

vABS: Towards Verifiable Attribute-Based Search over Shared Cloud Data

Yang Ji^{†1}, Cheng Xu^{†2}, Jianliang Xu^{†3}, Haibo Hu^{†4}

[†]Department of Computer Science, Hong Kong Baptist University, Hong Kong

[‡]Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Hong Kong

{¹yangji, ²chengxu, ³xujl}@comp.hkbu.edu.hk, ⁴haibo.hu@polyu.edu.hk

Abstract—With the proliferation of cloud computing and data-as-a-service (DaaS), more and more organizations and individuals outsource their data to a third-party service provider. While enjoying the benefits of cloud-based data outsourcing, the data owners are at the risk of losing control of data integrity and access management. In this demonstration, we present a system called vABS, which enables verifiable Attribute-Based Search over shared cloud data. The vABS system adopts the common DaaS architecture, in which the server provides search services to users on behalf of data owners. By employing a novel zero-knowledge approach proposed in our prior work [1], vABS not only provides users with good search experiences, but also supports authenticated query processing with fine-grained access control, which is crucial to many high-security applications.

I. INTRODUCTION

Driven by the low cost and high availability of cloud computing technologies, recent years are witnessing the proliferation of cloud-based data outsourcing. With the benefits of cost effectiveness and high performance, the cloud data engines, acting as a *service provider* (SP), can facilitate data storage and information search services for *data owners* (DOs). For example, Microsoft HealthVault [2] has developed a cloud-based platform that allows health data to be shared and exchanged among users.

Despite the benefits of cloud-based data outsourcing, it raises security issues which are deeply concerned by the public. First, from the perspective of data integrity, the correctness of search results cannot be guaranteed if the SP tampers with the data records deliberately. Second, due to the increasingly stringent data compliance requirement, any information leakage to unauthorized parties may lead to devastating consequences. Query authentication and access control have been studied by a large body of literature [3]–[6]. However, prior works only consider either one of them. There is a pressing need for enterprise cloud database systems to achieve both data integrity and access control. Below are two examples.

Example 1: Small and medium enterprises may prefer to outsource their database to a cloud service provider, and authorize it to provide search services for customers. However, when the customers search information from the server, they may have the following questions. Is there any record tampered with or omitted in the search results? How to verify whether all records not returned are either inaccessible to them or non-results?

Example 2: To prepare a cancer rehabilitation plan for a patient, Dr. Bob needs to retrieve detailed medical records of this patient that are related to this cancer disease, including historical diagnosis and lab test results. Bob can search all information that he is authorized to access. However, he may be curious about other medical records of this patient that he is not authorized to access, such as the HIV test result. Furthermore, he may also be curious about the roles he lacks to access such records.

Motivated by these application scenarios, we adopt a novel access-policy-preserving (APP) signature developed in our prior study [1] as *authenticated data structure* (ADS), which supports verifiable attribute-based search with fine-grained access control in *zero-knowledge*. The APP signature reveals nothing beyond accessible records. For example, if no record is returned for a given search key, the user cannot infer whether it is because there does not exist a matching record or because the matching record is inaccessible to her. To further improve the search performance, a grid-index-based tree structure is implemented to aggregate the APP signatures. It is our purpose in this demonstration to show the usability and feasibility of our proposed verifiable search techniques. To this end, we develop a vABS prototype system, which can ensure data search integrity and flexible access control simultaneously. Specifically, we design and implement a query processing module embedded with a *verification object* (VO) construction engine on the server side, and an application client for query submission, result presentation and verification. The vABS system also provides an interactive search experience to users.

The rest of the demonstration proposal is organized as follows. Section 2 elaborates the techniques that enable verifiable attribute-based search over shared cloud data. Section 3 overviews the vABS prototype system. The interface of vABS and demonstration details are presented in Section 4.

II. TECHNICAL BACKGROUND

We start with a brief introduction to the building blocks of vABS. A tuple $\langle o_i, v_i, \Upsilon_i \rangle$ denotes a record in a relational database \mathcal{D} , where o_i is the search key, v_i is the record content, and Υ_i is the access policy: $\{0, 1\}^n \rightarrow \{0, 1\}$. Access policies control the results of a search with respect to the roles of the user. For example, if Υ is defined as $(Role_A \wedge Role_C) \vee Role_B$, then $\Upsilon(Role_A) = false$ and $\Upsilon(Role_A, Role_C) = true$.

A. ADS Generation

In order to authenticate attribute-based search with fine-grained access control, a naive solution is to use the Merkle hash tree [7] for result verification and to use the ciphertext-policy attribute-based encryption (CP-ABE) [8] for access control. However, this solution has the following shortcomings. First, a mass quantity of inaccessible data need to be returned to the user for completeness checking, which incurs high communication and computation overheads. Second, although inaccessible records cannot be decrypted due to CP-ABE, it reveals the existence of such records, which violates the zero-knowledge confidentiality requirement.

A novel APP signature based on a variant of the attribute-based signature (ABS) [9] is proposed in our prior study [1] to address the above issues. To prevent information leakage caused by the non-existent records, we introduce a global pseudo access role $Role_\emptyset$, which is not possessed by any user. We treat each non-existent record as a *pseudo* record that is associated with this access policy $Role_\emptyset$. As such, for any equality search, there is always one matching record with one of the two possible outcomes: accessible or inaccessible. To authenticate accessible records, the APP signature is defined as follows.

Definition 1 (APP Signature): Consider a record $\langle o_i, v_i, \Upsilon_i \rangle$. Let sk_{DO} be the signing key of the DO, $hash(\cdot)$ a cryptographic hash function, and ‘|’ denote string concatenation. The APP signature for this record is generated as:

$$\sigma_i = \text{ABS.Sign}(sk_{DO}, hash(o_i) | hash(v_i), \Upsilon_i)$$

As for non-existent records, we assign a random value to v_i and $Role_\emptyset$ to Υ_i .

The APP signature has the following important properties. First, as a proof of data integrity, it captures all components of a record including search key o_i , record content v_i , and access policy Υ_i . Second, the APP signature can be tailored to support the proof of inaccessibility with the zero-knowledge confidentiality. If the APP signature is used directly as the VO for an inaccessible record, it would reveal the access policy. To cope with this problem, a super access policy $\hat{\Upsilon}_A$ is introduced. It is the weakest condition in which the user cannot access the record. For example, in Figure 1, if the access role universe \mathbb{A} is $\{Role_\emptyset, Role_A, Role_B, Role_C\}$, then $\hat{\Upsilon}_{\{Role_C\}} = Role_\emptyset \vee Role_A \vee Role_B$ for user u_2 . Based on this, we introduce the following access-policy-stripped (APS) signature to authenticate inaccessible records.

Definition 2 (APS Signature): Consider a record $\langle o_i, v_i, \Upsilon_i \rangle$. \mathbb{A} and \mathcal{A} denote the global access role set and the query user’s role set, respectively. Let sk_{DO} be the signing key of the DO, $hash(\cdot)$ a cryptographic hash function, and ‘|’ the string concatenation. The APS signature for this record and the user with access role set \mathcal{A} is defined as:

$$\hat{\sigma}_{i,\mathcal{A}} = \text{ABS.Sign}(sk_{DO}, hash(o_i) | hash(v_i), \hat{\Upsilon}_A),$$

where $\hat{\Upsilon}_A = a_1 \vee a_2 \vee \dots \vee a_n$, $a_i \in \mathbb{A} \setminus \mathcal{A}$.

It is worth noting that one can derive the APS signature from the corresponding APP signature without secret keys by

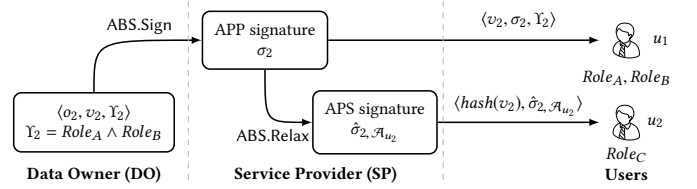


Fig. 1: Equality Query Authentication

invoking the $ABS.Relax(\cdot)$ function. More technical details can be found in [1].

B. Verifiable Equality Search

In an equality search, the user specifies a search key o as well as her access role set \mathcal{A} . If the access is granted, the SP simply returns the matching record with the APP signature directly. The user can verify the integrity of the search result by the returned APP signature. On the other hand, if the user cannot access the queried data record, an APS signature under the super access policy is computed as VO. Thanks to the super access policy and the perfect privacy-preserving property of the underlying cryptographic primitives, the user can verify that the queried data record is indeed inaccessible, but cannot deduce any additional information. Figure 1 presents an example of two different users issuing an equality search with key o_2 . User u_1 is allowed to access the record o_2 , so it is straightforward to return the APP signature σ_2 produced by the DO as VO. For user u_2 , the record o_2 is inaccessible. As such, the SP will derive an APS signature from the APP signature and send back $\langle hash(v_2), \hat{\sigma}_2, \mathcal{A}_{u_2} = \text{ABS.Relax}(\sigma_2, \mathbb{A} \setminus \mathcal{A}_{u_2}) \rangle$ for verification of inaccessibility.

To further prevent impersonation attacks, the SP employs the CP-ABE to encrypt a session key with the access policy $a_1 \wedge a_2 \wedge \dots \wedge a_n$ ($a_i \in \mathcal{A}$), which is then used to encrypt the search result using a standard symmetric cipher AES. In this way, only those users who possess all essential roles can decrypt the search result.

C. Verifiable Range Search

To support efficient range search, the *Access-Policy-Preserving Grid-Tree* (AP²G-Tree) is proposed, which is an authenticated index tree for the DO to construct and sign. Figure 2 shows a multi-layer grid-tree on a 2D data space, which partitions the search key space recursively into multiple levels of grid cells until each cell contains only one record.

Each grid cell in Figure 2a has a corresponding tree node N_i in the AP²G-Tree shown in Figure 2b. Each tree node consists of three components: the grid cell’s bounding box (denoted by gb_i), the access policy (denoted by p_i), and the APP signature (denoted by sig_i). The latter two components can be computed from its C child entries, c_1, \dots, c_C . For a non-leaf node, its access policy is $p_i = p_{c_1} \vee p_{c_2} \vee \dots \vee p_{c_C}$ and its corresponding APP signature $sig_i = \text{ABS.Sign}(sk_{DO}, gb_i, p_i)$, where sk_{DO} is the signing key of the DO. For a leaf node, its access policy and APP signature are identical to those of the corresponding record in the grid cell. For example, in Figure 2b, the access

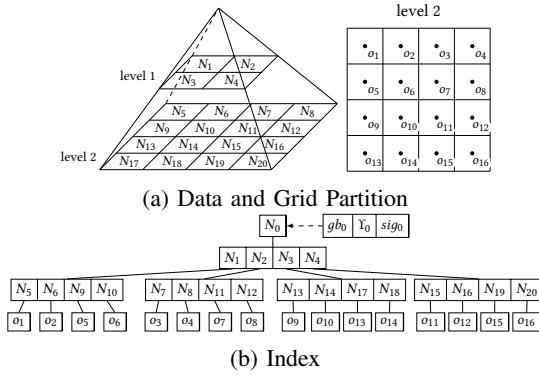


Fig. 2: Access-Policy-Preserving Grid-Tree (AP²G-Tree)

policy for N_1 is computed as $p_{N_1} = p_{N_5} \vee p_{N_6} \vee p_{N_9} \vee p_{N_{10}}$, and its APP signature is $sig_{N_1} = \text{ABS.Sign}(sk_{DO}, gb_{N_1}, p_{N_1})$; and for leaf node N_5 , its policy and signature are Υ_{o_1} and σ_{o_1} , respectively.

Being a space-partitioning index, the AP²G-Tree prevents information leakage through the tree structure. However, the grid-tree is not the most efficient index structure, especially for sparse multi-dimensional datasets. Thus, to optimize search performance, an alternative data-partitioning AP²kd-Tree has also been proposed in [1], by which the zero-knowledge confidentiality requirement is relaxed.

Regardless of the authenticated index structure employed, the query processing algorithm is executed in a top-down fashion. It performs a tree search starting from the root node. If the search range partially overlaps with the indexing space of the current tree node, the SP will branch the subtrees recursively. Otherwise, if a tree node is fully covered by the search range, the SP will check whether the user can access the node. If the access is denied, the SP will compute the APS signature for this node and add the derived signature to the VO. If the access is granted, the SP will further explore the subtrees until reaching the leaf nodes. For leaf nodes, the SP will return accessible records as search result, along with their corresponding APP signatures.

The user can verify the correctness of the search result by checking two conditions: (i) *soundness*: whether or not all signatures in the VO are valid; (ii) *completeness*: whether or not the indexing spaces of all entries in the VO together cover the whole search range.

III. VABS SYSTEM OVERVIEW

The vABS prototype system provides verifiable attribute-based search services over shared health record data, and ensures the integrity of search results. As depicted in Figure 3, vABS adopts a typical DaaS architecture that consists of client, server, and data owner. Specifically, the client side provides users with an attribute-based query interface, a verification module, and a result presentation module. The server side consists of two modules, namely, query processing and VO construction. In what follows, we elaborate the workflow and modules of vABS.

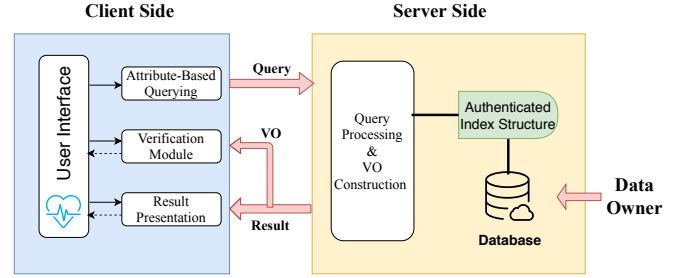


Fig. 3: System Architecture of vABS

A. Client Side

For the purpose of identity authentication, vABS requires user login. There is a user role set corresponding to each account in vABS. The main interface enables users to submit equality or range search requests by typing in a patient ID and a scope of his/her check-up numbers. All the search requests are transmitted to the server in the HTTP post method.

Once receiving the search result from the server, the client will present it to the user. Meanwhile, the verification module is invoked to verify the integrity of the search result using the VO. There can be two possible outcomes. One is that the search result passes the verification and is proved to be correct. The other is that the search result is incomplete or has been tampered with, thereby failing the verification.

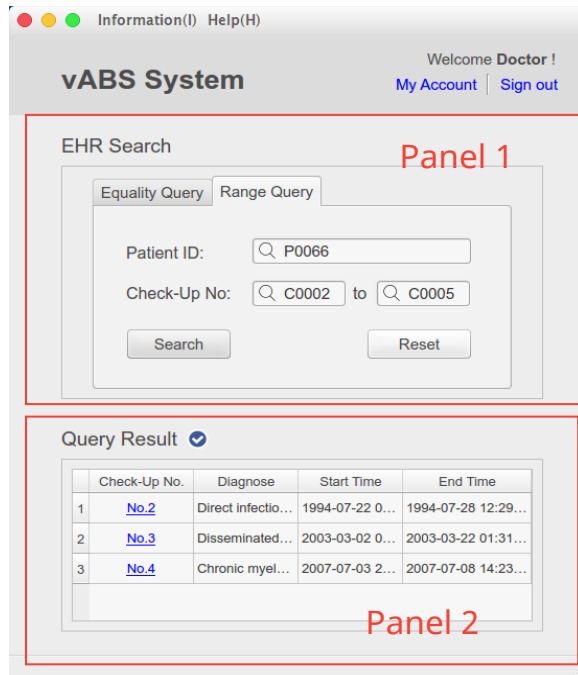
B. Server Side

The server-side modules are implemented in C++ with OpenMP to accelerate query processing. In the default setting, the server employs an AP²G-Tree with APP signatures as authenticated index structure. Once the server receives a search request, the query processing engine will perform a breadth-first search over the AP²G-Tree and check the accessibility by comparing the user's role set against the index nodes' access policies. During the query processing, the server also constructs the VO for the search result. For accessible data records, the APP signature of each record produced by the DO is attached directly. For inaccessible index nodes or data records, the server will generate new APS signatures as part of the VO, as discussed in Sections II-B and II-C.

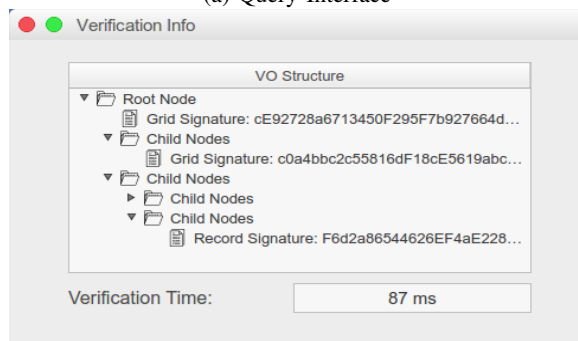
IV. DEMONSTRATION DETAILS

In our demonstration, we use a healthcare dataset from EMRBots [10], which are artificially generated electronic health records. The vABS system supports both equality search and range search over the attributes of patient ID and check-up number. In vABS, the authenticated index structure is built on a 2D data space for both attributes, in which each leaf node corresponds to the medical record of a patient at one check-up. A user can interact with vABS through the following scenarios.

Attribute-Based Search with Access Control. After logging in, users can search medical records through the query interface shown in Panel 1 of Figure 4a. Based on the access





(a) Query Interface



(b) VO Information

Fig. 4: Client Side

policies associated with the data records, different users, depending on their roles, may obtain different search results on Panel 2. They can select a particular record and view its details including the lab test data on a pop-up panel. On the server side, for demonstration purposes, all records in the search range will be highlighted with colored background. As shown in Panel 4 of Figure 5, we mark accessible records with green color and inaccessible ones with red color. Users can also check the access policies of these records.

Result Correctness Verification. In our demonstration, users can verify the search result interactively. A  mark in Panel 2 of Figure 4a indicates a successful verification and a  mark signifies that the search result is unsound or incomplete. By clicking the mark, users can view detailed VO structure and verification time on a pop-up panel (Figure 4b). Moreover, in Panel 4 of Figure 5, users can click the “Attack” button and falsify the data records to emulate the case where the SP tampers with data. In this case, the search result will be verified as incorrect on the client side.

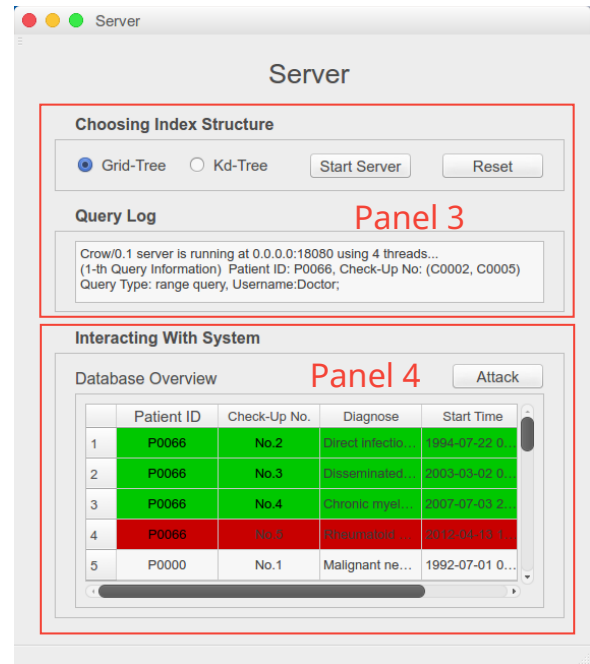


Fig. 5: Server Side

Understanding Performance Trade-off. Our demonstration enables users to observe the search performance of different index structures. By checking a radio button shown in Panel 3 of Figure 5, users can choose to employ either AP²G-Tree or AP²kd-Tree as the authenticated index structure. They can compare the query processing time and result verification time of both structures, and understand the trade-off between search performance and degree of privacy preservation.

ACKNOWLEDGEMENTS

This work was partially supported by HK RGC grants (Grant Nos. 12244916, 12201018, C1008-16G) and National Natural Science Foundation of China (Grant Nos. 61572413, U1636205).

REFERENCES

- [1] C. Xu, J. Xu, H. Hu, and M. H. Au, “When query authentication meets fine-grained access control: A zero-knowledge approach,” in *Proc. SIGMOD*, 2018, pp. 147–162.
- [2] (2018) Microsoft healthvault. [Online]. Available: <https://international.healthvault.com/>
- [3] F. Li, G. Kollios, and L. Reyzin, “Dynamic authenticated index structures for outsourced databases,” in *Proc. SIGMOD*, 2006.
- [4] Q. Chen, H. Hu, and J. Xu, “Authenticated online data integration services,” in *Proc. SIGMOD*, 2015, pp. 167–181.
- [5] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, “Authenticating aggregate queries over set-valued data with confidentiality,” *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 630–644, 2018.
- [6] M. I. Sarfraz, M. Nabeel, J. Cao, and E. Bertino, “DBMask: Fine-grained access control on encrypted relational databases,” in *Proc. CODASPY*, 2015.
- [7] R. C. Merkle, “A Certified Digital Signature,” in *CRYPTO*, 1989.
- [8] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-Policy Attribute-Based Encryption,” in *Proc. IEEE Symposium on Security and Privacy*, 2007.
- [9] H. K. Maji, M. Prabhakaran, and M. Rosulek, “Attribute-Based Signatures,” in *Topics in Cryptology*, 2011.
- [10] (2018) The emrbots website. [Online]. Available: <http://www.emrbots.org/>