An Inference Resistant Indexing Technique to Enforce Access Control in Data Outsourcing

Hadi Halvachi\textsuperscript{a,*}, Mohammad Ali Hadavi\textsuperscript{a}, Rasool Jalili\textsuperscript{a}
\textsuperscript{a}Data and Network Security Lab (DNSL), Sharif University of Technology, Tehran, Iran.

\textbf{A R T I C L E I N F O.}

\textbf{Article history:}
Received: 22 October 2016
Revised: 13 February 2017
Accepted: 15 April 2017
Published Online: 26 November 2017

\textbf{Keywords:}
Data Outsourcing; Secure Encrypted Data Indexing; Server-side Access Control Enforcement.

\textbf{A B S T R A C T}

Although data outsourcing provides many benefits, it suffers from privacy and security concerns such as enforcement of access control policies and confidentiality of stored sensitive data. Current encryption-based security solutions are inflexible in enforcing fine-grained access control policies and it is necessary for data owner to laboriously specify access control permissions per data item. Additionally, the current indexing methods disregard any control on accessing the relevant data and may cause information leakage. This paper proposes a novel indexing technique to enforce access control policies at server side and based on the value of encrypted data. By exploiting this indexing method and selective encryption, we introduce an access control aware indexing technique, which we refer to as Inference Resistant Indexing Technique (IRIT). The proposed technique, not only prevents leakage of information but also overcomes the overheads associated with a separate access control enforcement mechanism. Meanwhile, the overhead associated with updating access control policies is noticeably reduced. The paper provides a simulation of the proposed technique and a comparison with the alternative approaches to assess the performance of the proposed technique.

© 2016 JComSec. All rights reserved.

1 Introduction

The explosive growth and rapid evolution of data storage, data processing, and communication technologies have changed the traditional methods of data management and storage. Due to the growing costs of in-house storage and management of large collections of sensitive data, mainly because of its demands for both storage/processing capacity and skilled administrative staff, data outsourcing has been an emerging and growing approach. Data outsourcing is proved to be advantageous in terms of reducing management costs, providing a higher level of availability, and more effective disaster protection [1]. However, it introduces new security concerns, especially about the enforcing of data owner access control policies as well as confidentiality and integrity of sensitive data. The root of such concerns is storing data out of the territory of its owner, under the control and supervision of an honest-but-curious administrator. Such an administrator is relied upon for ensuring availability and integrity of data, returning correct and complete responses to queries, and enforcing the basic security controls. However, he might not be relied on in terms of the data confidentiality [1].

Many solutions have been proposed to protect the
confidentiality of data, most of which are based on encryption methods. Since efficient query execution on encrypted data is hard to achieve (and sometimes impractical), some indexing methods for searching over encrypted data such as those based on domain bucketization [2] or hash functions [3] have been proposed to generate indices for each tuple. Indices are used to execute queries on encrypted data by the server without the need of data decryption. The set of returned tuples generated by the server is usually a superset of the actual query response. Therefore, a refinement process should be executed by the client (or data owner) to remove extra tuples. The refinement process causes the overhead of decryption and query re-execution at the client (or owner) side.

Enforcing data owner access control policies is another important security challenge in data outsourcing. The challenge is that it is not efficient for data owner to mediate each user’s request to enforce his access control policies. On the other hand, the server hosting encrypted data is not fully trusted to correctly enforce data owner’s access policies. Moreover, access control enforcement cannot be delegated to the external server for privacy reasons such as the sensitivity of the policies. The external server can potentially collude with some malicious users to gain unauthorized access to sensitive data. Thus, this paper proposes an access control technique based on securely generated indices, which are driven by the data itself. The indexing method itself considers access control policies defined by the owner while it prevents inference relying on data indices. Our Inference Resistant Indexing Technique (IRIT) permits the data owner to define a single intuitive access control policy for each data attribute based on its domain partition. The proposed technique generates data indices depending on both data values and authorized users’ keys which are assigned to individual users based on access control policies. Using our proposed indexing technique, the server enforces the policies when an access request is being processed.

Other solutions to permit enforcement of access control policies without data owner intervention are also based on a combination of access control mechanism and encryption. In such techniques, the tuples are encrypted with different keys and some indices are generated to compute the result of queries by the server, without requiring tuples decryption. In such encryption based techniques, based on indexing technique, a query refinement on result tuples is needed by the client (or owner). Moreover, due to requiring re-encryption of the whole tuples by the owner, updating access control policies is expensive and may reveal the access control policies to the server. Additionally, the encryption based techniques may cause leakage of information when they are used with indices.

The remainder of this paper is structured as follows. Section 2 presents the related work. Section 3 investigates the problem of co-existence of selective access to encrypted data and indices. Section 4 defines an access control policy in a sample scenario and then introduces our access control aware indexing technique (IRIT), which supports the enforcement of IRIT policies. Section 5 provides an evaluation of our technique and shows the experimental results. Conclusions and future work are presented in Section 6.

2 Related Work

Encryption is the first choice for preserving outsourced data confidentiality. Since the server is not allowed to decrypt outsourced data, different techniques have been proposed for direct query execution over the encrypted outsourced data [2, 17]. Some of the techniques propose special encryption functions such as (somewhat/fully) homomorphic encryption [8, 11], order-preserving encryption [6, 12, 15], and searchable encryption [10, 17]. Departing from the complexity and efficiency issues regarding the design of such property-preserving encryption schemes, some others suggest creating some metadata to be used for search over encrypted values [2, 15, 19]. In metadata based solutions, an auxiliary data, also known as index, is constructed in a way to facilitate search over encrypted values. Metadata is stored at server side beside the encrypted values. Bucket-based [2], hash based [3, 20], and B+-tree indexing [3] are among some popular index-based methods. In bucket-based indexing, the domain of a searchable attribute is divided into a set of non-overlapping partitions, either of the same or different sizes. Each partition is then assigned a label as the metadata of the values fall in the partition. In hash-based indexing, the index values are computed using a hash function on the corresponding plaintext values. These techniques tolerate frequency attacks since they usually generate collisions by mapping multiple values to the same metadata. In B+-tree indexing, an encrypted version of a B+-tree is built over the searchable attribute. The tree is iteratively evaluated in response to both equality and range queries. B+-tree indexing, while offers sound query results and more security, imposes extra communication overhead for query execution. Index based methods have been used more to support different query types on numeric and character data [15, 19, 23], and to build metadata, which reveals the least information possible yet provides efficient search capability [3, 21, 27]. The above index-based solutions often neglect access control policies and do not consider users’ access rights to encrypted values,
which results in assigning similar metadata to values having different access lists.

Access control enforcement over encrypted outsourced data is another line of research taken into account at first by the idea of selective encryption and key derivation [28–29]. In selective encryption, data is encrypted with respect to access control policies. Then each user, having his private key, is able to decrypt the part of data which he is authorized to access. By emerging the paradigm of cloud computing, many aspects of access control enforcement were investigated such as enforcement of access policies while the server is not fully trusted [30–33], enforcing role-based access control policies [34–36], updateability of access policies [30–34], preserving policy confidentiality [35–37], and attribute-based encryption for managing users’ access to plain data [32–34]. The basic idea is to propose a mechanism to delegate the enforcement of access control policies to the server to which data is outsourced. The majority of these solutions enforce access policies on the plain data regardless of the query execution technique. That is, they limit users’ ability to decrypt query results with respect to the policies, but permit users to pose queries having different conditions and observe selected encrypted results. This is a source of information leakage due to ignoring access control policies when searching encrypted values. A few works consider this kind of information leakage and propose a method in which users cannot search on the encrypted values unless they have appropriate permissions [25, 45–46]. De Capitani di Vimercati et al. [45] produce metadata so that different occurrences of the same plaintext values, accessible by different sets of users, are not recognizable from their corresponding metadata. Hadavi et al. [46] propose an access control enforcement mechanism over secret shared based relational data. Their method prevents the possibility of information leakage caused by query processing through an access control aware retrieval of data shares. Shmueli et al. in [25], take access control into account by defining separate encrypted B-tree indexes for different access control list in the system. The encryption key of each B+-tree is accessible by the users in the corresponding access control list.

In the techniques based on selective encryption and key derivation, the update of access control policies is expensive and requires data re-encryption. In addition, the adoption of key derivation for access control policies enforcement may reveal the access control policies to the server. Whenever the policies are considered sensitive, the structure of the key derivation hierarchy should not be revealed to the external server [31]. We consider the searchability of encrypted values and also access control policies by presenting an access control aware indexing technique so that equal values having different access lists are assigned distinct indices, thus minimizing inference due to accessing indices.

3 Challenges and Motivation

Inference Resistant Indexing Technique (IRIT) is a new method for controlling the access of users to data without owner intervention. In IRIT, access control policies are fine-grained and defined based on the content of data. IRIT is especially usable in data outsourcing to protect databases containing potentially sensitive data. A good example would be a health record management system that allows different users to access database tuples depending on data contents. For instance, a nurse may have access to blood tests unless the test is an HIV test. Only certain people can access such sensitive records.

Current approaches, which enforce access control policies over outsourced data [40–42] are inappropriate since access control policies in this scenario, contrarily to other solutions, are based on data contents. Moreover, index based solutions to search on encrypted values [2–4, 23, 24] use indices whose values are constructed independent of the access control policies, so they are the sources of information leakage during query processing [45]. Additionally, in the majority of the existing index based methods, the server returns some extra tuples besides the actual result, which needs a client side refinement process. The refinement process includes query result post-processing after decryption at client side to filter-out extra results returned from the server. The reason for producing such extra tuples, which in turn imposes extra communication overhead and client side computation overhead, is the ignoring of access control policies while building data indices. That is, some tuples are returned by the servers with respect to their index values while the user is not permitted to decrypt them.

In this paper, we suppose that encryption is performed at the tuple level of granularity. If the plaintext data is a relation over the schema $R(A_1, A_2, \ldots, A_n)$, then the outsourced data is a relation over the schema $R'(t_{id}, etuple, I_1, I_2, \ldots, I_m)$, where $t_{id}$ is a numerical attribute added to the encrypted relation as a primary key, $etuple$ is an attribute containing the cipher-text resulted from the encryption of a tuple, and $I_i (i = 1, \ldots, m)$ is an attribute corresponding to the index over an attribute $A_i \in R$. Also, we use fine-grained access control policies and suppose access control policies are specified at the granularity of attribute values (field granularity). We assume that users have just read access to database tuples, while write operations are to be performed by the owner himself. For the sake of simplicity, we also assume that there
exists only one policy defined on an attribute of a relation. Data in this paper is organized as a relational table and indices are defined as attributes of the table. As a running example, we consider a relation Employee(eid, name, salary) illustrated as Table 1. The data owner specifies IRIT policy on salary values illustrated in Table 2.

Although selective encryption guarantees the enforcement of access control policies, the existence of indices beside encrypted data leads to information leakage and allows unauthorized users to infer the data contents. As index values are closely related to the plain data and also the index function is known to a user, he can infer the values supposed to be hidden to him. For example, suppose the application of selective encryption to our running example which results in the encrypted data illustrated in Table 3. Each data item is encrypted using a key and each key is accessible only to the users who are authorized to access and decrypt. Then he changes the query condition and poses a new query (2) and receives the response $R_2$. Comparing both results reveals that the SALARY value of the newly added tuple (strike-through tuple) is 5000. Accordingly, it can be concluded that each user can infer unauthorized values by executing authorized queries. So, an approach is required to enforce access control policies and produce indices while preventing information leakage.

**Table 1. Plaintext Data**

<table>
<thead>
<tr>
<th>eid</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Ali</td>
<td>5000</td>
</tr>
<tr>
<td>234</td>
<td>Reza</td>
<td>3000</td>
</tr>
<tr>
<td>345</td>
<td>Hasan</td>
<td>1500</td>
</tr>
<tr>
<td>456</td>
<td>Mojtaba</td>
<td>2500</td>
</tr>
<tr>
<td>567</td>
<td>Farid</td>
<td>800</td>
</tr>
<tr>
<td>678</td>
<td>Mahdi</td>
<td>7000</td>
</tr>
</tbody>
</table>

**Table 2. IRIT Policy of Data Owner**

<table>
<thead>
<tr>
<th>The Range of Data</th>
<th>Authorized Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>salary ≤ 2000</td>
<td>$u_1, u_2, u_3, u_4$</td>
</tr>
<tr>
<td>2000 &lt; salary ≤ 4000</td>
<td>$u_1, u_2$</td>
</tr>
<tr>
<td>salary &gt; 4000</td>
<td>$u_1$</td>
</tr>
</tbody>
</table>

**Table 3. Encrypted Data Using Selective Encryption**

<table>
<thead>
<tr>
<th>$t_id$</th>
<th>$c_{tuple}$</th>
<th>$S−index$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$E_{k1}(123, Ali, 5000)$</td>
<td>Index(5000)</td>
</tr>
<tr>
<td>2</td>
<td>$E_{k2}(234, Reza, 3000)$</td>
<td>Index(3000)</td>
</tr>
<tr>
<td>3</td>
<td>$E_{k3}(345, Hasan, 1500)$</td>
<td>Index(1500)</td>
</tr>
<tr>
<td>4</td>
<td>$E_{k4}(456, Mojtaba, 2500)$</td>
<td>Index(2500)</td>
</tr>
<tr>
<td>5</td>
<td>$E_{k5}(567, Farid, 800)$</td>
<td>Index(800)</td>
</tr>
<tr>
<td>6</td>
<td>$E_{k6}(678, Mahdi, 7000)$</td>
<td>Index(7000)</td>
</tr>
</tbody>
</table>

**Table 4. View of the User $u_2$**

<table>
<thead>
<tr>
<th>$t_id$</th>
<th>$c_{tuple}$</th>
<th>$S−index$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$E_{k1}(123, Ali, 5000)$</td>
<td>Index(5000)</td>
</tr>
<tr>
<td>2</td>
<td>$E_{k2}(234, Reza, 3000)$</td>
<td>Index(3000)</td>
</tr>
<tr>
<td>3</td>
<td>$E_{k3}(345, Hasan, 1500)$</td>
<td>Index(1500)</td>
</tr>
<tr>
<td>4</td>
<td>$E_{k4}(456, Mojtaba, 2500)$</td>
<td>Index(2500)</td>
</tr>
<tr>
<td>5</td>
<td>$E_{k5}(567, Farid, 800)$</td>
<td>Index(800)</td>
</tr>
<tr>
<td>6</td>
<td>$E_{k6}(678, Mahdi, 7000)$</td>
<td>Index(7000)</td>
</tr>
</tbody>
</table>

**4 Inference Resistant Indexing Technique**

In this section, we present our access control aware indexing technique to enforce access control policies in addition to enabling efficient query execution. A new presentation of access control policies named IRIT
policies is used to generate access control aware indices. An IRIT policy on relational databases is formally defined by Definition 1. Without loss of generality, each IRIT policy is defined on only one attribute of a relation.

**Definition 1.** IRIT Policy: for $D$ as a partitioning set of an attribute domain, $\forall d_i, d_j \in D, i \neq j : d_i \cap d_j = \emptyset$, and $U$ as the set of users, an IRIT policy, denoted as $A$, is a function $A : D \rightarrow 2^U$ that maps a range $d \in D$ onto a subset $r \subseteq U$ of users.

In Definition 1, the IRIT policy is specified over a domain of a relation attribute, partitioned into non-overlapping ranges. $D$ should be an IRIT Usable partitioning set of an attribute domain. A partitioning set $D$ is an IRIT Usable partitioning set if $\forall d_i \in D$ the same set of authorized users can see all attribute values in the corresponding range to $d_i$. There are many IRIT Usable partitioning sets for each data domain. If a data domain is a discrete domain, a primary partitioning set of the domain could be a set of all members of the domain. For example, considering the domain of “age” as a discrete domain, a partitioning set could be $D = \{\{1\}, \{2\}, \ldots, \{120\}\}$. The policy assigns to each partition of the domain a set of users who are authorized to read values of the attribute. For instance, an IRIT policy, illustrated in Table 2 is specified as follows:

$$A = \{<0 \ldots 2000>, \{u_1, u_2, u_3, u_4\}, <(2000, 4000), u_1, u_2>, <(4000 \ldots \infty), \{u_1\}>\}$$

(3)

Our proposed technique for indexing enforces IRIT policies and prevents information leakage since users do not have unauthorized access to data even in encrypted form.

In previous indexing techniques over encrypted data, the index is a function of the data value. Moreover, users are supposed to regenerate index values for querying encrypted data, regardless of being authorized to access the original values. Minimum leakage, in this case, is the inference on the existence of a data value that violates data confidentiality by itself. To prevent drawing such inferences, indices should be generated in a manner in which a user can only regenerate indices corresponding to his capability list. Therefore each user can submit queries only on his authorized view of data, according to access control policies. For this purpose, we propose utilizing encryption for indexing such that index values depend on users’ keys. Suppose that we have an indexing technique with a function $E(\cdot)$ that supports equality queries e.g. bucketing, or hash based indexing techniques. We use Equation (4) for generating the index of a value $v$, depending on both data values and authorized users’ keys:

$$index = E(\text{index}(v), \text{user.key})$$

(4)

$E(\cdot)$ in Equation (4) is a deterministic symmetric encryption function with key user.key. For each data value, the owner generates an index per authorized user or user group. Thus, for a value $v$, the owner inserts $|ACL(v)|$ tuples into the encrypted table with a different index value constructed for each tuple. $ACL(v)$ is the Access Control List of attribute value $v$ and $|ACL(v)|$ is the number of users in $ACL(v)$.

$|ACL(v)|$ shows the number of users or user groups (based on key distribution strategy) who have authorized access right to attribute value $v$. There are different key distribution methods and two overall strategies. In the first strategy, which is used in most methods, keys are assigned to groups of users through different methods such as a hierarchical key tree. In the second strategy, keys are assigned to individual users based on access control policies. We use the second strategy in this paper. For example, Table 3 is the encrypted version of Table 2, according to the access control policy in Table 2. For the first tuple that only $u_1$ is authorized to access, one encrypted tuple is inserted into the encrypted table with an index generated by $u_1$’s key. However, according to the access control policy in Table 2 for the second and the third tuple in Table 2, two and three tuples are inserted, respectively, into the encrypted table.

This technique permits the server to infer access control policies, as the different occurrences of a
The policy assigns to each partition of the domain a set of users who are authorized to read values of the domain. Each user has a special view on the indices, according to their access control policies. For example, in Table 8, the server infers that only user 1 can access the first tuple. To prevent access control policies being disclosed, an idea is to use a salt value for tuple encryption. Considering this change, the schema of an encrypted relation becomes $R'(\text{tid}, \text{salt}, \text{etuple}, I_1, \ldots, I_m)$. Table 9 illustrates new encrypted table in which the same tuples mapped onto different encrypted ones so that the server cannot infer about access control policies.

In most cases, users submit range queries to the server to retrieve a range of stored data. To this end, we extend our indexing technique to a new index construction scheme. An index for a value $v$ in the new scheme is constructed by Equation 5, where $R$ is an indexing function that supports range queries.

$$index = OPE(\text{user key}||R-Index(v)) \quad (5)$$

tuple show the number of their authorized users. For example, in Table 8, the server infers that only one user can access the first tuple. To prevent access control policies being disclosed, an idea is to use a salt value for tuple encryption. Considering this change, the schema of an encrypted relation becomes $R'(\text{tid}, \text{salt}, \text{etuple}, I_1, \ldots, I_m)$. Table 9 illustrates new encrypted table in which the same tuples mapped onto different encrypted ones so that the server cannot infer about access control policies.

In most cases, users submit range queries to the server to retrieve a range of stored data. To this end, we extend our indexing technique to a new index construction scheme. An index for a value $v$ in the new scheme is constructed by Equation 5, where $R-Index$ is an indexing function that supports range queries.

$$index = OPE(\text{user key}||R-Index(v)) \quad (5)$$

The function $OPE$ in Equation 5 is an Order Preserving Encryption function [12] and $||$ is the concatenation operation. Note that the user $\text{key}$ must appear at the left side of the concatenation as more significant bits. This order of inputs preserves the order of values in their corresponding indices. The views of different users are separated by Equation 6. Each user has a special view on the indices, according to the given IRIT policies. Figure 1 illustrates the separation of users’ view.

This is still possible for a user to submit a query without condition (or with a tautology condition) and retrieve tuples, out of his permitted view. To prevent such unauthorized tuple retrieval, the external server must verify the authenticity of each query. A query submitted by a user $u$ is an authentic query if it has WHERE clause and the range of indices appeared in the WHERE clause is a subset of $u$’s permitted view (Definition 2).

**Definition 2.** Authentic Query: Let $Q$ be a query submitted by a user $u$, $R$ be the authorized range of indices for $u$, and $W$ be the range of indices appeared in the WHERE clause of $Q$. The query $Q$ is said to be an authentic query if $W \subseteq R$.

The authenticity verification of queries is simple. The server stores the minimum and maximum values of the indices for each user $u$ and checks that the range of the indices falls into the range of $u$’s permitted view. For instance, the minimum value for user $u$ with key $K$ is $OPE(K||m)$ where $m$ is the minimum index value, generated by the $R-Index$ function.

The role of $OPE$ function is hiding user $\text{key}$ and more importantly, hiding the distribution of indices generated by the $R-Index$ function. Index construction methods are usually vulnerable to information exposure due to inferring over index distribution compared to the original data distribution [3]. However, our access control aware indexing technique decreases the possibility of such inference exposure. This is because a value $v$ has different corresponding indices with respect to its ACL that complicates inferring...
about original data distribution using index values.

5 Evaluation and Results

We now evaluate the security and efficiency of the proposed technique. Efficiency evaluation is based on some experimental tests on our indexing method.

5.1 Evaluation

Our presented indexing technique permits the server to enforce IRIT policies without the owner intervention. Moreover, this indexing technique prevents information being revealed to users by separating the view of different users having different access rights. The salt is also used to prevent the server from inferring access control policies. This technique guarantees the correct enforcement of IRIT policies. To add a new tuple to the encrypted table, the owner generates appropriate index values with respect to the tuple’s ACL. Thus, only authorized users can regenerate indices and retrieve associated tuples.

In previous indexing techniques, the query result consists of all tuples that satisfy the query condition, regardless of access control policies. Our indexing technique, however, decreases the communication cost by eliminating the tuples which are out of the user’s permitted view from the returned result set. This is the direct result of considering access control policies in index construction method. Moreover, eliminating tuples, which are not in the user’s permitted view, decreases the client side computational overhead.

Selective encryption and key derivation based methods [5–8], which are the basis of the majority of research on access control enforcement mechanisms for the data outsourcing scenario, suffer from policy update that usually imposes a considerable overhead to the system. Our index construction method, while enforcing IRIT policies, efficiently supports updating access control policies. It does not require re-encryption of resources after a revoke operation, which is the main challenge of the policy update. Updating access control policies is mapped to INSERT and DELETE queries. To grant an access to a user, the owner easily inserts new tuples into the encrypted table with appropriate indices generated by the user’s key. To revoke an access, the owner deletes the related tuples.

The shortcoming of our technique is its redundancy at the server side. Each tuple is mapped to some different encrypted tuples to make it visible to authorized users. The amount of redundancy depends on access control policies and the number of users authorized to access data. For calculating the redundancy, suppose each tuple $t_i$ has $\delta_i$ authorized user. So, $\delta_i - 1$ extra versions of encrypted tuple cause additional storage and processing overhead. Therefore, redundancy factor is equal to $\sum_{i=1}^{\nu} \delta_i - 1$. If each tuple has $\nu$ authorized user in average, the redundancy factor is $[\nu \times \text{(number of tuples)}]$.

Vimercati et al. present an indexing technique [15] for preventing the possible disclosure of information to unauthorized users. They assume access control policies with the granularity of tuple and investigate the problem of inferences over indices. The indexing technique presented in [15] has as much storage overhead on the server side as our presented technique, while, unlike our technique, supports only equality queries.

5.2 Experimental Results

To evaluate the presented indexing technique, we implemented a prototype and performed a series of experiments. We considered the schema of our running example and IRIT policy in Table 2. We generated more than 14,000 tuples randomly and compared our technique with the selective encryption in [29]. The parameters for evaluating the techniques are the number of the tuples in query response and the communication overhead. We submitted some queries from users $u_2$ and $u_3$ and counted the tuples of the response and measured the size of network traffic. We generated some queries in the form of Equation (6) where $\xi$ is a variable.

SELECT * FROM emp
WHERE $S-index < Index(\xi)$ (6)

The results of our experiments are illustrated in Figure 2 and Figure 3. The horizontal axis is the value of WHERE clause variable ($\xi$ in Equation (6)). Figure 2 shows the number of returned tuples from the server for different users. Figure 3 shows the communication overhead of queries submitted by $u_2$. In our technique, the number of tuples and consequently the communication overhead is dramatically less than selective encryption since our technique has restricted access to encrypted data.
Conclusion and Future Work

We presented a technique for indexing encrypted data to enforce granular access control policies. We suggested using IRIT to specify granular access control policies. The IRIT policies are enforced by the server without data owner intermediation. The indices are safe from inferences, meaning that they do not leak information about underlying data to users who are not authorized to access the data. The enforcement of access control policies while querying encrypted data eliminates extra tuples from the query result set, that itself decreases communication cost of query processing. Moreover, it reduces the imposed processing overhead at client side due to the client post-process on returned result. While specifying content aware policies in our technique is simple, the cost of policy update is reduced compared to other access control enforcement mechanisms. The major disadvantage of our indexing technique is storage and accordingly process overhead on the server side. The storage overhead is caused by generating multiple tuples per authorized users of a tuple.

In future, we plan to reduce the imposed redundancy especially when access control policies are specified on more than one attribute. Using approaches such as key derivation can help to eliminate storage, though it produces new challenges regarding policy updates. Another extension to the current work is to consider other access modes such as write and update while constructing data indices.

Acknowledgements

The authors would like to thank Mr. V. Khodakakhshi, Mr. M. Mehrabi, Mr. R. Mahfooz, and Ms. L. Karimi, for their valuable comments and suggestions. The research work reported here has been done in Data & Network Security Laboratory (DNSL) of Sharif University of Technology.

References


[9] Craig Gentry. A fully homomorphic encryption


Hadi Halvachi received his bachelor degree in Computer Engineering from Isfahan University of Technology in 2011 and the master degree in Computer Engineering from Sharif University of Technology in 2013. He is now a Ph.D. candidate in Computer Engineering at Computer Engineering Department of Sharif University of Technology. His research areas are Database Security, Ambient Intelligence (AmI), Internet of Things, IoT Security, and Automation.

Mohammad Ali Hadavi received his Ph.D. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 2015. He received his M.S. and B.S. degrees in Software Engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and from Ferdowsi University of Mashhad, Iran, in 2002, respectively. Focusing on information security, he has published more than 30 papers in national and international journals and conference proceedings. His research interests include software security, database security, and security aspects of data outsourcing.

Rasool Jalili received his bachelor degree in Computer Science from Ferdowsi University of Mashhad in 1985 and the master degree in Computer Engineering from Sharif University of Technology in 1989. He received his Ph.D. in Computer Science from The University of Sydney, Australia, in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, in 1995. He has published more than 130 papers in Computer Security and Pervasive Computing in international journals and conference proceedings. He is now an associate professor, doing research in the areas of access control, vulnerability analysis, and database security in his Data and Network Security Laboratory (DNSL).