Enhancing Data Privacy in the Cloud

Yanbin Lu, Gene Tsudik

Enhancing Data Privacy in the Cloud

Yanbin Lu and Gene Tsudik

University of California, Irvine
{yanbinl, gts}@uci.edu

Abstract. Due to its low cost, robustness, flexibility and ubiquitous nature, cloud computing is changing the way entities manage their data. However, various privacy concerns arise whenever potentially sensitive data is outsourced to the cloud. This paper presents a novel approach for coping with such privacy concerns. The proposed scheme prevents the cloud server from learning any possibly sensitive plaintext in the outsourced databases. It also allows the database owner to delegate users to conducting content-level fine-grained private search and decryption. Moreover, our scheme supports private querying whereby neither the database owner nor the cloud server learns query details. Additional requirement that user's input be authorized by CA can also be supported.

1 Introduction

Cloud computing involves highly available massive compute and storage platforms offering a wide range of services. One of the most popular and basic cloud computing services is storage-as-a-service (SAAS). It provides companies with affordable storage, professional maintenance and adjustable space.

On one hand, due to above-mentioned benefits, companies are excited by the public debut of SAAS. On the other hand, companies are reticent about adopting SAAS. One of the major concerns is the privacy as cloud service is generally provided by the third party. In the following, we call the company, who uses SAAS, the *database owner*. We call anyone who queries the company's database, the database *user*. And we call the cloud servers, which store the database, *the cloud server*. Now we start to clarify different types of privacy challenges during the deployment of cloud service.

From the perspective of the database owner, three challenges arise.

- *Challenge 1:* how to protect outsourced data from theft by hackers or malware infiltrating the cloud server? Encryption by the cloud server and authenticated access by users seems to be a straightforward solution. However, careful consideration should be given to both encryption method and its granularity.
- *Challenge 2:* how to protect outsourced data from abuse by the cloud server? A trivial solution is for the owner to encrypt the database prior to outsourcing. Subsequently, users (armed with the decryption key(s)) can download the entire encrypted database, decrypt it and perform querying *in situ*. Clearly, this negates most benefits of using the cloud. A more elegant approach is to use searchable encryption. Unfortunately, current searchable encryption techniques only support simple search (attribute=value), as opposed to complicated SQL, queries.

– *Challenge 3:* how to realize content-level fine-grained access control for users? This challenge is even harder to solve as it requires variable decryption capabilities for different users. Even trivial solution to the second challenge does not solve this challenge as it gives each user equal decryption capability (same decryption key). An ideal solution would entail the database owner issuing a given user a key that only allows the user to search and decrypt certain records.

From user's perspective, three more challenges arise.

– *Challenge 4:* how to query the cloud server without revealing query details? Learning user's query details means learning user's possibly sensitive search interest. In addition, by learning user queries, the cloud server gradually learns the information in the encrypted database.
– *Challenge 5:* how to hide query contents (e.g., values used in "attribute=value" queries) from the database owner. For the database owner to exercise access control over its outsourced data, a user should first obtain an approval from the database owner over its query contents. However, in some cases, the user may want to get the approval without revealing its query contents even to the database owner. This is the case when the user happens to be a high-level executive who is automatically qualified to search any value and is not willing to reveal query to anyone.
– *Challenge 6:* how to hide query contents while assuring database owner the hiden contents are authorized by some certificate authority (CA). Such challenge surfaces, for example, when the user is FBI who does not want to reveal the person it is investigating while database owner wants to get some confidence by making sure FBI is authorized by the court to do this investigation.

To address the above challenges, we need a scheme for the scenario shown in Fig. 1. In the initial deployment phase, the owner encrypts its database and transfers it to the cloud server. The encryption scheme should guarantee that no plaintext is leaked in the encrypted database, thereby addressing challenges 1–2. When user poses an SQL query, such as:

"select from sample where ((last_name='Lobb' AND birth_date='3/26/1983') OR blood_type='B')"

it first obtains a search token and decryption key from the database owner. Then, the user supplies the search token to the cloud server who uses the token to search the encrypted database. Matching encrypted records are returned to the user who finally decrypts them. The search token and the decryption key should only allow user to search and decrypt records meeting the conditional expression in the specific query, therefore addressing challenge 3. The search token should not reveal the conditional expression specified by user, therefore solving challenge 4. Further, user should be able to get the search token and decryption key without letting database owner know the query contents in order to solve challenge 5. Finally, to solve challenge 6, database owner, even though not knowing the query contents, should be able to verify if these contents are authorized by a CA.

In this paper, we present a new scheme that addresses aforementioned requirements. It relies on attribute-based encryption [1] and blind Boneh-Boyen weak signature scheme [2]. In fact, we amend the standard attribute-based encryption to make

it privately searchable in the cloud computing scenario. Furthermore, we use the blind Boneh-Boyen signature scheme to let user obliviously retrieve a search token and decryption key. Moreover, blind search token and decryption key extraction procedure can be coupled with CA authorization on user's input.

This paper aims to make four contributions: First, we define the adversary and security model for an encryption scheme aimed at the cloud database system. Second, we construct an encryption scheme that protects data privacy and allows access control. Third, we develop techniques for a user to retrieve search token and decryption key from database owner without revealing query contents. Fourth, we make it possible that the database owner, without knowing query contents, can make sure these contents are authorized by CA.

The rest of the paper is organized as follows. Sec. 2 overviews related work. Next, Sec. 3 defines the function and security model. Then, Sec. 4 discusses some background issues. The new scheme is presented in Sec. 5, followed by Sec. 6 that analyzes its performance. An in-depth performance evaluation is shown in Sec. 7. Limitations of our scheme are discussed in Sec. 8. Finally, Sec. 9 concludes this paper. A complete security proof is provided in the full version [3].

2 Related Work

Private Information Retrieval and Oblivious Transfer: Private Information Retrieval (PIR) [4] allows a user to retrieve an item from a server's (public) database without the latter learning which item is being retrieved. While PIR is not concerned with privacy of the server database, Oblivious Transfer (OT) [5] adds an additional requirement that the user should not receive records beyond those requested. Several results [6, 7] apply PIR/OT concepts to relational databases in order to hide user SQL queries from the database server.

There are significant differences between these approaches and our work. First these approaches target a user/server scenario and it is unclear how to extend them to the cloud setting with the additional requirement of protecting data from untrusted cloud server. Second user can query any items inside the database and there is no way to enforce access control in these approaches.

Search on encrypted database: Searching on encrypted data (SoE), also known as privacy preserving keyword-based retrieval over encrypted data, was introduced in the symmetric key setting by Song, et al. [8]. This scheme allows a user to store its symmetrically encrypted data on an untrusted server and later search for a specific keyword by giving the server a search capability, that does not reveal the keyword or any plaintext. Its security and efficiency was later improved in [9] and [10]. Golle, et al. [11] developed a symmetric-key version of SoE that supports conjunctive keyword search. Boneh, et al. [12] later proposed a public-key version of encryption with keyword search (PEKS), where any party in possession of the public key can encrypt and send encryption to an untrusted server, while only the owner of the corresponding private key can generate keyword search capabilities. The server can identify all messages containing the searching keyword, but learn nothing else.

Our work is different from SoE and PEKS since it supports flexible access control (any monotonic access structure) on encrypted data, i.e. the database owner can issue a user a decryption key that only decrypts data meeting a certain conditional expression. Also, our scheme supports oblivious (search token/decryption key) retrieval.

Attribute-based encryption: Sahai and Waters [13] introduced the concept of Attribute-Based Encryption (ABE) where a user's keys and ciphertexts are labeled with sets of descriptive attributes and a particular key can decrypt a particular ciphertext only if the cardinality of the intersection of their labeled attributes exceeds a certain threshold. Later, Goyal, et al. [1] developed a Key-Policy Attribute-Based Encryption (KP-ABE) where the trusted authority (master key owner) can generate user private keys associated with any monotonic access structures consisting of **AND, OR** or threshold gates. Only ciphertexts that satisfy the private key's access structure can be decrypted. Bethencourt, et al. [14] explore the concept of Ciphertext-Policy Attribute-Based Encryption where each ciphertext is associated with an access structure that specifies which type of secret keys can decrypt it. Ostrovsky, et al. [15] extended [1] by allowing negative constraints in a key access structure.

Our scheme is derived from that in [1]. However, compared to traditional ABE, there are several notable differences. First, ABE only achieves payload hiding, i.e., attributes are revealed in plaintext, while our scheme hides the attributes. Second, ABE does not support private search on encrypted data, while our scheme does. Third, ABE does not support oblivious private key retrieval from the authority, while our scheme does.

Predicate encryption: Predicate encryption can be considered as attribute-based encryption supporting attribute-hiding. Ciphertexts are associate with a set of hidden attributes I. The master secret key owner has the fine-grained control over access to encrypted data by generating a secret key sk_f corresponding to predicate f; sk_f can be used to decrypt a ciphertext associated with attribute I if and only if $f(I) = 1$.

Several results have yielded predicate encryption schemes for different predicates. Waters, et al. constructed an equality tests predicate encryption scheme [16]. Shi and Waters [17] constructed a conjunction predicate encryption scheme. In [18], Shi, et al. proposed a scheme for range queries. Boneh and Waters [19] developed a scheme that handles conjunctions and range queries while satisfying a stronger notion of attribute hiding. Katz, et al. [20] move a step further by making predicate encryption support inner products, therefore supporting disjunction and polynomial evaluation.

Our approach is different in several respects. First, no concrete private search scheme exists in predicate encryption. Although a predicate-only version is enough for private search [20], requiring private search on a cloud server and access control for users probably means that two separate implementations of predicate encryption are needed. Second, our scheme supports more flexible access control; although, range queries are not covered. Finally, no oblivious retrieval of decryption key for predicate encryption exists so far.

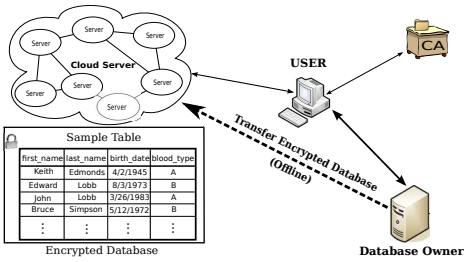

Fig. 1. Cloud storage architecture.

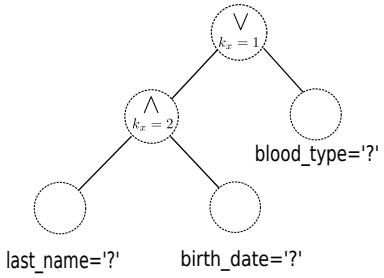

Fig. 2. Access tree example

3 Definition

3.1 Problem Description

Fig. 1 shows the architecture of the envisaged cloud storage scenario. There are four entities: the cloud server ($\mathcal{S}$), the database owner ($\mathcal{DO}$), the database user ($\mathcal{U}$) and the CA ($\mathcal{CA}$). $\mathcal{DO}$'s database table consists of w attributes $\{\alpha_1, \alpha_2, \ldots, \alpha_w\}$. Let $\Omega = \{1, \cdots, w\}$. For ease of description, we assume that every attribute is searchable. Each record m includes w values: $\{v_i\}_{1 \leq i \leq w}$ with each v_i corresponding to attribute α_i. Fig. 1 also illustrates a sample database. The first row describes attribute names and each subsequent row denotes a record.

$\mathcal{U}$ may issue $\mathcal{S}$ any SQL query with monotonic access structure. By monotonic access structure, we mean a boolean formula only involving 'AND/OR' combinations. We use an **access tree** (see Sec. 4.2 for details) to describe any monotonic access structure. In our context, the access tree describes a combination of 'AND/OR' of attribute names, without specifying their values. For example, Fig. 2 depicts one type of access tree corresponding to a conditional expression ((last_name=? AND birth_date=?) OR blood_type=?). If concrete values are supplied together with an access tree, a complete conditional expression can be defined. For example, if a value set (Lobb, 3/26/1983, B) is specified, the expression will be ((last_name='Lobb' AND birth_date='3/26/1983') OR blood_type='B'). We use $\mathcal{T}_\gamma$ to denote an access tree constructed over a subset γ of Ω and use $\mathbf{v}_\gamma$ to describe a set of values for $\mathcal{T}_\gamma$ to completely define a conditional expression. A complete record can be viewed as $\mathbf{v}_\Omega$. We use $\mathcal{T}_\gamma(\mathbf{v}_\gamma, \mathbf{v}_{\gamma'})$ to test whether a set of values $\mathbf{v}_{\gamma'}$ satisfies the conditional expression defined by $\mathcal{T}_\gamma$ and $\mathbf{v}_\gamma$.

Our basic encryption scheme is a set of components: Setup, Encrypt, Extract, Test, Decrypt. Before starting, the CA runs Setup to initialize some parameters. Then $\mathcal{DO}$ runs Encrypt over each record in its table to form an encrypted database. The encrypted database is exported to $\mathcal{S}$ (off-line) and $\mathcal{DO}$ can insert new encrypted items later. Whenever $\mathcal{U}$ forms an SQL query, it runs Extract with $\mathcal{DO}$ to extract a search token and decryption key. Then, $\mathcal{U}$ hands the search token to $\mathcal{S}$ and the latter runs Test over each encrypted record, in order to find matching records. After that. $\mathcal{S}$ sends matching records back and $\mathcal{U}$ runs Decrypt to recover plaintext records. If additional requirement that $\mathcal{DO}$ learns nothing about query content is needed, $\mathcal{U}$ can run BlindExtract instead of Extract with $\mathcal{DO}$. If further requirement that $\mathcal{U}$'s query should be

Authorized by CA is needed, $\mathcal{U}$ can engage in AuthorizedBlindExtract with $\mathcal{DO}$. We define each function in more detail below.

3.2 Basic Scheme Definition

The basic scheme includes following components:

Setup(1^k): on input a security parameter 1^k, outputs parameters $params$, $\mathcal{DO}$'s master key $msk_{\mathcal{DO}}$.

Encrypt($\mathcal{DO}(params, msk_{\mathcal{DO}}, \mathbf{v}_\Omega)$): $\mathcal{DO}$ on input $params, msk_{\mathcal{DO}}$ and a record $\mathbf{v}_\Omega$, outputs a ciphertext.

Extract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{DO}(params, msk_{\mathcal{DO}})$): $\mathcal{U}$ on input $(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma)$ and $\mathcal{DO}$ on input $(params, msk_{\mathcal{DO}})$ engage in an interactive protocol. At the end, $\mathcal{U}$ outputs a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and a decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$, and $\mathcal{DO}$ outputs $(\mathcal{T}_\gamma, \mathbf{v}_\gamma)$.

Test($\mathcal{S}(params, tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}, C)$): $\mathcal{S}$ on input parameters $params$, a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and a ciphertext $C = \mathsf{Encrypt}(msk_{\mathcal{DO}}, \mathbf{v}'_\Omega)$, outputs "yes" if $\mathcal{T}_\gamma(\mathbf{v}_\gamma, \mathbf{v}'_\Omega) = 1$ and "no" otherwise.

Decrypt($\mathcal{U}(params, tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}, C)$): $\mathcal{U}$ on input $params$, a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$, a decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and a ciphertext $C = \mathsf{Encrypt}(msk_{\mathcal{DO}}, \mathbf{v}'_\Omega)$, outputs $\mathbf{v}'_\Omega$ if $\mathcal{T}_\gamma(\mathbf{v}_\gamma, \mathbf{v}'_\Omega) = 1$ and $\perp$ otherwise.

3.3 Blind Extraction Definition

In order to protect $\mathcal{U}$'s query from $\mathcal{DO}$, we need to replace Extract with a blinded version, called BlindExtract.

BlindExtract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{DO}(params, msk_{\mathcal{DO}})$): $\mathcal{U}$ on input $(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma)$ and $\mathcal{DO}$ on input $(params, msk_{\mathcal{DO}}, \mathcal{T}_\gamma)$ engage in an interactive protocol. $\mathcal{U}$'s output is a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and a decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$, and $\mathcal{DO}$'s output is $\mathcal{T}_\gamma$.

Sometimes, it makes more sense to require $\mathcal{U}$ to prove that its input in BlindExtract is authorized by a CA before $\mathcal{U}$ can get anything useful. In order to realize that, we introduce two other functions Authorize and AuthorizedBlindExtract. Authorize helps a $\mathcal{U}$ get a commitment ψ and a signature σ from a CA. In AuthorizedBlindExtract, $\mathcal{DO}$ is provided with $\mathcal{T}_\gamma, \psi, \sigma$ while $\mathcal{U}$ can prove statements about commitment ψ using zero-knowledge proof.

Authoriz($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{CA}(params, msk_{\mathcal{CA}})$): $\mathcal{CA}$ generates a commitment ψ over $\mathcal{U}$'s input $(\mathcal{T}_\gamma, \mathbf{v}_\gamma)$, the randomness $open$ used to compute ψ and a signature σ over ψ. $\mathcal{CA}$'s output is $(\mathcal{T}_\gamma, \mathbf{v}_\gamma, \psi, open, \sigma)$. $\mathcal{U}$'s output is $(\psi, open, \sigma)$.

AuthorizedBlindExtract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma, \psi, open, \sigma), \mathcal{DO}(params, msk_{\mathcal{DO}})$): $\mathcal{U}$ on input $(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma, \psi, open, \sigma)$ and $\mathcal{DO}$ on input $(params, msk_{\mathcal{DO}})$ engage in an interactive protocol. $\mathcal{DO}$'s output is $(\mathcal{T}_\gamma, \psi, \sigma)$. If $\psi = \mathsf{Commit}((\mathcal{T}_\gamma, \mathbf{v}_\gamma), open)$ and $\mathsf{Vrfy}_{pk_{\mathcal{CA}}}(\psi, \sigma) = 1$, $\mathcal{U}$'s output is a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and a decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$, and otherwise, $\mathcal{U}$ outputs $\perp$.

3.4 Adversary Model and Security Requirement

In this paper, we assume the malicious adversary model (as opposed to semi-honest, aka "honest-but-curious") . A malicious adversary can arbitrarily deviate from the prescribed protocols. We also assume that $\mathcal{U}$ may collude with $\mathcal{S}$. However, $\mathcal{DO}$ does not collude with any party. In the full version of this paper [3], we will prove our scheme is secure against malicious adversary according to Def. 1, 2 and 3.

For the basic scheme, we define adversary's advantage by defining a security game under chosen plaintext attack in a selective set model, similar to [1].

Definition 1. (Selective-Set Secure (IND-SS-CPA)). *Let k be a security parameter. Above scheme is IND-SS-CPA-secure if every p.p.t. adversary $\mathcal{A}$ has an advantage negligible in k for the following game: (1) Run* $\mathsf{Setup}(1^k)$ *to obtain $(params, msk_{\mathcal{DO}})$, and give $params$ to $\mathcal{A}$. (2) $\mathcal{A}$ outputs two records m_1, m_2 to be challenged on (3) $\mathcal{A}$ may query an oracle $\mathcal{O}_{\mathsf{Extract}}(params, msk_{\mathcal{DO}}, \mathcal{T}_\gamma, \mathbf{v}_\gamma)$ such that $\mathcal{T}_\gamma(\mathbf{v}_\gamma, m_1) \neq 1$ and $\mathcal{T}_\gamma(\mathbf{v}_\gamma, m_2) \neq 1$. (4) Select a random bit b and give $\mathcal{A}$ the challenge $c^* \leftarrow$* $\mathsf{Encrypt}(params, msk_{\mathcal{DO}}, m_b)$. *(5) $\mathcal{A}$ may continue to query oracle $\mathcal{O}_{\mathsf{Extract}}(\cdot)$ under the same conditions as before. (6) $\mathcal{A}$ outputs a bit b'. We define $\mathcal{A}$'s advantage in the above game as $|\Pr[b' = b] - 1/2|$.*

BlindExtract must satisfy two security properties: *Leak-free Extract* [21] and *Selective-failure Blindness* [22]. Informally, the former means that a malicious $\mathcal{U}$ cannot learn more by executing the BlindExtract with an honest $\mathcal{DO}$ than by executing Extract with an honest $\mathcal{DO}$. Whereas, Selective-failure Blindness means that a malicious $\mathcal{DO}$ cannot learn anything about $\mathcal{U}$'s choice of $\mathbf{v}_\gamma$ during BlindExtract. Moreover, $\mathcal{DO}$ cannot cause BlindExtract to fail based on $\mathcal{U}$'s choice. Now we formally define *Leak-free Extract* and *Selective-failure Blindness*:

Definition 2. (Leak-Free Extract). BlindExtract *protocol is leak free if, for all p.p.t. adversaries $\mathcal{A}$, there exists an efficient simulator such that for every value k, $\mathcal{A}$ cannot determine whether it is playing Game Real or Game Ideal with non-negligible advantage, where*

Game Real: Run $\mathsf{Setup}(1^k)$. *As many times as $\mathcal{A}$ wants, $\mathcal{A}$ chooses its $\mathcal{T}_\gamma, \mathbf{v}_\gamma$ and executes* BlindExtract$(\cdot)$ *with $\mathcal{DO}$.*

Game Ideal: Run $\mathsf{Setup}(1^k)$. *As many times as $\mathcal{A}$ wants, $\mathcal{A}$ chooses its $\mathcal{T}_\gamma, \mathbf{v}_\gamma$ and executes* BlindExtract$(\cdot)$ *with a simulator which does not know $msk_{\mathcal{DO}}$ and only queries a trusted party to obtain $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$.*

Definition 3. (Selective-Failure Blindness). BlindExtract *is selective-failure blind if every p.p.t. adversary $\mathcal{A}$ has a negligible advantage in the following game: First, $\mathcal{A}$ outputs params and a pair of $(\mathcal{T}, \mathbf{v}_1)$, $(\mathcal{T}, \mathbf{v}_2)$. A random bit b is chosen. $\mathcal{A}$ is given black-box access to two oracles $\mathcal{U}(params, \mathcal{T}, \mathbf{v}_b)$ and $\mathcal{U}(params, \mathcal{T}, \mathbf{v}_{1-b})$. The $\mathcal{U}$ algorithm produces local output $s_b = (tk_{(\mathcal{T}, \mathbf{v}_b)}, sk_{(\mathcal{T}, \mathbf{v}_b)})$ and $s_{1-b} = (tk_{(\mathcal{T}, \mathbf{v}_{1-b})}, sk_{(\mathcal{T}, \mathbf{v}_{1-b})})$ respectively. If $s_b \neq \perp$ and $s_{1-b} \neq \perp$ then $\mathcal{A}$ receives (s_0, s_1). If $s_b = \perp$ and $s_{1-b} \neq \perp$ then $\mathcal{A}$ receives $(\perp, \epsilon)$. If $s_b \neq \perp$ and $s_{1-b} = \perp$, then $\mathcal{A}$ receives $(\epsilon, \perp)$. If $s_b = \perp$ and $s_{1-b} = \perp$, then $\mathcal{A}$ receives $(\perp, \perp)$. Finally, $\mathcal{A}$ outputs its guess bit b'. We define $\mathcal{A}$'s advantage in the above game as $|\Pr[b' = b] - 1/2|$.*

4 Preliminaries

4.1 Notation

Let $\{0,1\}^l$ denote the set of integers of maximum length l, i.e. the set $[0, 2^l - 1]$ of integers. we employ the security parameters $l_\phi, l_\mathcal{H}$ where l_ϕ (80) is the security parameter controlling the statistical zero-knowledge property, $l_\mathcal{H}$ (160) is the output length of the hash function used for the Fiat-Shamir heuristic. $\mathcal{H}(\cdot)$ and $\mathcal{H}'(\cdot)$ denote two distinct hash function. We use Enc_{pk}^{hom} and Dec_{sk}^{hom} to denote homomorphic encryption and decryption (respectively) under public key pk (or secret key sk). We use Enc_{k}^{sym} and Dec_{k}^{sym} to denote symmetric encryption and decryption under key k. We define Lagrange Coefficient as $\Delta_{i,S} = \prod_{j \in S, j \neq i} \frac{j}{j-i}$. Let Ω denote attributes index set, i.e. $\Omega = \{1, \cdots, w\}$. $\mathcal{DO}$'s private and public keys are $sk_{\mathcal{DO}}$ and $pk_{\mathcal{DO}}$, respectively. $server$'s master key is $msk_{\mathcal{DO}}$. $\mathcal{CA}$'s private and public keys are $sk_{\mathcal{CA}}$ and $pk_{\mathcal{CA}}$.

4.2 Access Tree

We use $\mathcal{T}$ to denote a tree representing an access structure. $\mathcal{T}$ represents a combination of 'AND/OR' of attribute names without specifying their values, as shown in Fig. 2. An access structure $\mathcal{T}_\gamma$ defined over a set γ of attributes, coupled with a set of values $\mathbf{v}_\gamma$ defined over the same set, completely defines a conditional expression (See Sec. 3.1 for example). We use $\mathcal{T}_\gamma(\mathbf{v}_\gamma, \mathbf{v})$ to test whether another set of values $\mathbf{v}$ satisfies the condition defined by $\mathcal{T}_\gamma$ and $\mathbf{v}_\gamma$. Each non-leaf node represents a threshold gate, described by its children and a threshold value. Let num_x be the number of children of a node x. The threshold value associated with node x is denoted by k_x that is either 1 or num_x, depending on the threshold gate. In case of an OR gate, $k_x = 1$; in case of an AND gate, $k_x = num_x$. Each leaf node x is described by an attribute with a threshold $k_x = 1$. Standard tree data structures can be used to represent and store $\mathcal{T}$. Since $\mathcal{T}_\gamma$ is exposed to $\mathcal{S}$ in Test, to prevent $\mathcal{S}$ from learning database schema, each leaf node can store an attribute index instead of the attribute name.

To facilitate working with the access trees, we define a few functions. We denote the parent of the node x as $parent(x)$. $node(\alpha_i)$ returns the leaf node corresponding to attribute α_i. $attr(x)$ is defined only if x is a leaf node; it returns the attribute index i of α_i associated with x. Access tree $\mathcal{T}$ also defines an ordering between the children of every node, i.e. each child y of a node x are numbered from 1 to num_x. $index(y)$ returns this number associated with the node y. Let S_x denote a set $[1, \ldots, num_x]$. Finally, let $child_i(x)$ return the ith child of node x.

We also define $\Gamma_{\mathcal{T}_\gamma}$ as a set of minimum subsets of γ that satisfies $\mathcal{T}_\gamma$. By "minimum", we mean the subset cannot become smaller while still satisfying $\mathcal{T}_\gamma$. For example, in Fig. 2, $\Gamma_{\mathcal{T}_\gamma} = \{\{1,2\},\{3\}\}$ where $1, 2, 3$ is the index of attribute $last\_name$, $birth\_date, blood\_type$ respectively. Here $\Gamma_{\mathcal{T}_\gamma}$ means that either $\{last\_name$, $birth\_date\}$ or $\{blood\_type\}$ can satisfy $\mathcal{T}_\gamma$. We can determine $\Gamma_{\mathcal{T}_\gamma}$ in a down-top manner. For each leaf node, define $\mathbb{S}_x = \{attr(x)\}$. For any other node x, $\mathbb{S}_x = \cup_{i \in S_x} \mathbb{S}_{child_i(x)}$ if $k_x = 1$. Otherwise if $k_x > 1$, $\mathbb{S}_x = \{x' : x' = \cup_{1 \leq i \leq k_x} x'_i, \forall x'_i \in \mathbb{S}_{child_i(x)}\}$. And the resulting $\mathbb{S}_r$ at root node r is $\Gamma_{\mathcal{T}_\gamma}$. For $\gamma' \in \Gamma_{\mathcal{T}_\gamma}$, we define $\mathcal{T}_{\gamma'}$ as a subgraph of $\mathcal{T}_\gamma$ with only attributes in γ' as leaves. For example, in Fig. 2, if $\gamma' = \{1,2\}$,

then $\mathcal{T}_{\gamma'}$ would be the left-hand subtree of the root node. Note in $\mathcal{T}_{\gamma'}$ each non-leaf node x's k_x should be its number of children, i.e., a conjunctive gate, since γ' is a minimum satisfiable subset.

4.3 Homomorphic Encryption

There are several additively homomorphic public key encryption schemes [23, 24]. We elect to use Paillier encryption [24] due to its easy implementation and amenability to proofs of knowledge. Let n denote an RSA modulus, $\mathsf{h} = \mathsf{n} + 1$ and g be an element of order $\phi(\mathsf{n}) \bmod \mathsf{n}^2$. Let $sk = \{\phi(\mathsf{n})\}$ and $pk = \{\mathsf{g}, \mathsf{n}\}$. Encryption is defined as $c = \mathsf{Enc}_{pk}^{hom}(m) = \mathsf{h}^m \mathsf{g}^r \bmod \mathsf{n}^2$ where $r \in_R \mathbb{Z}_{\phi(\mathsf{n})}$. Corresponding decryption is defined as: $\mathsf{Dec}_{sk}^{hom}(c) = \left[\frac{\left(c^{\phi(\mathsf{n})} \bmod \mathsf{n}^2\right)-1}{\mathsf{n}} \cdot \phi(\mathsf{n})^{-1} \bmod N \right]$. Note that, to encrypt, we use $\mathsf{h}^m \mathsf{g}^r$ instead of standard $\mathsf{h}^m r^{\mathsf{n}}$. If the order of g has no factor of n and is greater than 2, g^r is a random element from the same subgroup as r^{n}. Therefore $\mathsf{h}^m \mathsf{g}^r$ has the same distribution as $\mathsf{h}^m r^{\mathsf{n}}$. The purpose of using the former is to facilitate zero-knowledge proofs.

4.4 Zero-Knowledge Proof

Our scheme uses various protocols to prove knowledge of, and relations among, discrete logarithms. To describe these protocols, we use the notation introduced by Camenisch and Stadler [25]. For instance, $PK\{(a, b, c) : y = g^a h^b \wedge \mathfrak{y} = \mathfrak{g}^a \mathfrak{h}^c \wedge s \leq b \leq t\}$ denotes a zero-knowledge proof of knowledge of integers a, b, c such that $y = g^a h^b$ and $\mathfrak{y} = \mathfrak{g}^a \mathfrak{h}^c$ holds and $s \leq b \leq t$. The convention is that everything inside parentheses is only known to the prover, while all other parameters are known to both prover and verifier.

The technique for a proof of knowledge of a representation of an element $y \in G$ with respect to several bases $z_1, \ldots, z_v \in G$, i.e., $PK\{(a_1, \cdots, a_v) : y = z_1^{a_1} \cdots z_v^{a_v}\}$, is presented in [26]. A proof of equality of discrete logarithms of two group elements $y_1, y_2 \in G$ to bases $g \in G$ and $h \in G$, respectively, i.e., $PK\{(a) : y_1 = g^a \wedge y_2 = h^a\}$, is given in [27]. Generalizations to proving equalities among representations of elements $y_1, \ldots, y_v \in G$ to bases $g_1, \ldots, g_v \in G$ are straightforward [25]. Boudot [28] demonstrates proof of knowledge of a discrete logarithm of $y \in G$ with respect to $g \in G$ such that $\log_g y$ lies in integer interval $[s, t]$, i.e., $PK\{(a) : y = g^a \wedge a \in [s, t]\}$ under the strong RSA assumption and the assumption that the prover does not know the factorization of the RSA modulus.

4.5 Bilinear map

We now review some general notions about efficiently computable bilinear maps.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order q. Let g be a generator of $\mathbb{G}_1$ and $\hat{e}$ be a bilinear map, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. The bilinear map $\hat{e}$ has the following properties:

1. Bilinearity: for all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, we have $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$

2. Non-degeneracy: $\hat{e}(g, g) \neq 1$.

We say that $\mathbb{G}_1$ is a bilinear group if the group operation in $\mathbb{G}_1$ and the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ are both efficiently computable.

4.6 Cryptographic Assumption

Our scheme's security is based on the decisional bilinear Diffie-Hellman (BDH) assumption [29] and Boneh-Boyen Hidden Strong Diffie-Hellman (BB-HSDH) assumption [30].

Assumption 1 (Decisional Bilinear Diffie-Hellman (BDH) assumption.) *Let $a, b, c, z \in \mathbb{Z}_q$ be chosen at random and g be a generator of $\mathbb{G}_1$. We say that the BDH problem is hard if for all p.p.t. adversaries $\mathcal{A}$ there exists a negligible function* negl *such that $|Pr[\mathcal{A}(g^a, g^b, g^c, \hat{e}(g, g)^{abc}) = 1] - Pr[\mathcal{A}(g^a, g^b, g^c, \hat{e}(g, g)^z) = 1]| \leq \mathsf{negl}(n)$ where in each case the probabilities are taken over the random choice of the generator g, the random choice of a, b, c, z in $\mathbb{Z}_q$ and the random bits consumed by $\mathcal{A}$.*

Assumption 2 (Boneh-Boyen Hidden Strong Diffie-Hellman (BB-HSDH)). *Let x, $c_1, \cdots c_t \in_R \mathbb{Z}_q$. On input $g, g^x, u \in \mathbb{G}_1, h, h^x \in \mathbb{G}_2$ and the tuple $\{g^{1/(x+c_l)}, c_l\}_{l=1\ldots t}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

5 Scheme

We present our scheme Π which consists of following algorithms.

Setup(1^k): Run $\mathcal{G}(1^k)$ to obtain $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, \mathsf{n}, \mathsf{g}, \mathfrak{n}, \mathfrak{g}, \mathfrak{h})$. n is an RSA modulus larger than $2^k q^2$ with generator g. Let $sk_{\mathcal{DO}} = \phi(\mathsf{n})$ and $pk_{\mathcal{DO}} = \{\mathsf{g}, \mathsf{n}\}$. In other words, only $\mathcal{DO}$ knows the factors of n. $\mathfrak{n}$ is another RSA modulus with generator $\mathfrak{g}$ and $\mathfrak{h}$. Note neither factors of n nor $\log_{\mathfrak{g}} \mathfrak{h}$ is known to any party. Pick secret parameters t, t', y, y' which are only known to $\mathcal{DO}$. Make $Y = \hat{e}(g, g)^y$, $Y' = \hat{e}(g, g)^{y'}$, $T = g^t$, $T' = g^{t'}$, $e_t = \mathsf{Enc}^{hom}_{pk_{\mathcal{DO}}}(t)$, $e_{t'} = \mathsf{Enc}^{hom}_{pk_{\mathcal{DO}}}(t')$, and π^s proving e_t and $e_{t'}$ are well formed. Output $params \leftarrow (Y, Y', T, T', e_t, e_{t'}, \pi^s, pk_{\mathcal{DO}}, pk_{\mathcal{CA}}, \mathfrak{n}, \mathfrak{g}, \mathfrak{h})$, $msk_{\mathcal{DO}} \leftarrow (t, t', y, y', sk_{\mathcal{DO}})$.

Encrypt($\mathcal{DO}(params, msk_{\mathcal{DO}}, m)$): To encrypt a record $m = \mathbf{v}_\Omega = \{v_1, \ldots, v_w\}$, $\mathcal{DO}$ chooses random values $s, s' \in_R \mathbb{Z}_q$ and outputs the ciphertext as:

$$C = (E, E', \{E_i, E_i'\}_{i \in \Omega}) .$$

where $E = \mathsf{Enc}^{sym}_{Y^s}(m)$, $E' = Y'^{s'}$, $E_i = g^{s \cdot (t + \mathcal{H}(i, v_i))}$ and $E_i' = g^{s' \cdot (t' + \mathcal{H}'(i, v_i))}$.

Extract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{DO}(params, msk_{\mathcal{DO}})$): This is an interactive protocol between $\mathcal{U}$ and $\mathcal{DO}$.

1. $\mathcal{U}$ chooses an attribute set γ and constructs $\mathcal{T}_\gamma$ and $\mathbf{v}_\gamma$ to fully define a conditional expression it wants to query. Then it submits $\mathcal{T}_\gamma$ and $\mathbf{v}_\gamma$ to $\mathcal{DO}$.

2. $\mathcal{DO}$ defines a polynomial $Q_x(\cdot)$ of degree $k_x - 1$ for each node x in $\mathcal{T}_\gamma$ in a top-down manner. For the root node r, it sets $Q_r(0) = y$ and $k_r - 1$ other points of Q_r randomly to fully define $Q_r(\cdot)$. For any other node x, it sets $Q_x(0) = Q_{parent(x)}(index(x))$ and chooses $k_x - 1$ other points randomly to completely define $Q_x(\cdot)$. Then it outputs decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} = \{\{sk_i\}_{i\in\gamma}, \mathcal{T}_\gamma, \mathbf{v}_\gamma\}$ where $sk_i = g^{Q_{node(\alpha_i)}(0)/(t+\mathcal{H}(i,v_i))}$. $\mathcal{DO}$ defines $Q'_x(\cdot)$ in the same way as $Q_x(\cdot)$ except that $Q'_r(0) = y'$. And it outputs search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} = \{\{tk_i\}_{i\in\gamma}, \mathcal{T}_\gamma\}$ where $tk_i = g^{Q'_{node(\alpha_i)}(0)/(t'+\mathcal{H}'(i,v_i))}$. Last, $\mathcal{DO}$ sends $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ and $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ to $\mathcal{U}$.

Test($\mathcal{S}(params, tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}, C)$): To test whether an encrypted record $C = \mathsf{Encrypt}(msk_{\mathcal{DO}}, \mathbf{v}'_\Omega)$ matches a search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} = \{\{tk_i = g^{Q'_{node(\alpha_i)}(0)/(t'+\mathcal{H}'(i,v_i))}\}_{i\in\gamma}, \mathcal{T}_\gamma\}$, it first calculates $\Gamma_{\mathcal{T}_\gamma}$ from $\mathcal{T}_\gamma$. The search operation starts from the first $\gamma' \in \Gamma_{\mathcal{T}_\gamma}$. Let $i = attr(x)$. For each node x in $\mathcal{T}_{\gamma'}$, it computes a value z_x in a down-top manner. For each leaf node x in $\mathcal{T}_{\gamma'}$, $\mathcal{S}$ computes $z_x = \hat{e}(tk_i, E'_i)$. We use v'_i to denote the value embedded in E'_i. Note if $v_i = v'_i$, $z_x = \hat{e}(g^{Q'_x(0)/(t'+\mathcal{H}'(i,v_i))}, g^{s'\cdot(t'+\mathcal{H}'(i,v'_i))}) = \hat{e}(g,g)^{s'\cdot Q'_x(0)}$. For each non-leaf node x, it sets $z_x = \prod_{i\in S_x}(z_{child_i(x)})^{\Delta_{i,S_x}}$. Note if $\{v_i = v'_i\}_{i\in\gamma'}$, $z_x = \prod_{i\in S_x}(\hat{e}(g,g))^{s'\cdot Q'_{child_i(x)}(0)\cdot\Delta_{i,S_x}} = \prod_{i\in S_x}(\hat{e}(g,g))^{s'\cdot Q'_x(i)\cdot\Delta_{i,S_x}} = \hat{e}(g,g)^{s'\cdot Q'_x(0)}$. The procedure continues until it reaches the root node r. If $z_r = E'$, $\mathcal{S}$ outputs 'yes'. Otherwise, it continues to test the next γ'. If all γ's do not meet the criteria, it outputs 'no'.

Decrypt($\mathcal{U}(params, tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}, sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}, C)$): The decryption algorithm first identifies γ' satisfying $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$ as **Test** algorithm does. Note this step can be omitted if γ' is provided as input after it is identified by **Test**. Then it follows a down-top manner in $\mathcal{T}_{\gamma'}$. Let $i = attr(x)$. Then for each leaf node $x \in \mathcal{T}_{\gamma'}$, it computes $z_x = \hat{e}(sk_i, E_i)$. Note since v_i equals to v'_i, $z_x = \hat{e}(g^{Q_x(0)/(t_i+t\cdot v_i)}, g^{s(t_i+t\cdot v'_i)}) = \hat{e}(g,g)^{s\cdot Q_x(0)}$. For non-leaf node $x \in \mathcal{T}_{\gamma'}$, it computes $z_x = \prod_{i\in S_x}(z_{child_i(x)})^{\Delta_{i,S_x}} = \prod_{i\in S_x}(\hat{e}(g,g))^{s\cdot Q_{child_i(x)}(0)\cdot\Delta_{i,S_x}}$ $= \prod_{i\in S_x}(\hat{e}(g,g))^{s\cdot Q_x(i)\cdot\Delta_{i,S_x}} = \hat{e}(g,g)^{s\cdot Q_x(0)}$. The procedure continues until it reaches root r and $z_r = \hat{e}(g,g)^{s\cdot Q_r(0)} = \hat{e}(g,g)^{s\cdot y} = Y^s$ is computed. Then user recovers $m = \mathsf{Dec}^{sym}_{\mathcal{H}(Y^s)}(E)$.

BlindExtract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{DO}(params, msk_{\mathcal{DO}})$):

1. $\mathcal{U}$ first verifies π^s. If π^s passes verification, then the user chooses $r_{i,1}, r'_{i,1} \in_R \mathbb{Z}_q$ and $r_{i,2}, r'_{i,2} \in_R [0, \ldots, 2^k q]$ and computes

$$e_i = ((e_t \oplus \mathsf{Enc}^{hom}_{pk_s}(\mathcal{H}(i, v_i))) \otimes r_{i,1}) \oplus \mathsf{Enc}^{hom}_{pk_s}(r_{i,2}\cdot q), \forall i \in \gamma$$

$$e'_i = ((e_{t'} \oplus \mathsf{Enc}^{hom}_{pk_s}(\mathcal{H}'(i, v_i))) \otimes r'_{i,1}) \oplus \mathsf{Enc}^{hom}_{pk_s}(r'_{i,2}\cdot q), \forall i \in \gamma$$

   It also computes a zero-knowledge proof π^c proving e_i, e'_i are well formed and $r_{i,1}, r_{i,2}, r'_{i,1}, r'_{i,2}$ are in appropriate interval. Then it sends $\{e_i, e'_i\}_{i\in\gamma}$, $\mathcal{T}_\gamma$, π^c to $\mathcal{DO}$.
2. $\mathcal{DO}$ verifies π^c to make sure $e_i, e'_i, r_{i,1}, r'_{i,1}, r_{i,2}, r'_{i,2}$ are correctly embedded. Then $\mathcal{DO}$ starts to define a polynomial $Q_x(\cdot)$ of degree $k_x - 1$ for each node x in $\mathcal{T}_\gamma$

in a top-down manner. For the root node r, it sets $Q_r(0) = y$ and $k_r - 1$ other points of Q_r randomly to fully define Q_r. For any other node x, set $Q_x(0) = Q_{parent(x)}(index(x))$ and choose $k_x - 1$ other points randomly to completely define Q_x. $\mathcal{DO}$ defines another polynomial $Q'_x(\cdot)$ in the same way as $Q_x(\cdot)$ except that $Q'_x(0) = y'$. Next, for each $i \in \gamma$, $\mathcal{DO}$ decrypts $d_i = \mathsf{Dec}^{hom}_{sk_{\mathcal{DO}}}(e_i), d'_i = \mathsf{Dec}^{hom}_{sk_{\mathcal{DO}}}(e'_i)$ and sends $a_i = g^{Q_{node(\alpha_i)}(0)/d_i}$ and $a'_i = g^{Q'_{node(\alpha_i)}(0)/d'_i}$ to $\mathcal{U}$.

3. $\mathcal{U}$ computes $sk_i = {a_i}^{r_{i,1}} = g^{Q_{node(\alpha_i)}(0)/(t+\mathcal{H}(i,v_i))}$ and $tk_i = {a'_i}^{r'_{i,1}} = g^{Q'_{node(\alpha_i)}(0)/(t'+\mathcal{H}'(i,v_i))}$ for $i \in \gamma$. Then $\mathcal{U}$ checks the validity of sk_is. To do that, it computes $p_i = e(sk_i, T \cdot g^{\mathcal{H}(i,v_i)}) = e(g,g)^{Q_{node(\alpha_i)}(0)}$ for all $i \in \gamma$. After that, it starts to compute a value q_x for each node x in $\mathcal{T}_\gamma$ in a down-top manner starting from leaves. For each leaf node x in $\mathcal{T}_\gamma$, its q_x is set to $p_{attr(x)}$. For a non-leaf node x, q_x is dependent on k_x. If $k_x = 1$, user first verifies that each $q_{child_i(x)}$, for all $i \in S_x$, is the same. Then it sets $q_x = q_{child_i(x)}$, for arbitrary $i \in S_x$. If $k_x > 1$, it sets $q_x = \prod_{i \in S_x} (q_{child_i(x)})^{\Delta_{i,S_x}}$. The procedure continues until it reaches the root node r. Finally, the user checks whether $q_r \stackrel{?}{=} Y$. If any above verification fails, $\mathcal{U}$ quits. $\mathcal{U}$ checks tk_i in the same way as it does sk_i except that q_r should be equal to Y' this time. $\mathcal{U}$ outputs decryption key $sk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} = \{\{sk_i\}_{i\in\gamma}, \mathcal{T}_\gamma, \mathbf{v}_\gamma\}$ and search token $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)} = \{\{tk_i\}_{i\in\gamma}, \mathcal{T}_\gamma\}$.

Authorize($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma), \mathcal{CA}(params, sk_{\mathcal{CA}})$): $\mathcal{U}$ submits $\mathcal{T}_\gamma, \mathbf{v}_\gamma$ to $\mathcal{CA}$. $\mathcal{CA}$ verifies that $\mathcal{U}$ has the right to search for the conditional expression defined by $(\mathcal{T}_\gamma, \mathbf{v}_\gamma)$. If it approves user request, then $\mathcal{CA}$, on $\mathcal{U}$'s behalf, makes pedersen commitments c_{v_i}, c'_{v_i} on each $v_i \in \mathbf{v}_\gamma$, i.e. $c_{v_i} = \mathfrak{g}^{\mathcal{H}(i,v_i)}\mathfrak{h}^{r_{v_i}}$ and $c'_{v_i} = \mathfrak{g}^{\mathcal{H}'(i,v_i)}\mathfrak{h}^{r'_{v_i}}$. Next, $\mathcal{CA}$ maps $\mathcal{T}_\gamma$ to a Merkle hash tree. Specifically, it computes a hash value for each node x in $\mathcal{T}_\gamma$. For each leaf node x, its hash value is $h_x = \mathcal{H}(k_x)$. For non-leaf node, its hash value is defined as the hash of concatenations of its k_x and its children's hash values, i.e. $h_x = \mathcal{H}(k_x || h_{child_1(x)} || \cdots || h_{child_{num_x}(x)})$. Let h_r denote the hash value for the root node r. CA issues a signature σ on h_r and $\{c_{v_i}, c'_{v_i}\}_{i\in\gamma}$, i.e. $\sigma = \mathsf{Sign}_{sk_{\mathcal{CA}}}(h_r, \{c_{v_i}, c'_{v_i}\}_{i\in\gamma})$, and send $\{\{r_{v_i}, c_{v_i}, r'_{v_i}, c'_{v_i}\}_{i\in\gamma}, \sigma\}$ back to $\mathcal{U}$.

AuthorizedBlindExtract($\mathcal{U}(params, \mathcal{T}_\gamma, \mathbf{v}_\gamma, \psi, open, \sigma)$,
$\mathcal{DO}(params, msk_{\mathcal{DO}})$): This protocol is detailed in [3]. Here $\psi = \{c_{v_i}, c'_{v_i}\}_{i\in\gamma}$ and $open = \{r_{v_i}, r'_{v_i}\}_{i\in\gamma}$. The protocol basically follows the **BlindExtract** protocol except that $\mathcal{U}$ needs to prove statements about commitments using zero-knowledge proof.

6 Performance Analysis

Before presenting performance analysis, we point out two possible improvements to the scheme. First, in Test algorithm, if the identified matching set γ' is sent to $\mathcal{U}$, then Decrypt algorithm does not need search token to seek γ' again. Second, as pointed out in [1], instead of exponentiating at each level during the computation of z_x in Decrypt, for each leaf node in γ', we can keep track of which Lagrange coefficient is multiplied with each other. Using this, we can compute the final exponent f_x for each leaf node $x \in \mathcal{T}_{\gamma'}$ by doing multiplication in $\mathbb{Z}_q$. Now z_r is simply $\prod_{i\in\gamma'} \hat{e}(sk_i, E_i)^{f_{node(\alpha_i)}}$. The same optimization applies to Test algorithm.

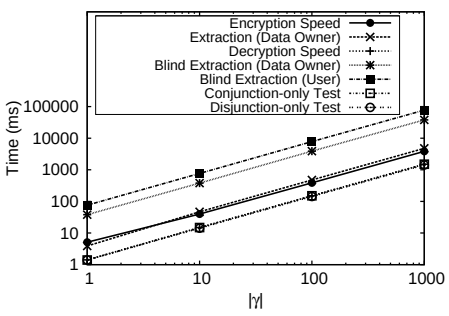

Fig. 3. Performance of Encryp, Extract, Decrypt vs. number of attributes.

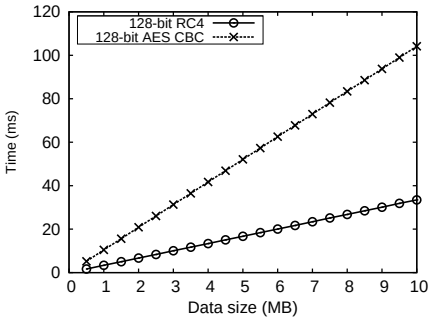

Fig. 4. Symmetric encryption overhead.

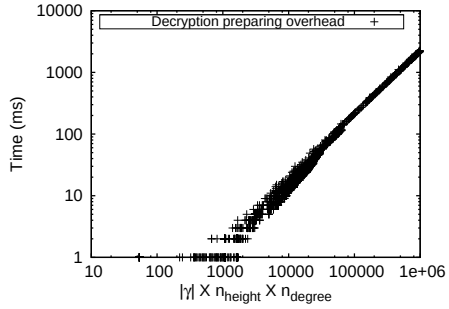

Fig. 5. Decryption preparing time.

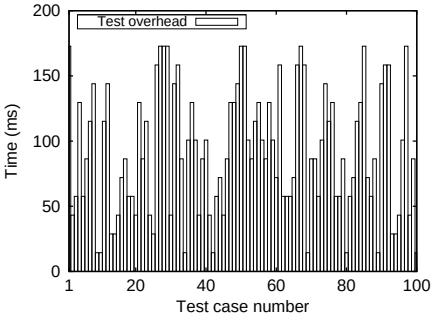

Fig. 6. Performance of Test when $|\gamma| = 10$.

We now consider the efficiency of the scheme. The Encrypt algorithm takes $2n$ group exponentiations in $\mathbb{G}_1$. The Extract algorithm takes $2 \cdot |\gamma|$ group exponentiations in $\mathbb{G}_1$. In BlindExtract algorithm, $\mathcal{DO}$ spends $20 \cdot |\gamma|$ group exponentiations in $\mathbb{G}_1$. $\mathcal{U}$ spends $28 \cdot |\gamma|$ group exponentiations in $\mathbb{G}_1$ plus some verification time dependent on access tree. The Test algorithm's performance depends on the access tree $\mathcal{T}_\gamma$. In conjunction-only case, it involves 1 test of $|\gamma|$ pairing and $|\gamma|$ exponentiation in $\mathbb{G}_2$. In disjunction-only case, it involves $|\gamma|$ tests of 1 pairing operation. Compared to $|\gamma|$ pairing overhead in [17,19,20], our scheme has similar overhead while supporting more flexible queries. The optimized Decrypt algorithm takes $|\gamma'|$ pairing and $|\gamma'|$ group exponentiations in $\mathbb{G}_2$.

7 Performance Evaluation

We implemented the proposed scheme in C++ using PBC (ver. 0.57) [31] and OpenSSL (ver. 1.0.0) [32] library. This section discusses the performance of each function in our scheme. All benchmarks were performed on a Ubuntu 9.10 desktop platform with Intel Core i7-920 (2.66GHz and 8MB cache) and 6GB RAM.

Since performance of each function only depends on the access tree, we do not consider the performance impact of the contents associated with leaf nodes. We use a random access tree (in all tests) that is generated as follows. First we fix the number of leaves, n_{leaves}. Then a random tree height n_{height} between 1 and 5 is chosen. The node degree is computed as $n_{degree} = \lceil n_{leaves}^{1/n_{leaves}} \rceil$. After $n_{leaves}, n_{height}, n_{degree}$ is determined, the random tree is constructed in a down-top manner. At depth l, one parent node is constructed for every n_{degree} nodes at depth $l+1$. If less than n_{degree} nodes are left at depth $l+1$, one parent node is constructed for these remaining nodes. The procedure continues until only one parent (root) can be constructed. For simplicity, we assume the total number of attributes $w = |\gamma| = n_{leaves}$.

First we test the speed of Encrypt. Fig. 3 (Encryption Speed line) shows the overhead to compute $Y^s, E', \{E_i, E_i'\}_{i \in \Omega}$ versus the number of attributes $|\gamma|$. As we can see, its overhead increases linearly with $|\gamma|$. Fig. 4 shows the performance of symmetric encryption, which is needed to compute $E = \mathsf{Enc}^{sym}_{\mathcal{H}(Y^s)}(m)$.

Extract and BlindExtract performance is also shown in Fig. 3. In this test, the threshold gates in the access tree are chosen randomly. The overhead of Extract (Extraction (Data Owner) line) is solely at $\mathcal{DO}$ side and it increases linearly with $|\gamma|$. The overhead of BlindExtract is at both $\mathcal{U}$ side and $\mathcal{DO}$ side. The overhead at $\mathcal{DO}$ side (Blind Extraction (Data Owner) line) is almost nine times that of normal extraction. The overhead at $\mathcal{U}$ side (Blind Extraction (User) line) doubles that at $\mathcal{DO}$ side.

To test Decrypt, we assume $\gamma' = \gamma$, i.e., all attributes should be involved in the decryption. Since all threshold gates in $\mathcal{T}_{\gamma'}$ should be conjunctive gates, we make them conjunctive in the random access tree $\mathcal{T}_\gamma$ as well. Fig. 3 (Decryption Speed line) shows the speed to recover Y^s. We find that decryption overhead increases linearly with $|\gamma|$ and it is even cheaper than extraction. The reason is because pairing operation and exponentiation in $\mathbb{G}_2$ is faster than exponentiation in $\mathbb{G}_1$ [1]. Fig. 5 shows the speed of computing f_x for all leaf node x, which is necessary for the optimization of decryption. Its speed is almost linear with the product of $|\gamma|$, tree height and tree degree. Note this part of operation can be conducted offline and only needs to be computed once for one type of access tree. The performance of $\mathsf{Dec}^{sym}_{\mathcal{H}(Y^s)}(E)$ is same as $\mathsf{Enc}^{sym}_{\mathcal{H}(Y^s)}(m)$ as shown in Fig. 4.

As to Test performance, it highly depends on the access tree. During the following test, the performance is recorded in the worst case, i.e. all possible subtrees $\mathcal{T}_{\gamma'}$ of $\mathcal{T}_\gamma$ are tried. Fig. 3 shows the conjunction-only Test and disjunction-only Test performance. As we can see, they all increases linearly with $|\gamma|$. The reason why they are almost the same is because conjunction-only Test has 1 test involving $|\gamma|$ pairing and $|\gamma|$ exponentiation in $\mathbb{G}_2$ while disjunction-only Test has $|\gamma|$ tests involving 1 pairing. To further test Test operation, we use random access tree. We restrict $|\gamma|$ to be 10, which is usually enough for normal query, and set each threshold gate in the tree randomly. Fig. 6 shows the results of 100 test cases. As we can see the maximum Test time is 170ms and the average Test time is 85ms. In cloud computing scenario, multiple Test operations can run simultaneously and therefore spending average 85ms on each record is acceptable.

[1] In our benchmark of Type A pairing family in [31], one exponentiation in $\mathbb{G}_1$ takes 1.9 ms, one exponentiation in $\mathbb{G}_2$ takes 0.18 ms while one group pairing takes 1.4 ms.

8 Limitation

The proposed scheme has some limitation and it should be considered in future work. First it only supports equality testing. Practical privacy-preserving comparison is not available yet. Second, it only hides concrete value in the conditional expression and the structure $\mathcal{T}_\gamma$ is revealed to the adversary. Third, join operations between two tables are not supported. Fourth, if the set of possible attribute values in γ is small, the adversary can always try to encrypt something under all possible values and run Test over the encryptions to see if there is a match. This would reveal v_γ within $tk_{(\mathcal{T}_\gamma, \mathbf{v}_\gamma)}$. However, the complexity of such brute force attacks against this intrinsic weakness of public key-based searchable encryption, grows exponentially with $|\gamma|$. Fifth, $\mathcal{DO}$ is required to be online to help $\mathcal{U}$ extract search tokens and decryption keys. However, we expect that this functionality can be finished by some secure hardware that can be safely installed at $\mathcal{U}$ side without compromising $msk_{\mathcal{DO}}$.

9 Conclusion

This paper provides an overview of privacy challenges facing cloud storage and develops a novel encryption scheme for coping with these challenges. The scheme hides the plaintext of database and user's query content from the cloud server. It allows data owner to do content-level fine-grained access control by issuing users appropriate search tokens and decryption keys. The scheme also supports blind retrieval of search tokens and decryption keys in the sense neither data owner nor cloud server learns the query content. Additional feature of user input authorization by CA can also be supported. Our evaluation shows that its performance falls within the acceptable range.

References

1. V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM CCS'06*, 2006.
2. M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *CRYPTO'09*, 2009.
3. Y. Lu and G. Tsudik, "Enhancing data privacy in the cloud," http://eprint.iacr.org/2011/158.
4. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.
5. M. Rabin, "How to exchange secrets by oblivious transfer," Harvard Aiken Computation Lab, Tech. Rep. TR-81, 1981.
6. J. Reardon, J. Pound, and I. Goldberg, "Relational-complete private information retrieval," University of Waterloo, Tech. Rep. CACR 2007-34, 2007.
7. F. Olumofin and I. Goldberg, "Privacy-preserving queries over relational databases," in *PETS'10*, 2010.
8. D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *S&P'00*, 2000.
9. Y. C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *ACNS'05*, 2005.
10. Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving queries on encrypted data," in *ESORICS*, 2006.

11. P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *ACNS'04*, 2004.
12. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Eurocrypt'04*, 2004.
13. A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Eurocrypt'05*, 2005.
14. J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *S&P'07*, 2007.
15. R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *CCS'07*, 2007.
16. B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in *NDSS'04*, 2004.
17. E. Shi and B. Waters, "Delegating capabilities in predicate encryption systems," in *ICALP'08*, 2008.
18. E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *S&P'07*, 2007.
19. D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *TCC'07*, 2007.
20. J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Eurocrypt'08*, 2008.
21. M. Green and S. Hohenberger, "Blind identity-based encryption and simulatable oblivious transfer," in *ASIACRYPT'07*, 2007.
22. J. Camenisch, G. Neven, and abhi shelat, "Simulatable adaptive oblivious transfer," in *EUROCRYPT'07*, 2007.
23. N. Koblitz, "Elliptic curve cryptosystems," in *Mathematics of Computation*, 1987.
24. P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT'99*, 1999.
25. J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *CRYPTO'97*, 1997.
26. D. Chaum, J.-H. Evertse, and J. Van De Graaf, "An improved protocol for demonstrating possession of discrete logarithms and some generalizations," in *EUROCRYPT'87*, 1988.
27. D. Chaum, "Zero-knowledge undeniable signatures," in *EUROCRYPT'90*, 1990.
28. F. Boudot, "Efficient proofs that a committed number lies in an interval," in *EUROCRYPT'00*, 2000.
29. D. Boneh and X. Boyen, "Efficient selective-id secure identity based encryption without random oracles," in *EUROCRYPT'04*, 2004.
30. ——, "Short signatures without random oracles," in *EUROCRYPT'04*, 2004.
31. B. Lynn, "PBC: The Pairing-Based Cryptography Library," http://crypto.stanford.edu/pbc/.
32. "OpenSSL," http://www.openssl.org/.