal{O}(\lvert\Sigma\rvert)$
Parameter Size	$2\lvert\Sigma\rvert \cdot \lvert\mathbb{G}\rvert$	$(4\lvert\Sigma\rvert + 3)\lvert\mathbb{G}\rvert + \lvert\pi\rvert$
Fine-grained Authorization	✗	✓

In the table, we let "$|\Sigma|$" denote the number of attributes of an encrypted file, "$|\pi|$" denote the size of a message authentication code.

6.2 Experiments in Real Cloud Environment

We use HUAWEI Cloud to conduct experiments for the work [15], [17] and AasBirch, where they are implemented with 4,000 lines of Python 3 codes. That is, the separately configured environment of cloud and clients are:

-Cloud: on Ubuntu 18.04 system with an Intel(R) Xeon(R) CPU E5-2680 v4 @2.40GHz and 8.00 GB RAM;
-Clients: on Ubuntu 18.04 system with an Intel(R) Core(TM) i5-6200U CPU @2.30GHz and 4.00 GB RAM.

For [15] and AasBirch implemented by Pypbc 0.2 library, we choose AES-CBC module (key is 256 bits and Initialization Vector is 128 bits) to encrypt files, SHA-256 as employed hash functions and curve $y^2 = x^3 + x$ for Type-A pairings (q-bits=512 and r-bits=160). Moreover, we employ PyCharm to call BSW ciphertext-policy ABE [28] for realizing [17]'s dynamic attribute-based keyword search; and utilize HMAC-SHA256 from wolfcrypt to instantiate the employed MAC scheme. In the experiment, the attributes number in an attribute set Σ or authorization policy Λ are both assumed to range from 1 to 50.

Dataset. For a respective real-world Email dataset Enron [26], we randomly choose "MAILDIR/SOLBERG-G" that includes 1,081 files as a partially testing dataset of Enron. In addition, we extract a set of keywords from the context of each email by PyTextRank 3.2.2 [29]. Accordingly, the number of keywords (#KWD) in a searching query varies from 1 to 50.

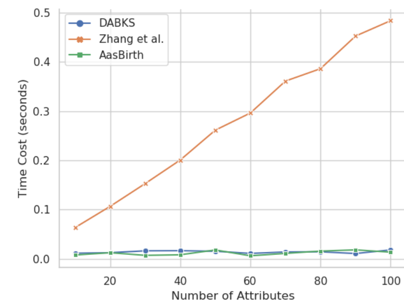
6.3 Performance Analysis

We examine the performance from *system initialization*, *file encryption*, *policy switching* and *file retrieval* phases.

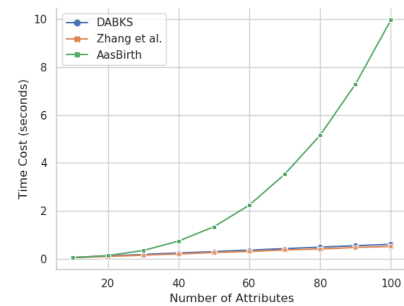
6.3.1 *System Initialization*

The system initialization includes Setup and KeyGen algorithms, where we show a time cost comparison about setup and KeyGen between DABKS [17], Zhang et al.'s SE [15] and AasBirth in Fig. 3(a) and Fig 3(b). Along with the growth of attribute number $|\Sigma|$, the time of setup in [15] grows and reaches 0.5 seconds when $|\Sigma|$=100, while AasBirch maintains a fixed 0.01 seconds time cost and runs faster than [15]. Nevertheless, the key generation time cost of AasBirch is slower than [15], [17], we say the cost can be accepted when $|\Sigma| = 50$ as depicted in Fig. 3(b). Moreover, an entity is usually described with approximately 50 attributes in practice.

Moreover, Table. 6 shows public parameter PP and secret key sk size comparison. When $|\Sigma|$=50, [15] needs 63.5 KB for storing PP, while [17] and AasBirch only needs 0.88 KB and 0.49 KB constant-size storage cost respectively. In particular, AasBirch reaches 250× smaller than [15] when $|\Sigma|$=100. Similarly, the storage cost of sk in AasBirch still holds a constant-size 0.36 KB. When $|\Sigma|$=100, AasBirch achieves roughly 74× and 200× smaller than [17] and [15] respectively. Accordingly, we may conclude that AasBirth runs faster in setup but slightly slower in KeyGen, and greatly saves PP and sk storage cost.



(a) Time cost of setup



(b) Time cost of key generation

Fig. 3. Time cost of system initialization phase

TABLE 6
Size of Public Parameter and secret key

$\|\Sigma\|$	Public Parameter (KB)			Secret Key (KB)		
	[17]	[15]	AasBirch	[17]	[15]	AasBirch
10		14.5		2.8	10.0	
20		26.8		5.4	19.4	
30		39.0		8.0	28.7	
40		51.3		10.7	38.1	
50	0.88	63.5	0.49	13.3	47.5	0.36
60		75.8		15.9	56.8	
70		88.0		18.9	66.2	
80		100.3		21.2	75.6	
90		112.5		23.9	84.9	
100		124.8		26.5	71.4	

6.3.2 *File Encryption*

To evaluate performance of encryption Encrypt algorithm, we first note that the file encryption in AasBirch should essentially include Encrypt and TKGen algorithms. This is because data owners run Encrypt under a dummy authorization policy Λ_0 and need to produce an actual authorization Λ via TKGen for the cloud. Table. 7 shows an efficiency comparison, where AasBirth encrypts keywords and files together with less than 650 seconds where the attribute number $|\Sigma|$ ranges from 10 to 100. In particular, it achieves 16× and 14× faster when $|\Sigma| = 50$ respectively, compared to DABKS [17] and Zhang et al.'s system [15]. With $|\Sigma|$ increasing, the encryption cost in AasBirch slightly rises and accordingly achieves even 23× (resp. 20×) faster when $|\Sigma| = 100$ than [17] (resp. [15]). In particular, we observe that the consumed time cost of Encrypt in our AasBirch is actually fixed to 360 seconds since the algorithm only deal with a dummy authorization policy Λ_0.

TABLE 7
Time cost of file encryption (minutes).

$\lvert\Sigma\rvert$	10	20	30	40	50	60	70	80	90	100
[17]	37.9	62.1	86.9	111.9	135.7	158.8	183.2	206.8	234.1	257.2
[15]	31.9	52.4	73.2	93.7	114.3	134.9	156.0	176.5	197.4	216.8
AasBirch	6.8	7.2	7.6	8.1	8.5	8.9	9.3	9.8	10.3	10.6

Furthermore, encrypting indexes may produce WSET.DAT and EDB.DAT files, which are stored in the client side. As can be seen in Table 8, the storage cost of EDB.DAT for [15], [17] increase along with attribute number and requires 0.74 GB and 1.82 GB respectively when $|\Sigma| = 100$. However, AasBirch only requires 0.02 GB storage cost regardless of different $|\Sigma|$. For [15] and AasBirch, the storage of Wset are fixed to 229.8 KB and 234 KB respectively.

TABLE 8
Size of encrypted indexes EDB (GB).

$\lvert\Sigma\rvert$	10	20	30	40	50	60	70	80	90	100
[17]	0.08	0.16	0.23	0.30	0.38	0.45	0.52	0.60	0.67	0.74
[15]	0.22	0.40	0.58	0.76	0.93	1.11	1.29	1.47	1.64	1.82
AasBirch					0.028					

Hence, we may conclude that AasBirch greatly reduces time cost and storage cost of file encryption phase, since the cost-expensive encrypted file generation and transformation are efficiently and securely transferred to the cloud.

6.3.3 *Policy Switching*

The policy switching phase includes TKGen and AuzAdp algorithms. We manually vary the number of attributes in a switching authorization from 20 to 100. Concretely for our AasBirch, Fig. 4 shows the time cost of client generating a new authorization policy, which is bounded up to 0.75 seconds. Fig. 4 also shows time cost of cloud transforming encrypted files (EDB) under one authorization into another one, where it ranges from 60 seconds to 78 seconds. It can be concluded that: (i) the main overhead of policy switching

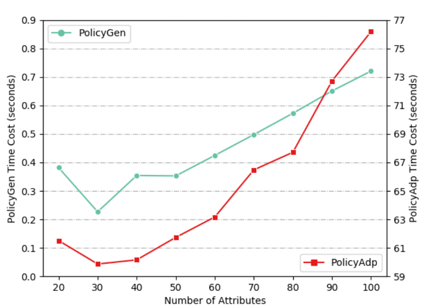


Fig. 4. Time cost of TKGen and AuzAdp algorithms in AasBirch

phase falls on the cloud server side, while the time cost for client side is lightweight; (ii) the time cost of TKGen and AuzAdp is related to assumed number of attributes in an authorization.

6.3.4 *File Retrieval*

The file retrieval phase includes TrapGen, Search and Retrieve algorithms.

To clarify the performance of TrapGen, we show a comparison for token generation cost between [15], [17] and AasBirth in Fig. 5. As can be seen in Table. 5, the time cost of [15], [17] increase with the increment of attribute number $|\Sigma|$, which is influenced by both the number of keywords (#KWD) in a query and the number of attribute set $|\Sigma|$ of a client. And [17] are rising faster while [15] brings about more overhead. Alternatively, AasBirch almost achieves a fixed cost for a same #KWD (as depicted in Fig. 6(a)), which is relatively related with #KWD and independent of $|\Sigma|$. In particular, AasBirch achieves 2$\times$ and 25$\times$ faster than [15], [17] respectively when #KWD=5 and $|\Sigma| = 50$, and even 80$\times$ faster than [15] when #KWD=10 and $|\Sigma| = 100$. For our AasBirch, we additionally measure search token storage cost (TOKEN.DAT) in Fig. 6(b) with different s-term, where it increases with WSet and produces large storage overhead with s-term increases. In particular, the token size is 60 KB When s-term=43 and WSet=10 (the worst case), which it only requires almost 1.2 seconds to be sent from client side to the cloud as shown in Fig. 6(c). In Fig. 6(c), we give detailed communication cost for sending a search token that relates to different s-term and Wset as Fig. 6(b), where the communication cost are all less than 1.2 seconds. Hence, AasBirch indeed shows practical efficiency for token generation, storage and client-to-cloud communication cost.

To illustrate the performance of Search, we show a comparison for searching cost between [15], [17] and AasBirth in Table. 9. The traditional dynamic ABKS [17] nearly consumes 0.2 hour for processing a single keyword search where #KWD=10, and rapidly increase with #KWD and $|\Sigma|$ rising. However, the searching cost in [15] is independent of #KWD but a little relatively related to $|\Sigma|$. Particularly for Σ=40 and #KWD=6, AasBirch outperforms 28,800$\times$ (resp. 34$\times$) faster than DABKS [17] (resp. [15]) by setting s-term=21. It can be deduced that the larger the Σ, #KWD and the smaller the s-term, the more cost savings by AasBirch will be obtained. For example, AasBirch may run 70,000$\times$ faster than known dynamic ABKS system [17] when Σ=100 and #KWD=10. For further observing the influence of s-term, Wset and $|\Sigma|$, we show searching time cost distribution in Fig. 7(a) and Fig. 7(b). As can be seen in Fig. 7(a), the searching time cost under different $|\Sigma|$ varies within a same range; while it is easily influenced by the value of s-term as shown in Fig. 7(b). In particular, searching time is 0.1 seconds When s-term=1 and #KWD=10; and is still no more than 0.4 seconds under s-term=10 and #KWD=10. Although the searching time cost grows along with an increment of s-term, it achieves high efficiency that less than 0.5 seconds. Hence, we can see that s-term has oblivious influence in searching time cost in AasBirch and [15], but AasBirch is actually highly efficient and independent of $|\Sigma|$ in AasBirch since no (semi-)black-box implementation of ABE is employed.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3227650

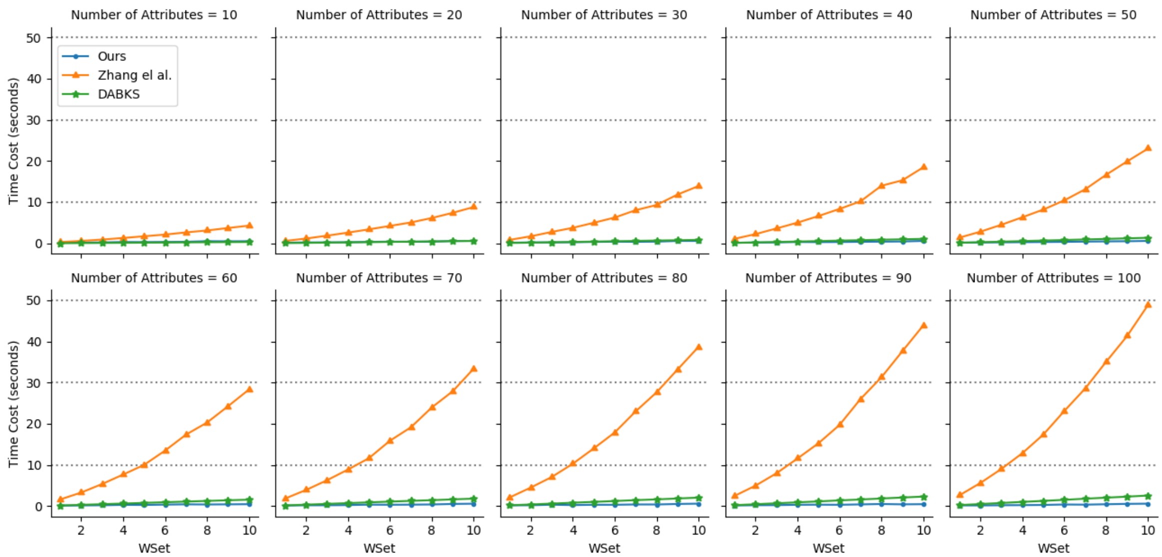
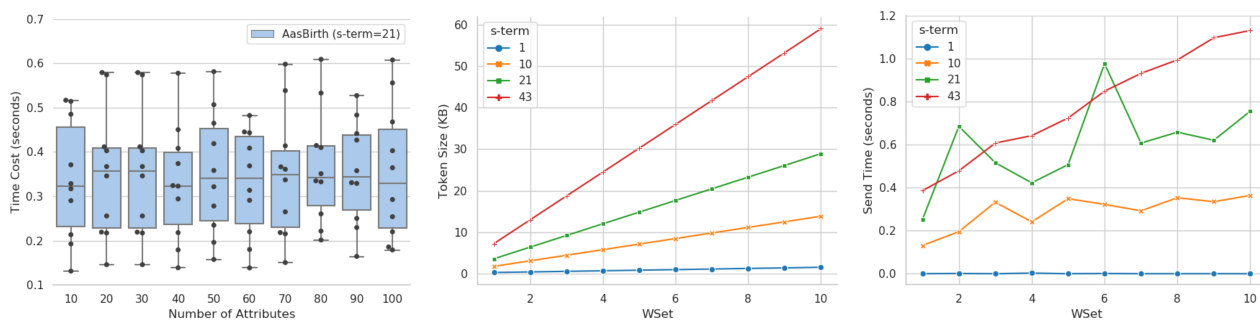


Fig. 5. Time cost of token generation of AasBirth



(a) Search token generation time cost (b) Search token size (c) Search token communication cost

Fig. 6. Time cost for token generation, token storage and client-to-cloud communication in AasBirch

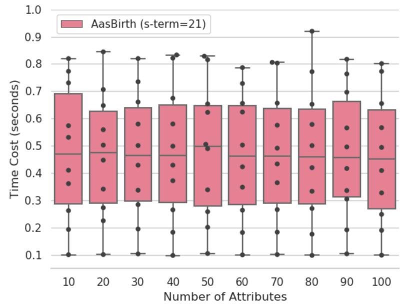
TABLE 9
Searching time cost comparison between AasBirch (#s-term=21) and state-of-the-art solutions.

$\|\Sigma\|$	#KWD=2			#KWD=3			#KWD=4			#KWD=5			#KWD=6			#KWD=7			#KWD=8			#KWD=9			#KWD=10		
	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours	[17]	[15]	Ours
10	0.40h	4.5s	0.19s	0.60h	6.7s	0.26s	0.80h	9.1s	0.36s	1.00h	11.5s	**0.41s**	1.20h	13.8s	0.53s	1.40h	16.2s	0.57s	1.60h	18.6s	0.73s	1.80h	20.8s	0.77s	2.00h	23.8s	0.82s
20	0.71h	5.1s	0.23s	1.07h	7.7s	0.27s	1.43h	10.2s	0.34s	1.78h	12.9s	0.45s	2.14h	15.6s	0.50s	2.51h	18.3s	**0.56s**	2.86h	21.4s	**0.65s**	3.23h	24.3s	**0.71s**	3.59h	27.4s	0.84s
30	1.03h	5.2s	0.19s	1.54h	7.9s	0.28s	2.05h	10.5s	0.34s	2.56h	13.4s	0.43s	3.08h	16.4s	0.50s	3.58h	19.6s	0.58s	4.11h	22.8s	0.66s	-	25.9s	0.73s	-	29.2s	0.82s
40	1.33h	5.3s	**0.18s**	2.00h	8.0s	0.27s	2.66h	10.8s	0.37s	3.33h	14.0s	0.43s	4.00h	17.0s	0.50s	-	20.5s	0.58s	-	24.0s	0.67s	-	27.8s	0.82s	-	32.0s	0.83s
50	1.67h	5.4s	0.20s	2.52h	8.3s	0.26s	3.36h	11.3s	0.34s	4.19h	14.5s	0.49s	-	17.9s	0.51s	-	21.3s	0.62s	-	25.1s	**0.65s**	-	29.0s	0.82s	-	33.0s	0.83s
60	1.95h	5.5s	**0.18s**	2.91h	8.5s	0.26s	3.89h	11.9s	0.35s	-	15.3s	0.42s	-	18.8s	0.50s	-	22.4s	0.62s	-	26.4s	0.66s	-	30.8s	0.73s	-	35.7s	**0.79s**
70	2.25h	5.5s	**0.18s**	3.37h	8.5s	0.27s	-	11.8s	0.36s	-	15.3s	0.43s	-	19.2s	**0.49s**	-	23.2s	0.58s	-	27.4s	0.66s	-	31.9s	0.80s	-	36.6s	0.81s
80	2.57h	5.6s	**0.18s**	3.83h	8.8s	0.27s	-	12.3s	0.36s	-	15.9s	0.43s	-	19.9s	**0.49s**	-	24.2s	0.58s	-	28.6s	0.66s	-	33.3s	0.80s	-	38.7s	0.81s
90	2.95h	5.6s	0.19s	-	8.8s	0.30s	-	12.3s	0.34s	-	16.2s	0.42s	-	20.5s	0.50s	-	25.2s	0.57s	-	29.8s	0.70s	-	34.6s	0.76s	-	40.1s	0.82s
100	3.20h	5.7s	0.19s	-	9.0s	**0.25s**	-	12.9s	**0.33s**	-	16.9s	**0.41s**	-	21.4s	**0.49s**	-	25.9s	0.57s	-	30.9s	**0.65s**	-	36.2s	0.77s	-	41.9s	0.80s

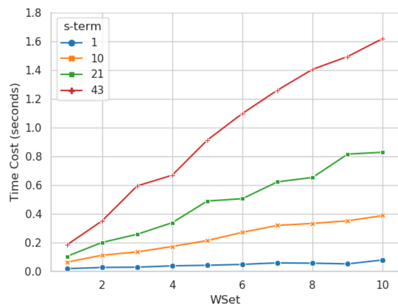
In the table, we highlight the best in "●" and the worst in "●" under different attributes in an attribute set Σ of AasBirch. We denote "-" as "$>$ 4.16 hours (i.e., 15,000 seconds)" that omitted to test.

To clarify practical utility of Retrieve, we see the time cost of *encrypted indexes retrieval*, *encrypted files receiving* and *encrypted files decryption* step-by-step in Fig. 8. As depicted in Fig. 8(a), the time cost for retrieving one file index converges to 0.01 seconds, which is a sufficient low cost for clients. In Fig. 8(b), it shows that the consumed time distribution between client's sending request and server's returning result. That is, most of the file receiving overhead

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3227650



(a) Different attribute set $|\Sigma|$



(b) Different s-term and Wset

Fig. 7. Searching cost with different s-term, Wset and $|\Sigma|$ in AasBirch

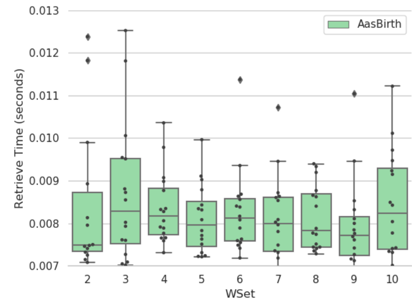
are around 0.2 seconds. Fig. 8(c) shows encrypted files decryption overhead of clients, where it is highly efficient with nearly 0.001 seconds. Ultimately, we may conclude that the Retrieve algorithm for a client is efficient enough and highly acceptable for practical secure cloud search services.

6.3.5 Summary

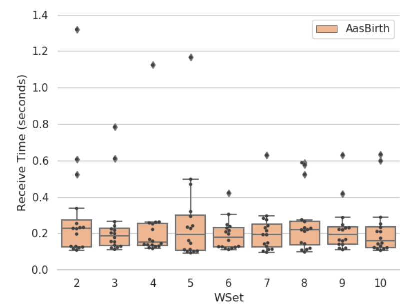
Generally, AasBirth enjoys better time and communication efficiency of system initialization, document encryption and document search. The system-running cost of [17]'s ABKS is related to the base of attribute universe $|\mathcal{U}|$, the number of attributes of an encrypted document $|\Sigma|$ and the number of keywords associated with a document |Wset|. While the main factors that influence the performance of [15] and our AasBirch are the s-term (the least frequent keyword in a query) and the number of keywords in a query q. Compared to [15], the efficiency of system initialization in our AasBirch is independent of the base of attribute universe $|\mathcal{U}|$ and the number of attributes of an encrypted document $|\Sigma|$.

7 Related Work

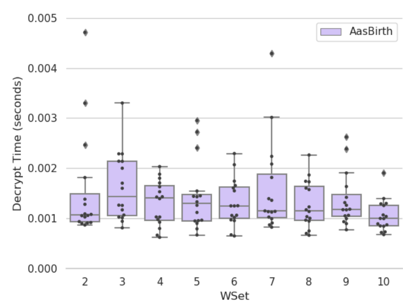
In the following, we discueesion the related work in 1ddddd(ABKS, ddddMCSSE. **Attribute-Based Keyword Search.** With an adoption of proxy re-encryption primitive, Liang et al. [8] proposed a novel ABKS scheme that simultaneously achieves encrypted data sharing and keyword search. Zheng et al. [7] introduced the notion of verifiable ABKS, which allows clients to verify whether the cloud honestly runs the searching process. By employing user revocation technique, Sun et al. [9] presented a scalable ABKS that supports fine-grained owner-enforced search authorization. For outsourced ABKS with key-issuing and



(a) Encrypted Indexes Retrivial



(b) Encrypted Dcouments Receiving



(c) Encrypted Documents Decryption

Fig. 8. Time cost for retrieving encrypted file index, receiving/decrypting encrypted indexes, *where only one file assumed to satisfy query pattern*

data outsourcing decryption, a KSF-OABE presented by Li et al. [10] only supports partial encrypted data retrieval that related with the issued keyword. To enhance user privacy, Wang et al. [19] proposed an effective hidden policy ABKS that realizes constant-size keyword search and documents storage. Xue et al. [30] proposed a robust and auditable access control for enhancing authorization in cloud storage services.

Nevertheless, these schemes constructed based on black-box or semi-black-box implementation of ABE, which leads to restricted query models and costly running overhead.

(Multi-client) Symmetric Searchable Encryption. Cash et al. [20] introduced a novel highly-scalable SSE scheme with the support of sub-linear boolean queries, where a s-term is introduced and employed to locate the least set of files under inverted index data structure. By employing blind storage technique, Naveed et al. [31] introduced an efficient SSE scheme that the cloud does not learn how many files are stored that belong to a data owner. Following [20], Sun et al. [11], [13], Kermanshahi et al. [32], Zeng et al. [12], Zhang et al. [15] studied the multi-client setting of SSE supporting boolean queries, which covers a variety of extension (e.g., fast efficiency, fine-grained authorization, public-key situation). Recently, Du et al. [14] combined a client's authorization information into search tokens and encrypted indexes, and thus proposed a DM-SSE scheme that allows a data owner to authorize multiple clients to perform boolean queries. Moreover, forward privacy and backward privacy have been formally considered for MC-SSE by [33], [34] that extended from [35], [36], [37]. Very recently, Du et al. [14] presented a dynamic MC-SSE scheme where data owners can update the search authorization of a data user, but it does not work with fine-grained authorization.

To summarize, a variety of desirable features are studied in existing MC-SSE solutions, such as boolean queries, fine-grained access control or forward and backward security. However, existing MC-SSE systems with fine-grained authorization have not well-considered authorization dynamic updating, and system-running time and storage efficiency can be further optimized. Consequently, the purpose of the work is to propose a dynamic and effective cloud-based file retrieval system with dynamic fine-grained authorization.

8 Conclusion

In this work, an AasBirch system for secure multi-client cloud search services is presented that allows data owners to flexibly switch enforced authorization. Furthermore, the realization for such authorization is direct and lightweight, which is not a (semi-)black-box implementation of ABE frameworks as existing solutions. In addition, AasBirch achieves constant-size public parameter, secret key and encrypted files indexes, where the overhead of file encrypting, authorization switching and file searching are also highly efficient. Nevertheless, the direct authorization considered in AasBirch is assumed as an "AND"-gate formula, while not supporting such as "OR", "Threshold" even "NOT" access control policy. Hence, it seems an interesting work for enabling AasBirch to support direct and more expressive authorization formulas.

Acknowledgments

This work was supported by National Natural Science Foundation of China (U1936213, 61872230, 61972094, 62032005), Shanghai Rising-Star Program (22QA1403800) and Program of Shanghai Academic Research Leader (21XD1421500).

References

[1] "Share of corporate data stored in the cloud in organizations worldwide from 2015 to 2021," https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate-data/.
[2] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet computing*, vol. 16, no. 1, pp. 69–73, 2012.
[3] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–51, 2014.
[4] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.
[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004, Proceedings*, 2004, pp. 506–522.
[6] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*. Ieee, 2010, pp. 1–9.
[7] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: verifiable attribute-based keyword search over outsourced encrypted data," in *2014 IEEE Conference on Computer Communications, INFOCOM*, 2014, pp. 522–530.
[8] K. Liang and W. Susilo, "Searchable attribute-based mechanism with efficient data sharing for secure cloud storage," *IEEE Trans. Information Forensics and Security*, vol. 10, no. 9, pp. 1981–1992, 2015.
[9] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1187–1198, 2016.
[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. Services Computing*, vol. 10, no. 5, pp. 715–725, 2017.
[11] S. Sun, J. K. Liu, A. Sakzad, R. Steinfeld, and T. H. Yuen, "An efficient non-interactive multi-client searchable encryption with support for boolean queries," in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Proceedings, Part I*, 2016, pp. 154–172.
[12] M. Zeng, K. Zhang, H. Qian, X. Chen, and J. Chen, "A searchable asymmetric encryption scheme with support for boolean queries for cloud applications," *The Computer Journal*, vol. 62, no. 4, pp. 563–578, 2019.
[13] S.-F. Sun, C. Zuo, J. K. Liu, A. Sakzad, R. Steinfeld, T. H. Yuen, X. Yuan, and D. Gu, "Non-interactive multi-client searchable encryption: Realization and implementation," *IEEE Transactions on Dependable and Secure Computing*, 2020, https://doi.org/10.1109/TDSC.2020.2973633.
[14] L. Du, K. Li, Q. Liu, Z. Wu, and S. Zhang, "Dynamic multi-client searchable symmetric encryption with support for boolean queries," *Information Sciences*, vol. 506, pp. 234–257, 2020.
[15] K. Zhang, M. Wen, R. Lu, and K. Chen, "Multi-client sub-linear boolean keyword searching for encrypted cloud storage with owner-enforced authorization," *IEEE Transactions on Dependable and Secure Computing*, 2020, https://doi.org/10.1109/TDSC.2020.2968425.
[16] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Proceedings*, pp. 89–98.
[17] B. Hu, Q. Liu, X. Liu, T. Peng, G. Wang, and J. Wu, "Dabks: Dynamic attribute-based keyword search in cloud computing," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.
[18] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang, and K. Ren, "Enabling generic, verifiable, and secure data search in cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1721–1735, 2018.
[19] H. Wang, J. Ning, X. Huang, G. Wei, G. S. Poh, and X. Liu, "Secure fine-grained encrypted keyword search for e-healthcare cloud," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.
[20] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology - CRYPTO 2013, Part I*, 2013, pp. 353–373.
[21] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 875–888.
[22] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Advances in Cryptology - EUROCRYPT 2017, Proceedings, Part III*, 2017, pp. 94–124.
[23] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *European symposium on research in computer security*. Springer, 2015, pp. 123–145.
[24] J. Lai, R. H. Deng, Y. Yang, and J. Weng, "Adaptable ciphertext-policy attribute-based encryption," in *International Conference on Pairing-Based Cryptography*. Springer, 2013, pp. 199–214.
[25] S. Hohenberger and B. Waters, "Online/offline attribute-based

encryption," in *International workshop on public key cryptography*. Springer, 2014, pp. 293–310.
[26] "Enron dataset," http://www.cs.cmu.edu/~./enron/.
[27] "Huaweicloud," https://www.huaweicloud.com/.
[28] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.
[29] "Pytextrank," https://spacy.io/universe/project/spacy-pytextrank.
[30] K. Xue, Y. Xue, J. Hong, W. Li, H. Yue, D. S. Wei, and P. Hong, "Raac: Robust and auditable access control with multiple attribute authorities for public cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.
[31] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 639–654.
[32] S. K. Kermanshahi, J. K. Liu, R. Steinfeld, S. Nepal, S. Lai, R. Loh, and C. Zuo, "Multi-client cloud-based symmetric searchable encryption," *IEEE Transactions on Dependable and Secure Computing*, 2019, https://doi.org/10.1109/TDSC.2019.2950934.
[33] A. Bakas and A. Michalas, "Multi-client symmetric searchable encryption with forward privacy." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 813, 2019.
[34] Q. Gan, X. Wang, D. Huang, J. Li, D. Zhou, and C. Wang, "Towards multi-client forward private searchable symmetric encryption in cloud computing," *IEEE Transactions on Services Computing*, 2021, https://doi.org/10.1109/TSC.2021.3087155.
[35] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1038–1055.
[36] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy," in *European Symposium on Research in Computer Security*. Springer, 2019, pp. 283–303.
[37] Y. Zheng, R. Lu, J. Shao, F. Yin, and H. Zhu, "Achieving practical symmetric searchable encryption with search pattern privacy over cloud," *IEEE Transactions on Services Computing*, 2020, https://doi.org/10.1109/TSC.2020.2992303.

Kai Zhang received the Bachelor's degree with Computer Science and Technology from Shandong Normal University, China, in 2012, and the Ph.D. degree with Computer Science and Technology from East China Normal University, China, in 2017. He visited Nanyang Technological University in 2017. He is currently an Associate Professor with Shanghai University of Electric Power, China. His research interest includes applied cryptography and information security.

Xiwen Wang received the bachelor's degree from Shanghai University of Electric Power, China, in 2021. He is currently pursuing his master degree in College of Computer Science and Technology from Shanghai University of Electric Power, China. His research interests include cloud security and applied cryptography.

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 Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, China. Previously, he was a Research Scientist at the School of Computing and Information Systems, Singapore Management University. He has published papers in major conferences/journals, such as ACM CCS, ASIACRYPT, ESORICS, ACSAC, IEEE TIFS, and IEEE TDSC. His research interests include applied cryptography and information security.

Mi Wen received the M.S. degree in Computer Science from University of Electronic Science and Technology of China in 2005 and the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China in 2008. She is currently a Professor of the College of Computer Science and Technology, Shanghai University of Electric Power. From May 2012 to May 2013, she was a visiting scholar at University of Waterloo, Canada. She serves Associate Editor of Peer-to Peer Networking and Applications (Springer). She keeps acting as the TPC member of some flagship conferences such as IEEE INFOCOM, IEEE ICC, IEEE GLOEBECOM, etc from 2012. Her research interests include privacy preserving in wireless sensor network, smart grid etc

Rongxing Lu is an associate professor, University Research Scholar, at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. Rongxing Lu worked as a Postdoctoral Fellow at the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal", when he received his PhD degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012; and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. Dr. Lu is an IEEE Fellow.