DABKS: Dynamic Attribute-based Keyword Search in Cloud Computing

Baishuang Hu [1], Qin Liu [1], Xuhui Liu [1], Tao Peng [2], Guojun Wang [3], and Jie Wu [4]

[1] Hunan University, China
[2] Central South University, China
[3] Guangzhou University, China
[4] Temple University, USA

Outline

01

Introduction

Introduction

Due to its fast deployment and scalability, cloud computing has become a significant technology trend. Organizations with limited budgets can achieve great flexibility at a low price by outsourcing their data and query services to the cloud. Since the cloud is outside the organization's trusted domain, existing research suggests encrypting data before outsourcing to preserve user privacy.

Two main problems that the cloud user faces while searching over encrypted data are ***how to achieve a fine-grained search authorization*** and ***how to efficiently update the search permission***. The existing attribute-based keyword search (ABKS) scheme addresses the first problem, which allows a data owner to control the search of the outsourced encrypted data according to an access policy. However, the research on how to efficiently update the search permission is relatively little.

The ABKS scheme

Scheme description:

- The attribute-based keyword search (ABKS) scheme utilized attribute-based encryption (ABE) to achieve fine-grained search authorization for public-key settings.
- In ABKS, each keyword w_i is associated with an access policy AP, and each search token is associated with a keyword w_j and a set of attributes S. The data user can search the file only when her attributes satisfy the access policy, denoted as $S \prec AP$, and $w_j = w_i$.

Drawback of ABKS:

- ABKS never considered the problem of a dynamic access policy for keywords.
- If AP is changed to AP', the data owner needs to re-encrypt the relevant keywords with AP' so that only the users whose attributes satisfy AP' have search permission. For frequent updates on a large number of files, the workload on the data owner is heavy.

Solution

In this paper, we focus on solving the policy updating problem in the ABKS scheme, and propose a dynamic attribute-based keyword search (DABKS) scheme by incorporating proxy re-encryption (PRE) and a secret sharing scheme (SSS) into ABKS.

Instead of retrieving and re-encrypting the data, data owners only send a policy updating query to the cloud server, and the cloud server can update the policy of the encrypted data without decrypting it. Our scheme can can largely reduce the workload on the data owner by delegating the policy updating operations to the cloud.

Contributions

1. To the best of our knowledge, this is the first attempt made to devise a dynamic access policy for fine-grained search authorization in a cloud environment.
2. The proposed DABKS scheme can largely reduce the workload on the data owner by delegating the policy updating operations to the cloud.
3. We conduct experiments on real data sets to validate the effectiveness and efficiency of our proposed scheme.

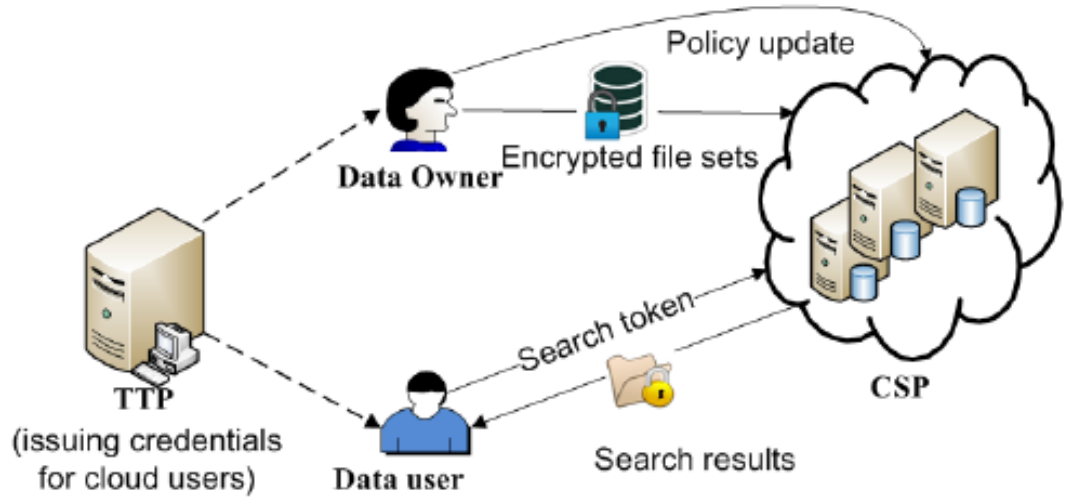
02

Preliminaries

System model

The system is composed of the following parties: the cloud users, the cloud service provider (CSP), and a trusted third party (TTP).

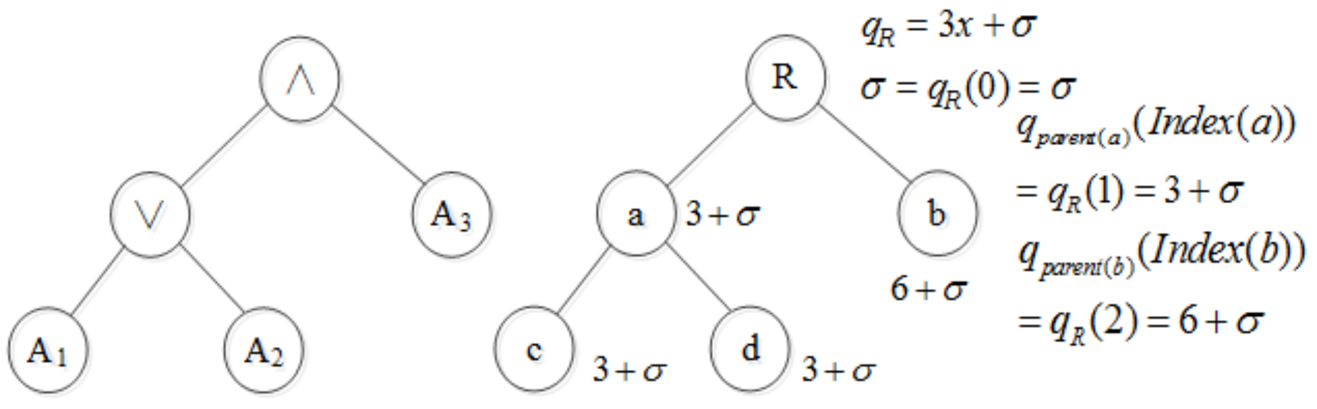
- The cloud users, who pay the services residing on the cloud or deploy their own applications/systems in the cloud, can be further classified into data owner and data user. The data owner outsources the encrypted data and keywords to the cloud and authorizes multiple data users to access them. The data user will retrieve data of interest according to a keyword-based search.
- The CSP operates the cloud platforms, which provide not only the storage and search services, but also perform policy updating operations on behalf of the data owner.
- The TTP is responsible for issuing credentials to all cloud users.



Access tree

The access policy *AP* can be depicted as an access tree *T* where each interior node denotes a gate and each leave node is depicted as an attribute. In *T*, each node *x* is associated with a threshold value k_x. For the interior node *x* with N_x children, $k_x = 1$ when x is an OR gate, and $k_x = N_x$ when x is an AND gate. For all leave nodes, the threshold value is 1. Let lev(*T*) denote leave nodes in *T*. If *x* *lev*(*T*), *att*(*x*) is used to denote the attribute associated with node *x*. Furthermore, *T* defines an ordering between the children of each node, and *parent*(*x*) and *index*(*x*) return the parent and the order number of children node *x*, respectively.

◆ access policy: $(A_1 \vee A_2) \wedge A_3$



(a) Sample access tree (b) Sample tree implementation

Secret Sharing Scheme(sss)

- To share the secret σ in T, the SSS generates $\Delta = \{q_x(0)\}_{x\in T}$ as follows:

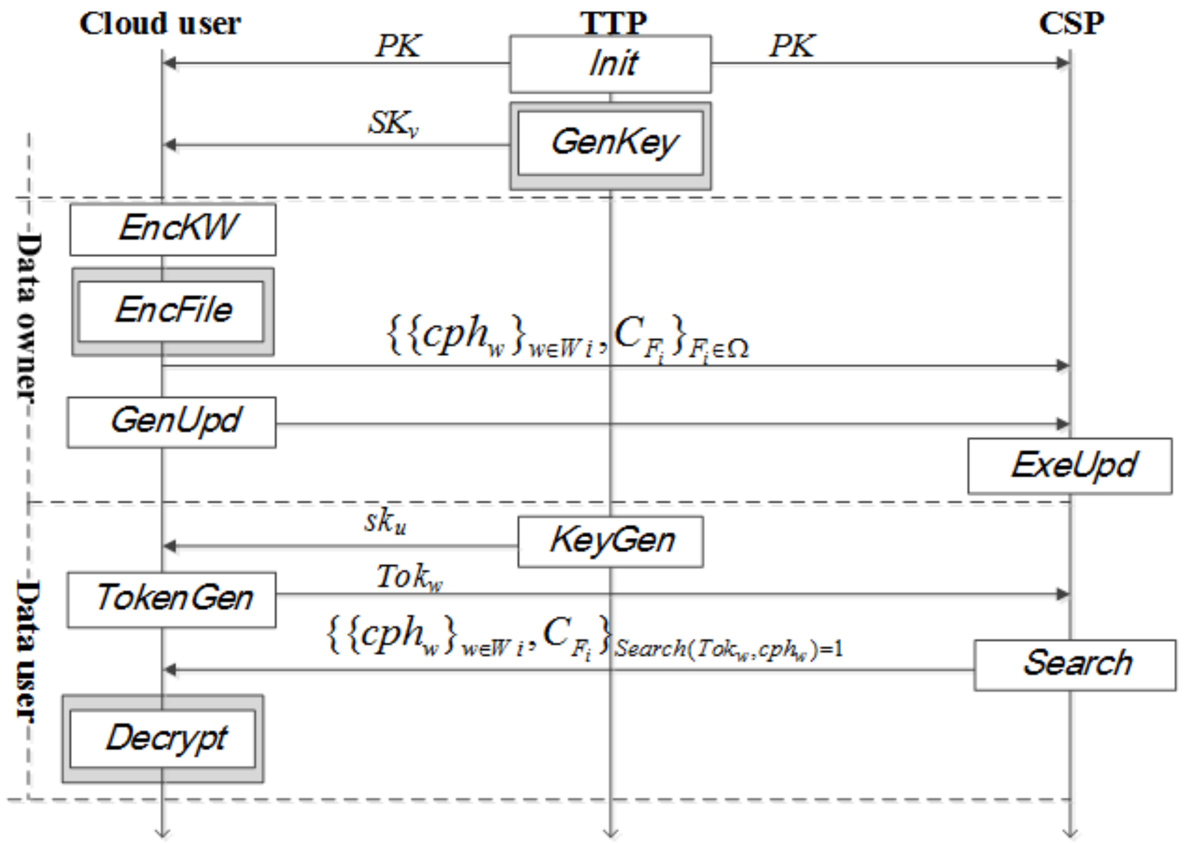
  SSS(σ , T) $\rightarrow \Delta$. A random polynomial q_R of degree $k_R - 1$ is chosen for $q_R(0) = \sigma$. The rest of the points in q_R are randomly chosen. For each node $x \in T$, a random polynomial q_x of degree $k_x - 1$ is chosen for $q_x(0) = q_{parent(x)}(index(x))$. The rest of points in q_x are chosen randomly. To recover the secret σ, the users with sufficient secret shares can perform Lagrange interpolation recursively.

03

Scheme description

Working process of DABKS scheme

- Figure on the right shows the working process of the DABKS scheme, where algorithms *GenKey*, *EncFile*, and *Decrypt* related to ABE are used to preserve file privacy.
- Since our workfocuses on preserving keyword privacy and query privacy, we omit the construction of the following algorithms in this paper : *Init*, *KeyGen*, *EncKW*, *TokenGen*, *Search*, *GenUpd*, *ExeUpd*.
  - ✓ *GenUpd* algorithm generates an update key **UK** for policy updating.
  - ✓ *ExeUpd* algorithm updates the original ciphertext cph_w to new ciphertext cph'_w .



Policy updating

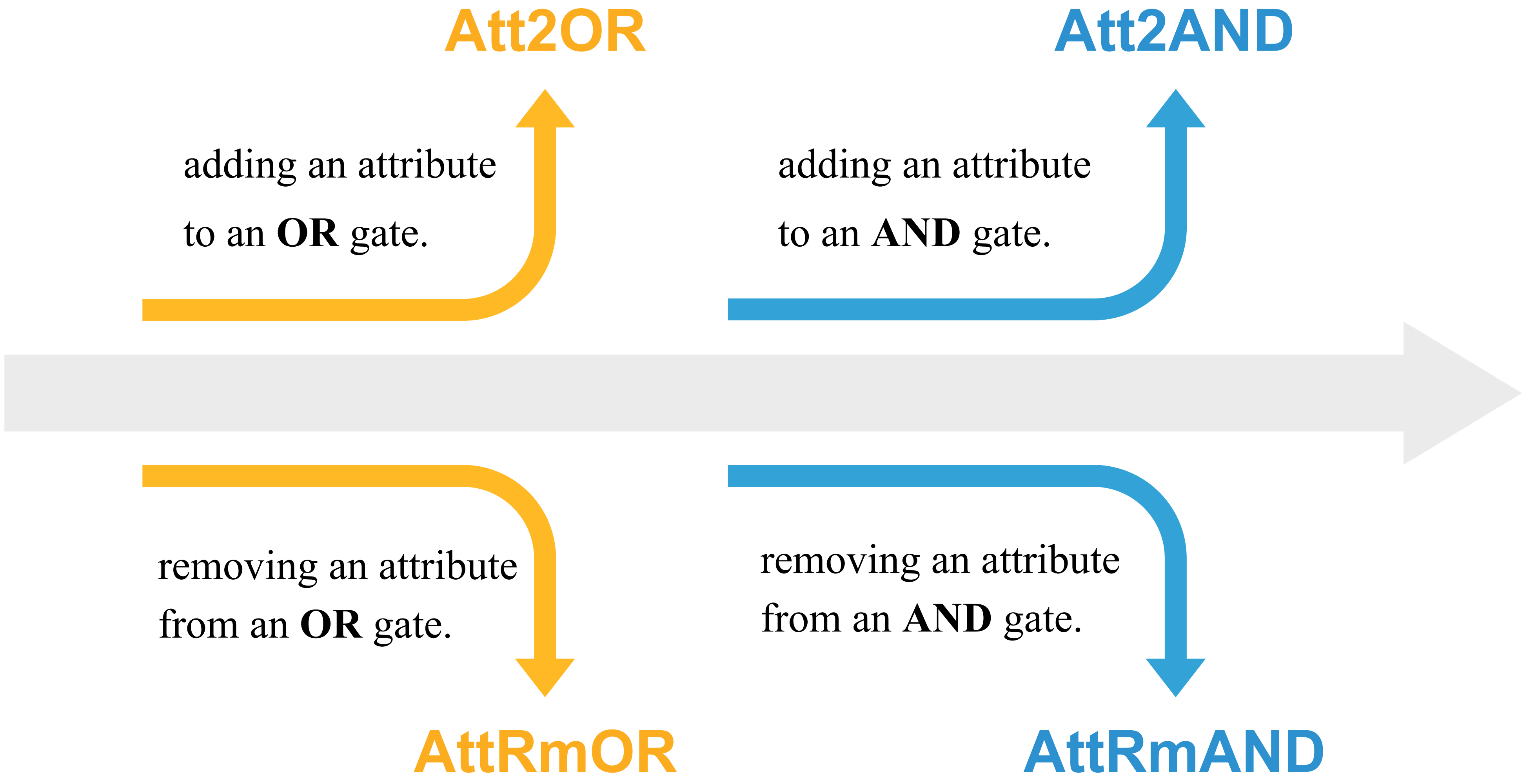
Description of problem:

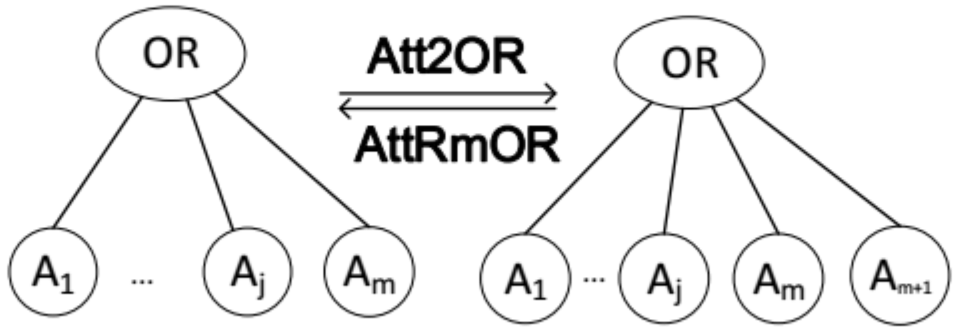
The policy updating is a difficult issue in attribute-based access control systems, because once the data owner outsourced the data into the cloud, it would not keep a copy in local systems. When the data owner wants to change the access policy, it has to transfer the data back to the local site from the cloud, re-encrypt the data under the new access policy, and then move it back to the cloud server. By doing so, it incurs a high communication overhead and heavy computation burden on data owners.

Our work:

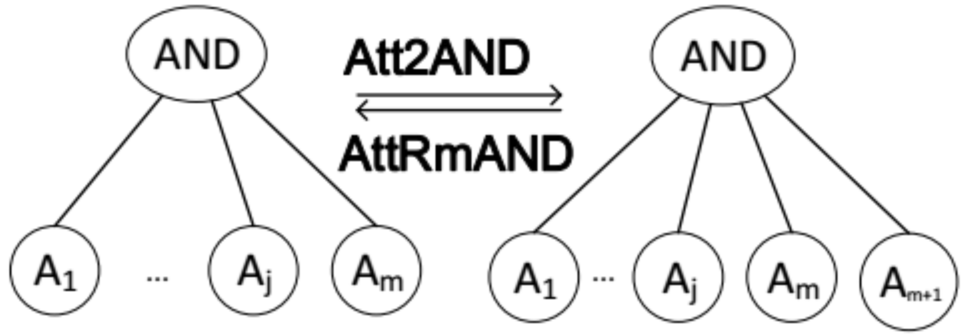
developing a new method to outsource the policy updating to the cloud server.

Four basic operations involved in policy updating


Att2OR
adding an attribute to an **OR** gate.
Att2AND
adding an attribute to an **AND** gate.
removing an attribute from an **OR** gate.
AttRmOR
removing an attribute from an **AND** gate.
AttRmAND



(a)Att2OR and AttRmOR



(b)Att2AND and AttRmAND

LOREM IPSUM DOLOR

- Our scheme expresses the access policy *AP* as an access tree and transforms the problem of updating an **AND/OR** gate in *AP* to that of updating a threshold gate.
- For example, the **AND** gate is transformed to *(t,t)* gate, and the **OR** gate is transformed to *(1,t)* gate. Therefore, the updating of the **AND** gate can be treated as updating *(t,t)* gate to *(t′,t′)* gate, and the updating of the **OR** gate can be treated as updating *(1,t)* gate to *(1,t′)* gate, where t′ = t + 1 for adding an attribute to the **AND/OR** gate and t′ = t – 1 for removing an attributes from the **AND/OR** gate.

Att2OR

Att2OR:

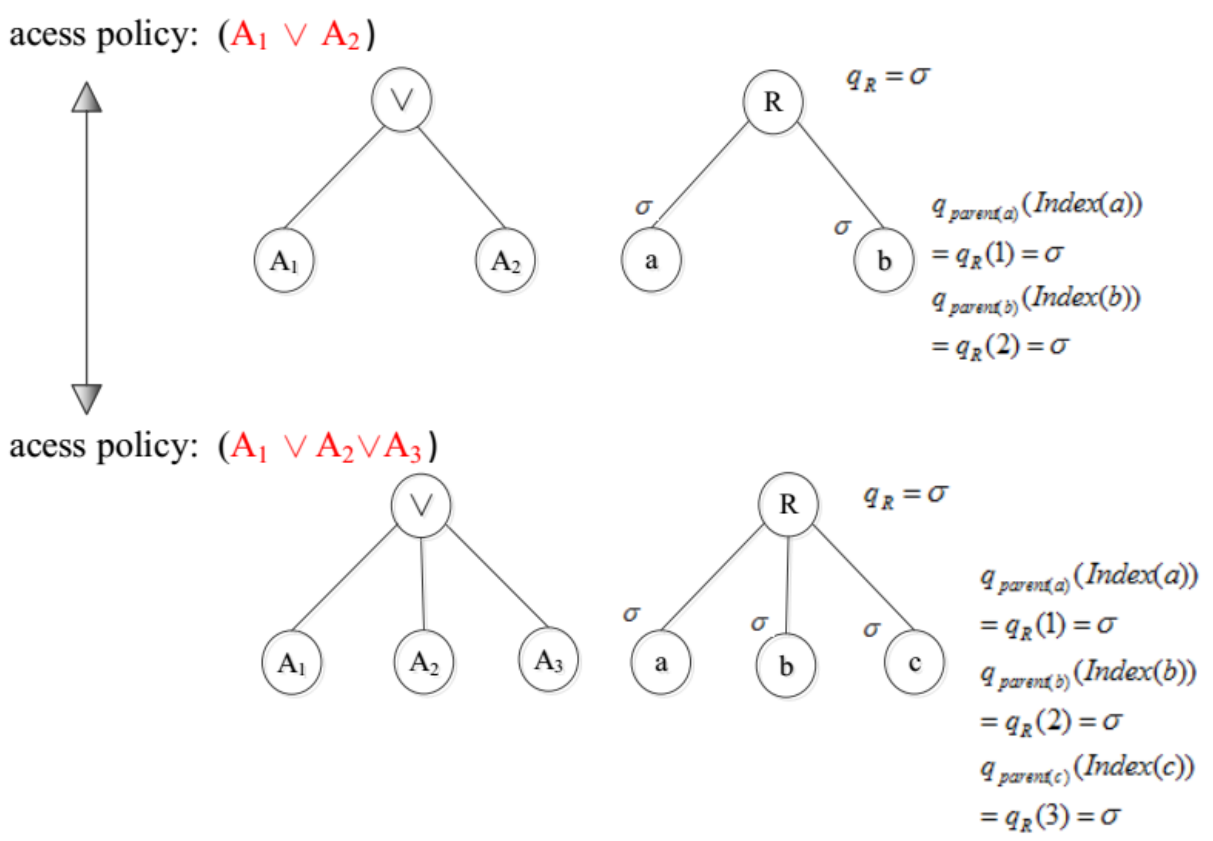
✓Given $q_R(0)$ and the new access policy AP', the data owner runs SSS to generate new shares $\{q'_a(0), q'_b(0), q'_c(0)\}$ for attributes A_1, A_2, A_3.

$$q'_a(0) = q'_b(0) = q'_c(0) = q_R(0)$$

✓The ciphertexts for the original attributes will not be changed,i.e., $C'_1 = C_1$, $C'_2 = C_2$.

✓For the newly added attribute A_3, the data owner needs to generate a new ciphertext C'_3 based on $q'_c(0)$.

✓The CSP will add the new ciphertext C'_3 to the cph_w , and update the access tree by adding A_3 under node R.



acess policy: $(A_1 \vee A_2)$
$\vee$
A_1
A_2
R
$q_R = \sigma$
σ
a
b
$q_{parent(a)}(Index(a)) = q_R(1) = \sigma$
$q_{parent(b)}(Index(b)) = q_R(2) = \sigma$
acess policy: $(A_1 \vee A_2 \vee A_3)$
$\vee$
A_1
A_2
A_3
R
$q_R = \sigma$
σ
a
b
c
$q_{parent(a)}(Index(a)) = q_R(1) = \sigma$
$q_{parent(b)}(Index(b)) = q_R(2) = \sigma$
$q_{parent(c)}(Index(c)) = q_R(3) = \sigma$

Att2AND

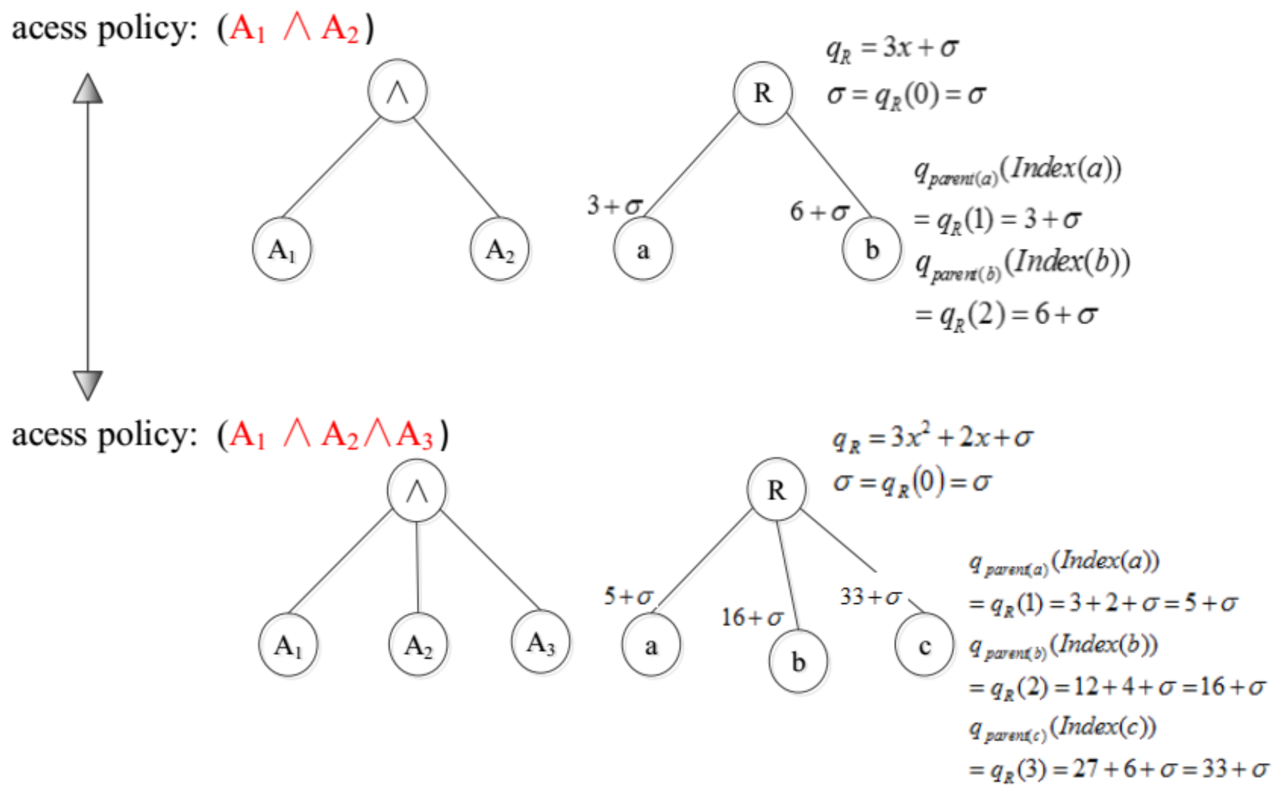
Att2AND:

➤Given $q_R(0)$ and the new access policy AP′, the data owner runs SSS to generate new shares $\{q'_a(0), q'_b(0), q'_c(0)\}$ for attributes A_1, A_2, A_3.

➤Executing the ***GenUpd*** algorithm to generate the update key **UK** for attributes A_1, A_2.

➤ For the newly added attribute A_3, the data owner needs to generate a new ciphertext C'_3 based on $q'_c(0)$.

➤The CSP will execute the ***ExeUpd*** algorithm to update the ciphertext C_1 to C'_1, C_2 to C'_2, add the new ciphertext C'_3 to the cph_w, and update the access tree by adding A_3 under node *R*.



acess policy: $(A_1 \wedge A_2)$
$\wedge$
A_1
A_2
R
$q_R = 3x + \sigma$
$\sigma = q_R(0) = \sigma$
$3+\sigma$
$6+\sigma$
a
b
$q_{parent(a)}(Index(a)) = q_R(1) = 3+\sigma$
$q_{parent(b)}(Index(b)) = q_R(2) = 6+\sigma$
acess policy: $(A_1 \wedge A_2 \wedge A_3)$
$\wedge$
A_1
A_2
A_3
R
$q_R = 3x^2 + 2x + \sigma$
$\sigma = q_R(0) = \sigma$
$5+\sigma$
$16+\sigma$
$33+\sigma$
a
b
c
$q_{parent(a)}(Index(a)) = q_R(1) = 3+2+\sigma = 5+\sigma$
$q_{parent(b)}(Index(b)) = q_R(2) = 12+4+\sigma = 16+\sigma$
$q_{parent(c)}(Index(c)) = q_R(3) = 27+6+\sigma = 33+\sigma$

04

Evaluation

Parameter setting

In this section, we will analyze the performance of our DABKS scheme. Our experiments are conducted with Java programming language. We implement our scheme on a local machine with an Inter Core i5 CPU running at 3.2GHz and 8GB memory.

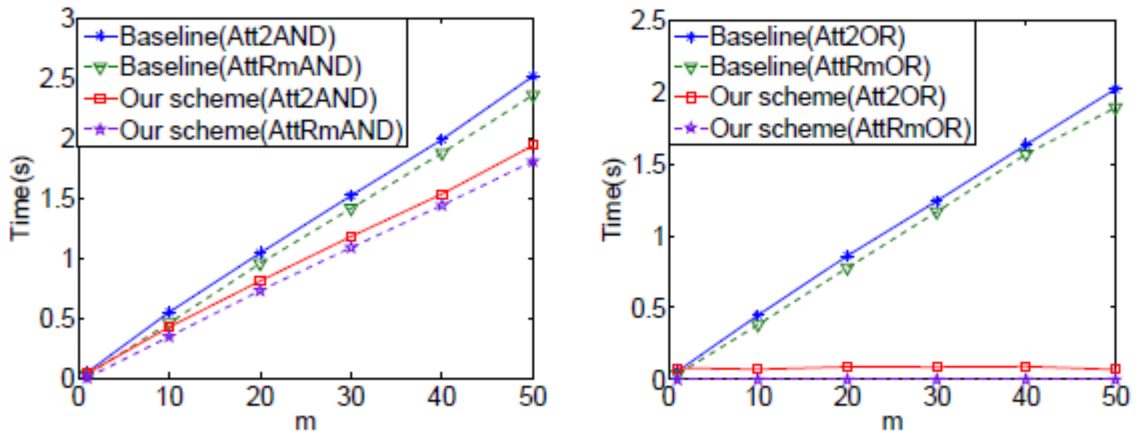
- The parameter setting in the experiments is as follows:

  the number of attributes under a updating gate node m ranges from 1 to 50

- For the ***Att2AND*** and ***AttRmAND*** operations, the access policy is set as $AP = (A_1 \wedge A_2 \wedge \ldots \wedge A_m)$
- For the ***Att2OR*** and ***AttRmOR*** operations, the access policy is set as $AP = (A_1 \vee A_2 \vee \ldots \vee A_m)$

Experiment results

- In ABKS, once the access policy is changed, the data owner needs to re-encrypt the relevant keywords with the new access policy and to send the new ciphertexts to the cloud.
- The baseline denotes the re-encryption cost on the data owner in ABKS.
- The execution time of the *GenUpd* algorithm is shown in Figure 5. For the four basic operations involved in policy updating, the execution time in our scheme are smaller than those in baseline.
- The experiment results prove that the cloud user should outsource the keyword search and policy update operations to the CSP in order to take full advantage of the CSP's vast computation capabilities. Therefore, the workload of the data user will be largely reduced.



(a) $Att2AND$ and $AttRmAND$ (b) $Att2OR$ and $AttRmOR$

Fig. 5. Execution time (in seconds) for *GenUpd*.

05

Conclusion

Conclusion

- In this paper, we propose a DABKS scheme to simultaneously achieve fine-grained search authorization and efficient update of access policy. Our scheme takes full advantage of cloud resources by delegating policy update operations to the CSP.
- Experiment results verify its feasibility and effectiveness. However, the DABKS scheme supports only the single-keyword search.
- As part of our future work, we will try to extend our scheme to a multi-keyword search scenario, which supports conjunctive, subset, and range queries on encrypted outsourced data.

THANKS