Supporting Complex Queries and Access Policies for Multi-user Encrypted Databases

Muhammad Rizwan Asghar
CREATE-NET
International Research Center
Trento, Italy
asghar@create-net.org

Giovanni Russello
Department of
Computer Science
The University of Auckland
New Zealand
g.russello@auckland.ac.nz

Bruno Crispo
Department of
Information Engineering
and Computer Science
University of Trento, Italy
crispo@disi.unitn.it

Mihaela Ion
Department of
Information Engineering
and Computer Science
University of Trento, Italy
mihaela.ion@disi.unitn.it

ABSTRACT

Cloud computing is an emerging paradigm offering companies (virtually) unlimited data storage and computation at attractive costs. It is a cost-effective model because it does not require deployment and maintenance of any dedicated IT infrastructure. Despite its benefits, it introduces new challenges for protecting the confidentiality of the data. Sensitive data like medical records, business or governmental data cannot be stored unencrypted on the cloud. Companies need new mechanisms to control access to the outsourced data and allow users to query the encrypted data without revealing sensitive information to the cloud provider. State-of-the-art schemes do not allow complex encrypted queries over encrypted data in a multi-user setting. Instead, those are limited to keyword searches or conjunctions of keywords. This paper extends work on multi-user encrypted search schemes by supporting SQL-like encrypted queries on encrypted databases. Furthermore, we introduce access control on the data stored in the cloud, where any administrative actions (such as updating access rights or adding/deleting users) do not require re-distributing keys or re-encryption of data. Finally, we implemented our scheme and presented its performance, thus showing feasibility of our approach.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls; E.3 [**Data**]: Data Encryption; H.2 [**Database Management**]: Security, integrity, and protection; K.6 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Encrypted Databases, Complex Encrypted Queries, Access Control, Data Outsourcing

1. INTRODUCTION

The *cloud computing paradigm* offers customers the opportunity to have (virtually) unlimited storage and computation capacities with attractive billing scheme. Storage and computation are offered as services to customers that are billed on the actual amount of consumed resources and can be dynamically adapted to the current situation. Such a revolutionary computational model offers customers very reliable and high-performance storage and computation facilities without the burden of expensive hardware, software and maintenance personnel.

With its cost-effective model, cloud storage is being adopted by a large number of users. However, one of the main limitations of this model is that it offers a very limited end-to-end data confidentiality. Business data, private emails, medical records and government data are few examples of data that could be exposed to unsolicited accesses if simply stored in the cloud. Traditional access control mechanisms are not sufficient for protecting the data confidentiality in the cloud as they rely on a trusted infrastructure that correctly enforces authorisation policies. A malicious cloud provider could easily bypass an access control mechanism on the cloud servers to gain access to the stored data. To circumvent such a limitation, cryptography techniques could be employed. A naive approach is to encrypt the data before storing it on the cloud. The user could download all the data in its local (trusted) environment and perform the decryption and search operations. If the dataset is very large, this trivial solution does not scale well. The matter becomes more complicated when multiple users need to

access the data concurrently.

To partially solve the data protection issue, *single-user searchable encryption* schemes have been proposed [1–8]. In such schemes, the cryptographic components are divided between the user and the server. The user performs encryption and decryption operations while the server performs computations on the encrypted data. In this way, the server is able to perform keyword-based search on the encrypted dataset. The server can retrieve the matching data without learning the key, plaintext and queries. In the single-user searchable encryption schemes, only one user has the key for encrypting, querying and decrypting the data. The secret key could also be shared among a group of authorised users. However, such a decision complicates the key management process as the removal of one user from the authorised group requires the distribution of a new key and the re-encryption of all the stored data.

Semi-fledged multi-user schemes (where a single writes while multiple users read) [9–17] allow users to perform search operations without sharing any key. However, writing operations can be performed by a single user while the other users are only authorised to read. Both single-user and semi-fledged multi-user schemes require key sharing among users. Unfortunately, key sharing increases the risk of key exposure. To partially resolve this issue, keys can be updated regularly. The key updating mechanism may require decrypting all the data with the old keys and re-encrypting it using the new keys. However, this approach is not practical, particularly when data sets are large.

More recently, *full-fledged multi-user* schemes (where each user can write and read) have been introduced, where multiple users can perform read and write operations without sharing keys [18–21]. These schemes are very attractive for large organisations because they support multi-user access to encrypted data with a scalable key management. However, current full-fledged multi-user schemes have two important limitations that hamper their up-taking: (i) these schemes support only keyword-based searches without supporting more complex queries, such as numeric inequalities and range queries; (ii) these schemes offer a coarse-grained access control model. There are solutions that aim at providing the fine-grained access control on data stored in the outsourced environment [22, 23]. However, those solutions are not feasible where administrative actions are updated dynamically; this is because any administrative actions including updating access rights, adding users (or resources) and removing users (or resources) require re-distribution of new keys, as well as re-encryption of existing data with those keys.

In this paper, we tackle these data protection issues and introduce a novel full-fledged multi-user scheme for running encrypted queries over encrypted data outsourced to a cloud environment. The main contributions of this paper are listed as follows.

- First of all, we provide a very extensive overview and categorisation of existing research carried out in the area of outsourced databases. We have categorised existing work according to (i) query expressiveness and (ii) the type of supported key management.
- To the best of our knowledge, this is the first work to present a full-fledged multi-user scheme supporting complex SQL-like queries with WHERE-clause that can express conjunctions and disjunctions of equalities and inequalities.
- Another feature of our scheme is the enforcement of access policies. In our scheme, the database administrator can assign to each user read and write access rights for specific tables (or columns), where any administrative changes (such as updating access rights and removing users or tables) neither require re-distribution of keys nor re-encryption of data.
- Finally, we have implemented our scheme and evaluated its performance.

The rest of this paper is organised as follows. In Section 2, we review state-of-the art. Section 3 explains the system and threat models. The proposed approach is described in Section 4. In Section 5, we elaborate the proposed solution. Section 6 provides some definitions and construction details of functions executed during different phases of the overall system. Section 7 analyses the proposed solution from a security point of view. In Section 8, we evaluates performance of the proposed scheme. Section 9 is dedicated for discussion. Finally, we conclude in Section 10 and provide directions for future work.

2. STATE OF THE ART

In this section, we survey the existing approaches based on expressiveness of the queries, and the key and user management. We have categorised state-of-the-art. Table 1 summarises our categorisation.

2.1 Single-user Schemes

Song *et al.* [1] are the first to address practical keyword search on encrypted data using symmetric encryption, where document is encrypted word by word. To perform search, the user sends keyword encrypted with the same key to the server, which tests each word in every document. This scheme reveals statistical information, such as the frequency of each word. To overcome this weakness, Goh [3] proposes an efficient secure index construction built using pseudo-random functions and Bloom filters. To prevent statistical analysis attacks, each Bloom filter is randomised using a unique document identifier. Bosch *et al.* [7] extend [3] with wildcard searches. However, bloom filters introduce false positives. Chang and Mitzenmacher [5] propose a similar solution for building an index for each document. Their solution provides more security than [3] because number of words in a document are not disclosed. However, it is less efficient and does not support arbitrary updates with new words. Golle *et al.* [4] propose a scheme allowing multi-keyword search with one encrypted query. However, like [5] this scheme is not practical for large databases.

Curtmola *et al.* [10] introduce the first Symmetric Searchable Encryption (SSE) scheme. Kamara *et al.* [28] extend this scheme and introduce a Dynamic Symmetric Search Encryption (DSSE) scheme that supports addition and deletion of documents from the index. Both schemes are limited to keyword searches, without supporting range queries. Bucketisation [24] has been proposed for reducing range queries to equality searches. The main issue is that this scheme has some false positives on query execution. Hacigumus *et al.* [2] propose a solution that enables SQL queries by using

Table 1: Comparison of schemes providing search on encrypted data schemes

Category	Keyword	Conjunction of keywords	Complex queries
Single-user	Song *et al.* (2000) [1] Goh (2003) [3] Chang & Mitzenmacher (2005) [5] Hacigumus *et al.* (2005) [2]	Golle *et al.* (2004) [4] Bosch *et al.* (2011) [7]	Hore *et al.* (2004) [24] Wang & Lakshmanan (2006) [6] Popa *et al.* (2011) [8] Hore *et al.* (2012) [25]
Semi-fledged multi-user	Boneh *et al.* (2004) [9] Curtmola *et al.* (2006) [10] Zhu *et al.* (2011) [15]	Baek *et al.* (2008) [12] Rhee *et al.* (2010) [14] Cao *et al.* (2011) [26]	Boneh & Waters (2007) [11] Katz *et al.* (2008) [13] Yang *et al.* (2011) [16] Li *et al.* (2011) [17] Lu & Tsudik (2011) [27]
Full-fledged multi-user	Bao *et al.* (2008) [20] Dong *et al.* (2008) [19] Shao *et al.* (2010) [21]	Hwang *et al.* (2007) [18]	***No solution yet***

Bucketisation and support range queries after several interactions between the user and the server. Hore *et al.* [25] extend Bucketisation to multi-dimensional data that enables range queries. However, the Bucketisation technique is not scalable.

Another popular method for supporting range queries is Order Preserving Encryption (OPE) [29–31]. The OPE does not introduce false positives and is efficient. However, it reveals the order relation between ciphertext values. Popa *et al.* [8] propose CryptDB for protecting databases. The system relies on a trusted proxy server that intercepts user queries to the protected database. The proxy holds a secret master key and encrypts and decrypts data and queries. For matching keywords, CryptDB uses an implementation of [1] and range queries are supported using the OPE [30]. Simple computations on numeric data are performed using homomorphic encryption based on the Paillier cryptosystem [32]. However, maintaining and securing the proxy server may not be feasible for many companies that choose cloud computing as a way of simplifying operations and reducing costs. Second, the methods used for providing keyword search and range queries have been shown not to provide sufficient security. Last but not least, CryptDB is a single-user scheme while we propose the full-fledged multi-user scheme.

2.2 Semi-fledged Multi-user Schemes

All the above schemes are based on symmetric encryption. The first Public-key Encryption scheme with Keyword Search (PEKS) was proposed by Boneh *et al.* [9]. However, their trapdoor encryption scheme is vulnerable to inference attacks. Subsequently, [12, 14, 15] improved the security of the scheme and [12] also introduced conjunctions of keywords. Public-key encryption is computationally expensive, which makes these schemes too inefficient for large databases. Yang *et al.* [16] propose a solution in which the database owner encrypts the data and assigns to each user a unique key for searching and reading the data. The main idea is that the data owner splits the master secret uniquely between each user and the server. A major drawback of this method is that the search operation is inefficient because it requires expensive pairing operations.

Boneh and Waters [11] propose a public key-based predicate encryption scheme that supports conjunctions, range queries and subset queries on encrypted data. Katz *et al.* [13] extend [11] and propose a predicate encryption for inner products scheme that supports conjunctions and disjunctions. Shen *et al.* [33] propose a symmetric key based predicated encryption scheme that achieves predicate privacy. Li *et al.* [17] propose a solution based on Hidden Vector Encryption (HVE) that uses multiple trusted authorities to distribute search capabilities to users. All these solutions are inefficient because these require expensive evaluations involving pairing operations.

Cao *et al.* [26] propose multi-keyword ranked search over encrypted data, where the database owner creates an initial index. Building the index is very expensive and it cannot be changed dynamically. Lu and Tsudik [27] propose a solution based on Attribute-Based Encryption (ABE) [34] and blind Boneh-Boyen weak signature scheme [35]. However, in their solution, only the data owner is able to encrypt the data and it needs to be online to help authorised users extract search tokens and decryption keys.

2.3 Full-fledged Multi-user Schemes

Hwang *et al.* [18] extend Public key Encryption with Conjunctive Keyword search (PECK) to multi-user settings. To encrypt a message that can be read by a set of users, the sender needs the public keys of all the users. Besides inefficient pairing operations, ciphertext grows linearly with the number of users. Moreover, adding new users to the system requires re-encrypting all the data. Bao *et al.* [20] propose a multi-user solution for keyword searches on encrypted databases, where each user has its own key for writing and reading. The use of bilinear maps and the interactive encryption algorithm make this scheme inefficient. Dong *et al.* [19] propose Searchable Data Encryption (SDE), a multi-user scheme that supports keyword search. The SDE is based on proxy encryption and does not require interactive protocols or pairing. As a result, the SDE is efficient scheme for performing search on encrypted data. Shao *et al.* [21] introduce Proxy Re-Encryption with keyword Search (PRES), a combination of proxy re-encryption and PEKS. Like most of the schemes, this scheme is also based on pairing for testing keywords that makes it inefficient.

As illustrated in Table 1, most solutions provide keyword search or conjunctions of keywords. None of the surveyed schemes is able to support both full-fledged multi-user support and complex queries.

3. SYSTEM MODEL

In our scenario, we consider an outsourced database hosted by a cloud service provider. In the system model, we assume the following entities:

- **Database User (DBU)**: It represents an authorised entity that is part of the system. As such, a DBU can encrypt data and insert it in the database. Each DBU is also able to retrieve data from the database performing complex queries. Once the matching data has been

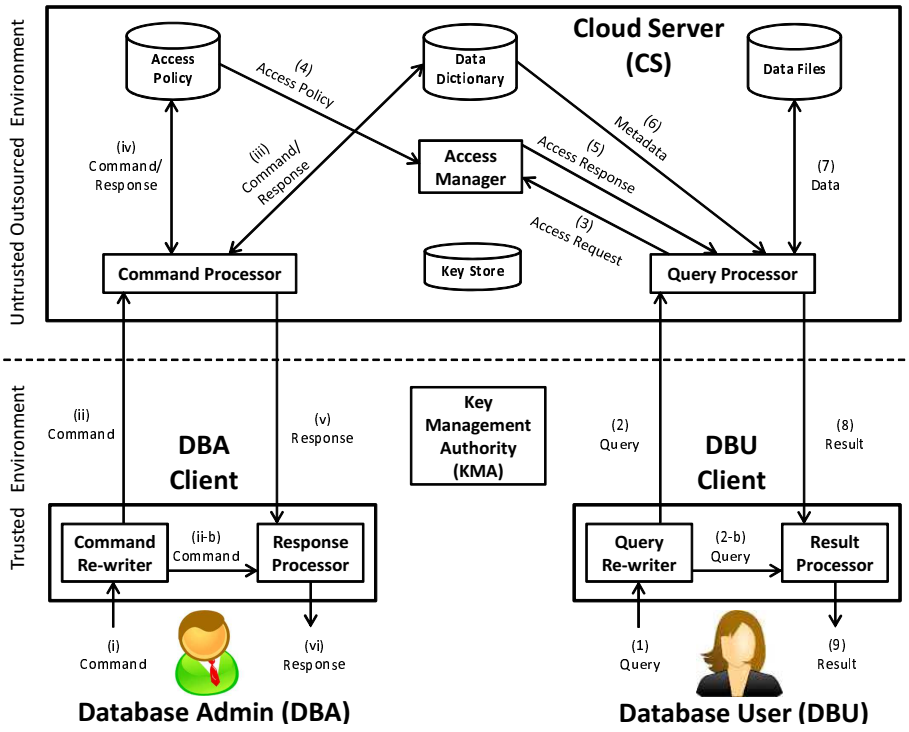

Figure 1: The proposed architecture for encrypted databases, consisting of the CS in the outsourced environment and both the DBA and the DBU (clients) and the KMA in the trusted environment

retrieved, the DBU can decrypt it. Moreover, a DBU can modify or delete data stored in the database.

- **Database Administrator (DBA)**: It is responsible for managing tables and users in the database. The DBA can create new tables or drop existing ones. Furthermore, it administrates access policies for DBUs. That is, it decides who is authorised to access tables in the database.
- **Cloud Server (CS)**: A CS is part of the infrastructure provided by the cloud service provider. It stores the encrypted data and access policies. After making authorisation checks, it performs encrypted searches according to the DBUs' requests.
- **Key Management Authority (KMA)**: This authority is responsible for generating and revoking keys. For each DBU that is authorised by the DBA to join the system, the KMA generates a set of keys. The set of keys is securely distributed. When a DBU (or a DBA) has no longer permissions to be part of the system, the KMA revokes the corresponding keys with the support of the CS.

Threat Model: We assume that the KMA is fully trusted. Typically, the KMA is directly under the control of the organisation that is outsourcing data to the cloud. Since the KMA has to manage a limited amount of data, it is easy for the organisation to properly secure the KMA. Once DBUs are authorised to join the system by the DBA, they are trusted to securely store their keys and the data.

The CS is modelled as honest-but-curious (as considered in state-of-the-art on data outsourcing [22,36]). That is, the CS is trusted to "honestly" perform operations requested by the DBUs according to the protocol specification. However, the CS is not trusted to guarantee data confidentiality and it could analyse the message flow in order to learn information about the stored data. We assume that the CS will not mount active attacks, such as modifying the message flow or denying access to the database.

It is assumed that there are mechanisms in place for data integrity and availability. We consider that DBUs with different access rights may collude in order to retrieve information they are not allowed to access.

4. THE PROPOSED APPROACH

In this section, we explain architecture for supporting complex queries and access policies for multi-user encrypted databases stored in outsourced environments, where the DBUs/DBAs can query without disclosing to the CS any information about the query or the records stored in the database. The main idea behind our approach is to employ the proxy encryption scheme (described in Section 5), where the DBU (or the DBA) performs one round of encryption while the CS performs another round of encryption before storing or querying data in the outsourced environment. In our proposed scheme, it is the DBA who creates or drops tables in the database. Furthermore, the DBA is responsible for regulating access on encrypted tables stored in the database. The CS performs authorisation checks before providing DBAs access to the requested data.

Figure 1 illustrates the proposed architecture for protecting data in the outsourced environment. For adding a new table in the database, the DBA issues a *create* Command (i) using the *DBA Client*. The *Command Re-writer* module

Table 2: A sample table schema indicating column name, its type and size all stored in the Data Dictionary, where String size represents number of characters while size of Boolean and Integer represents number of bits

Table	Column	Type	Size
Personnel	Name	String	25
	Gender	Boolean	1
	Position	String	15
	Level	Integer	2

Table 3: An example of a table *Personnel* with four columns

Personnel			
Name	**Gender**	**Position**	**Level**
Andy	Male	Manager	2

of the DBA Client performs the first round of proxy encryption and sends the Command to the CS (ii). The *Command Processor* module on the CS receives (ii) Command and performs the server side round of encryption before storing/retrieving scheme of new/existing table in/from the *Data Dictionary* (iii). Table 2 illustrates a sample scheme of a table (i.e., *Personnel* shown in Table 3) stored in the Data Dictionary. In the schema of Personnel table, we can notice four columns including Name, Gender, Position and Level. The data type of Name and Position is String, Gender is Boolean and Level is of type Integer. For a String, size represents number of characters while size of Boolean and Integer represents number of bits.

Table 4: Access right matrix showing DBUs (in rows) access on tables (in columns)

DBU \ Table	*Personnel*	*Customer*
Alice	Grant	Deny
Bob	Deny	Grant

In the proposed architecture, it is the DBA who defines (updates or deletes) access policies for tables in the database. The Command Processor on the CS processes commands concerning access policies and stores/retrieves those policies in/from the *Access Policy* Repository (iv) after performing the server side round of proxy encryption. For managing access policies, we consider access right matrix, where each row represents capability of a DBU and each column represents the access control list of a table. For each new table, the DBA adds a column, specifying who can get access. Similarly, the DBA specifies access for a new user by adding a new row in the access control matrix. Table 4 shows the access control matrix indicating DBUs *Alice* and *Bob* have access to *Personnel* and *Customer* tables, respectively. There are three such matrices; one for read (SELECT), one for write (INSERT) and one for delete. The update operation is equivalent to both read and write. That is, a DBU has update access if she has both read and write access. All three matrices are stored in the Access Policy repository managed by the CS.

A Command Processor may send a (v) Response after running the requested command, which could be read/write access policies or schema stored in the Data Dictionary. The *Response Processor* on the DBA Client performs the client side round of proxy re-encryption, processes the response based on original (ii-b) Command, and finally displays a (vi) Response back to the DBA.

A DBU can issue a (i) *Query* using the *DBU Client*. The query can be SELECT, INSERT, UPDATE or DELETE for processing data files. However, a DBU has SELECT (read-only) access to the Data Dictionary. Before leaving the trust environment, the issued query is first encrypted by the DBU Client (2). The *Query Processor* on the CS receives (2) Query and performs the server side round of proxy encryption. Before executing the query, the Query Processor checks the access rights corresponding to the DBU who issued the query. For this reason, the Query Processor sends the (3) Access Request to the *Access Manager* that is responsible to evaluate the access policies. For this evaluation, the Access Manager retrieves the (4) Access Policy from the Access Policy Repository and sends the (5) Access Response back to the Query Processor. The Query Processor executes the query if the Access Response is *grant* and sends (6) Metadata or (7) Data in (8) Result to the DBU Client after performing server side round of proxy encryption. The Result Processor of the DBU Client processes the results while taking into account the original issues (2-b) Query and finally displays the (9) Result to the DBU.

The data including data files, data dictionary and access policy may leak sensitive information if stored in the cleartext. In the following, we elaborate how we protect confidentiality of the data in outsourced environments.

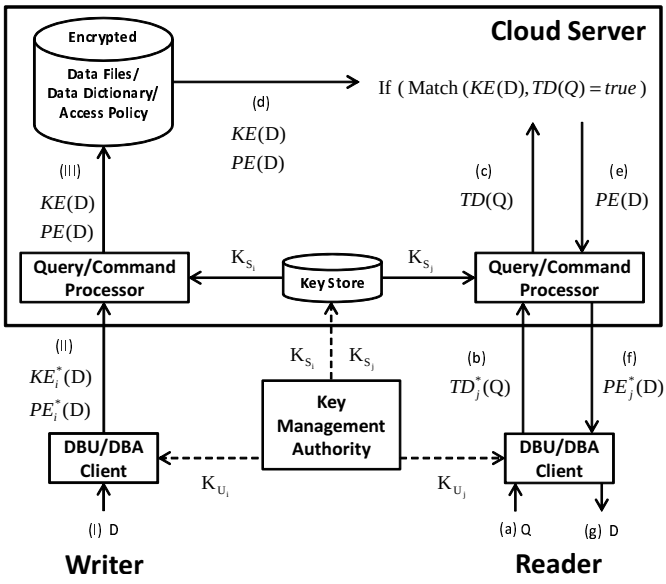

Figure 2: The overview of detailed architecture illustrating key distribution and both writer and reader processes using two variants of encryption (including KE and PE) that are employed

5. SOLUTION DETAILS

This section describes our full-fledged multi-user searchable scheme supporting complex encrypted queries over the encrypted database stored in the cloud. Among the full-fledged multi-user schemes we surveyed, SDE [19] is the most efficient for encrypted search. However, SDE supports only equality. Furthermore, we support the access control. In

this section, we describe how we extend SDE with a complex access structure inspired from the ABE [34, 37]. The ABE uses an access structure able to support conjunctions and disjunctions of equalities and inequalities. Nevertheless, it does not guarantee confidentiality of the access structure or of the attributes that are tested on the structure, being unsuitable for encrypted search. In the following, we propose a practical solution that provides confidentiality to the access structure by using SDE.

The proposed solution is based on two variants of encryption scheme including Keyword Encryption (KE) and Proxy Encryption (PE), initially proposed in SDE [19]. The KE scheme is employed for supporting encrypted match (equality only) while the PE scheme is used for protecting data in outsourced environments. That is, the PE scheme is employed for retrieving the data. Figure 2 shows the detailed architecture illustrating both KE and PE schemes for writer and reader processes. For performing match using the KE scheme, the query is transformed into trapdoors, where a trapdoor is implemented using the one-way function. In both KE (including trapdoor) and PE schemes, the client side performs one round of encryption while the server side performs second round of encryption as shown in Figure 2. The KE and PE schemes, and trapdoor representation does not leak information about database or query, respectively.

Table 5: Description of frequently used symbols and functions

Symbol	Description
K_{U_i}	User key corresponding to DBU/DBA i
K_{S_i}	Server key corresponding to DBU/DBA i
$KE(.)$	Keyword Encryption
$PE(.)$	Proxy Encryption
$TD(.)$	Trapdoor

In the following, we discuss various aspects of the proposed system. The frequently used symbols and functions are described in Table 5.

Key Distribution: The KMA generates keys and distributes them to DBUs/DBAs and the CS. For each DBU or DBA, the KMA generates a pair of key including a user key and a server key and sends to the DBU/DBA and the CS respectively (as illustrated with dotted lines in Figure 2). The CS stores the keys in *Key Store*.

Table 6: An example of table Personnel encrypted using proxy encryption and stored on the CS

$\{\textbf{Personnel}\}_{KE}$			
$\{\textbf{Name}\}_{KE}$ / $\{\textbf{Name}\}_{PE}$	$\{\textbf{Gender}\}_{KE}$ / $\{\textbf{Gender}\}_{PE}$	$\{\textbf{Position}\}_{KE}$ / $\{\textbf{Position}\}_{PE}$	$\{\textbf{Level}\}_{KE}$ / $\{\textbf{Level}\}_{PE}$
$\{\text{Andy}\}_{KE}$ / $\{\text{Andy}\}_{PE}$	$\{\text{Male}\}_{KE}$ / $\{\text{Male}\}_{PE}$	$\{\text{Manager}\}_{KE}$ / $\{\text{Manager}\}_{PE}$	$\{1^*\}_{KE}$, $\{*0\}_{KE}$ / $\{2\}_{PE}$

Data Writing Process: A DBU or a DBA can serve as a writer to write data, which will be stored in the outsourced environment as shown in Figure 2 (I). More precisely, it could be an INSERT (or UPDATE) query issued by a DBA or a command to define (or update) Access Policy or Data Dictionary. For simplicity, let us consider that a DBA has issued INSERT (Andy, Male, Manager, 2) query to write a record in table Personnel, shown in Table 3. Since each column can be searched or retrieved, it will be encrypted with both KE and PE, respectively. Please note that Figure 2 is illustrating only a single element. The DBA Client will perform the first round of encryption using the user key and will send the user encrypted data to the CS (II). The CS will perform the second round of encryption using the server side key corresponding to the writer before storing the data (III). Table 6 shows how (encrypted) data will be stored on the CS. There is clearly data expansion that we can realise from one to two tables, two rows, two columns or two cells. To apply the KE scheme, each string value is represented with a single element and each numerical value is represented as a bag of bits, where each numerical value of size s bits is represented with s elements as shown in Table 6. That is, one numerical value of size s bits is equal to s string values.

In case if the data is a command to write schema in the Data Dictionary then only table name (first column in Table 2) needs to be encrypted with both KE and PE while rest of columns do not require KE as those are not searched. Similarly, only table names are protected in access matrix shown in Table 4.

Data Reading Process: As a reader, a DBU or a DBA can query and read data stored in the outsourced environment as shown in Figure 2 (a). The DBU/DBA Client can transform plaintext query into trapdoor before sending to the CS (b). The trapdoor representation does not leak information on the plaintext query. The Query/Command Processor receives client encrypted trapdoor and generates server side trapdoor (c). The CS performs match between (KE) encrypted data and trapdoor. If the match is successful, the CS selects the (PE) encrypted data (e), pre-decrypts it and sends to the DBU/DBA Client (f). Finally, the DBU/DBA Client can user her key to decrypt the data (g).

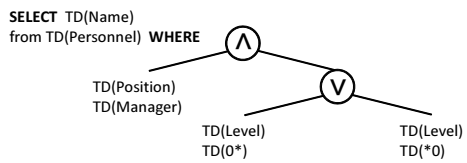

Figure 3: An example of query encryption where original query is *SELECT name from Personnel WHERE Position = Manager and Level < 3*

Query Representation: For supporting complex queries, we use access structure used in the ABE [34, 37], which can represent conjunctions and disjunctions of equalities and inequalities. In our proposed scheme, we transform each query into trapdoors. For each string field in query, the client generates a single trapdoor while for each numerical field in query, one or more trapdoors are generated. We use bag of bits to represent numerical fields. For a numerical field of size s bits, s trapdoors are generated in the worst case. The WHERE-clause of the query is represented as a tree, where each leaf node represents field name and value and non-leaf node represent conjunctions and disjunctions. Figure 3 illustrates an example of an encrypted query, where the original query is for retrieving names of managers whose level is less than 3, from the Personnel table.

Query Evaluation: To evaluate encrypted queries, the CS

checks if WHERE-clause of the query satisfies any encrypted record stored on the CS. For each encrypted leaf node in the WHERE-clause, the CS first find column and then compares its value. The leaf node will be considered as satisfied if there is a match between value in the query and value in the database record. The non-leaf node will be considered as satisfied based on satisfaction of its children as discussed in Section 6.2. A record will be selected if and only if the root node of the WHERE-clause tree is considered as satisfied. Finally, only matching records are sent back to the client.

Policy Enforcement: The Access Manager on the CS performs authorisation check before granting requested operations. For the sake of simplicity, we did not show the Access Manager in Figure 2; however, it can be seen in the abstract architecture shown in Figure 1. Let us consider the encrypted query illustrated in Figure 3. Before executing this query, the CS will first check whether the DBU who has issued the query has access on requested table. This is accomplished by matching trapdoor of table (i.e., TD(Personnel)) against encrypted table names (i.e., KE(Personnel)) in the access control matrix for which the DBU has 'grant'. The query is executed if and only if the match is successful.

Supporting Joins: In the proposed scheme, we can support joins using trapdoors of columns that could be joined. To support joins, a DBU has to generate trapdoor of the column value in addition to use KE and PE schemes. Given two columns to be joined, the trapdoor value of one column will be matched against the KE value of the other column.

User Revocation: For certain reasons (such as in case of compromise), a DBA/DBU can be removed from the system. For any revocation, the KMA interacts with the Access Manager to remove the server side key of the compromised DBAs/DBUs from the Key Store on the CS. Therefore, the Access Manager ensures that the server side key is not removed before granting access to the DBA/DBU.

6. DEFINITIONS AND CONSTRUCTION DETAILS

This section provides some definitions and the construction details of algorithms used in different phases.

6.1 Definitions

Our propsoed scheme consists of the following algorithms:

- $Init(1^k)$. The KMA runs the initialisation algorithm in order to generate the public parameters $Params$ and the master secret key set MSK after taking the security parameter k as input.
- $KeyGen(MSK, i)$. The KMA runs the key generation algorithm to generate a keying material for DBUs and DBAs. For each user (i.e., a DBU/DBA) i, this algorithm generates two key sets K_{U_i} and K_{S_i}. K_{U_i} is the key set for user i while K_{S_i} is the corresponding server side key for user i.
- $User\text{-}KW\text{-}Enc(D, K_{U_i})$. A user i runs the keyword encryption algorithm to make encrypted data searchable. It takes data D and user key K_{U_i} and results in $KE_i^*(D)$.
- $Server\text{-}KW\text{-}Re\text{-}Enc(KE_i^*(D), K_{S_i})$. The server re-encrypts the user encrypted searchable data $KE_i^*(D)$. The server retrieves the key K_{S_i} corresponding to user i and computes the re-encrypted keyword $KE(D)$.
- $User\text{-}P\text{-}Enc(D, K_{U_i})$. A user i runs the data encryption algorithm to encrypt data D using user key K_{U_i}. This results in ciphertext $PE_i^*(D)$.
- $Server\text{-}P\text{-}Re\text{-}Enc(PE_i^*(D), K_{S_i})$. The server re-encrypts the user encrypted data $PE_i^*(D)$. The server retrieves the key K_{S_i} corresponding to user i and computes the re-encrypted ciphertext $PE(D)$.
- $Query\text{-}Enc(Q, K_{U_j})$. A user j runs this algorithm to encrypt the query. It takes as input query Q and user key K_{U_j} and outputs the encrypted query $TD_j^*(Q)$.
- $Query\text{-}Re\text{-}Enc(TD_j^*(Q), K_{S_j})$. The server runs this algorithm to re-encrypt the query. It takes as input the user encrypted query $TD_j^*(Q)$. The server retrieves the key K_{S_j} corresponding to user j and computes the re-encrypted query $TD(Q)$.
- $Match(KE(D), TD(Q))$. The server runs this algorithm to perform search on encrypted data. It takes as input the encrypted data $KE(D)$ and the encrypted query $TD(Q)$. It returns *true* if there is a match and *false* otherwise.
- $Search(R, CQT)$. The server performs match between record R (a set of encrypted keywords $\{KD(D_1), KD(D_2), \ldots, KD(D_n)\}$) and the complex query tree CQT. If record satisfies the complex query tree, it returns *true* and *false* otherwise.
- $Server\text{-}Pre\text{-}Dec(PE(D), K_{S_j})$. The server runs this algorithm to partially decrypt the encrypted data for the user j. It takes as input $PE(D)$. The server retrieves the key K_{S_j} corresponding to user j and computes the pre-decrypted data $PE_j^*(D)$.
- $User\text{-}Dec(PE_j^*(D), K_{U_j})$. The user runs this algorithm to decrypt the data. It takes as input the pre-decrypted data $PE_j^*(D)$ and user key K_{U_j} and decrypts data D.
- $Revoke(i)$. The Access Manager on the CS runs this algorithm to revoke user i access to the data. Given i, the Access Manager removes K_{S_i} from the Key Store.

6.2 Construction Details

In this following, we provide the concrete construction of each algorithm.

- $Init(1^k)$. The KMA takes as input the security parameter k. It outputs two prime numbers p, q such that q divides $p-1$, a cyclic group $\mathbb{G}$ with a generator g such that $\mathbb{G}$ is the unique order q subgroup of Z_p^*. It chooses a random x from Z_q^* and outputs $h = g^x$. Also, it chooses a collision-resistant hash function H, a pseudorandom function f and a random key s for f. Finally, it publishes the public parameters $Params = (\mathbb{G}, g, q, h, H, f)$ and keeps securely the master secret key $MSK = (x, s)$.

- $KeyGen(MSK, i)$. For each user i, the KMA chooses a random x_{i1} from Z_q^* and computes $x_{i2} = x - x_{i1}$. Next, it transmits $K_{U_i} = (x_{i1}, s)$ securely to user i and $K_{S_i} = (i, x_{i2})$ to the server.

- $User\text{-}KW\text{-}Enc(D, K_{U_i})$. It is run by user i before inserting a new record in the database. It makes encrypted data searchable and encrypts each element in the record. For each data element D, the user chooses a random r_D from $\mathbb{Z}_q^*$ and computes $\sigma_D \leftarrow f_s(D)$. Next, it computes $KE_i^*(D) = (\hat{c}_1, \hat{c}_2, \hat{c}_3)$, where $\hat{c}_1 = g^{r_D + \sigma_D}$, $\hat{c}_2 = \hat{c}_1^{x_{i1}}$ and $\hat{c}_3 = H(h^{r_D})$. Finally, the $KE_i^*(D)$ is sent to the server.

- $Server\text{-}KW\text{-}Re\text{-}Enc(KE_i^*(D), K_{S_i})$. It is run by the server to re-encrypt the user encrypted searchable data. For each encrypted element $KE_i^*(D)$, the server computes $KE(D) = (c_1, c_2)$, where $c_1 \leftarrow (\hat{c}_1)^{x_{i2}} \cdot \hat{c}_2 = \hat{c}_1^{x_{i1}+x_{i2}} = (g^{r_D+\sigma_D})^x = h^{r_D+\sigma_D}$ and $c_2 = \hat{c}_3 = H(h^{r_D})$.

- $User\text{-}P\text{-}Enc(D, K_{U_i})$. A user i encrypts data using proxy encryption. For each data element D, it chooses random r'_D from $\mathbb{Z}_q^*$. Next, it computes $PE_i^*(D) = (\hat{e}_1, \hat{e}_2)$, where $\hat{e}_1 = g^{r'_D}$ and $\hat{e}_2 = g^{r'_D x_{i1}} D$. Finally, $PE_i^*(D)$ is sent to the server.

- $Server\text{-}P\text{-}Re\text{-}Enc(PE_i^*(D), K_{S_i})$. It is run by the server to re-encrypt the user encrypted data. First, the server computes $(\hat{e}_1)^{x_{i2}} \cdot \hat{e}_2 = (g^{r'_D})^{x_{i2}} \cdot g^{r'_D x_{i1}} D$
$= g^{r'_D x_{i1} + r'_D x_{i2}} D = g^{r'_D x} D$ in order to get $PE(D) = (e_1, e_2)$, where $e_1 = g^{r'_D}$ and $e_2 = g^{r'_D x} D$.

- $Query\text{-}Enc(Q, K_{U_j})$. A user j runs this algorithm to encrypt the query. First, the user computes $\sigma_Q \leftarrow f_s(Q)$. Next, it chooses a random r_Q from $\mathbb{Z}_q^*$. Finally, it calculates $TD_j^*(Q) = (t_1, t_2)$, where $t_1 \leftarrow g^{-r_Q} g^{\sigma_Q}$ and $t_2 \leftarrow h^{r_Q} g^{-x_{j1} r_Q} g^{x_{j1} \sigma_Q} = g^{x_{j2} r_Q} g^{x_{j1} \sigma_Q}$.

- $Query\text{-}Re\text{-}Enc(TD_j^*(Q), K_{S_j})$. The server runs this algorithm to re-encrypt the query. The server computes $T \leftarrow t_1^{x_{j2}} \cdot t_2 = g^{x \sigma_Q}$ and returns $TD(Q) = T$.

- $Match(KE(D), TD(Q))$. The server runs this algorithm to perform search on encrypted data. The server checks whether $c_2 \stackrel{?}{=} H(c_1.T^{-1})$. On successful match, it returns *true* and *false* otherwise.

- $Search(R, CQT)$. The server performs match between record R (a set of encrypted keywords $\{KD(D_1), KD(D_2), \ldots, KD(D_n)\}$) and the complex query tree CQT, which is basically WHERE-clause. First, the server matches (using Match algorithm) each leaf node in tree with the encrypted keyword in record. If a match is found, the leaf node is considered as satisfied. For internal nodes, the algorithm checks if the number of children that are satisfied is at least the threshold value of the node, which is equal to number of children in case of AND and one in case of OR gates. On success, the internal node is considered as satisfied. If root node of the tree is satisfied it returns *true* and *false* otherwise.

- $Server\text{-}Pre\text{-}Dec(PE(D), K_{S_j})$. When a match is found, the server retrieves matching records requested by the user and partially decrypts using K_{S_j}. The ciphertext $PE(D)$ is decrypted as $e_2 \cdot (e_1)^{-x_{j2}} = g^{r'_D x} D \cdot (g^{r'_D})^{-x_{j2}} = g^{r'_D (x - x_{j2})} D = g^{r'_D x_{j1}} D$. The server sends to the user $PE_j^*(D) = (\hat{e}_1, \hat{e}_2)$, where $\hat{e}_1 = g^{r'_D}$ and $\hat{e}_2 = g^{r'_D x_{j1}} D$.

- $User\text{-}Dec(PE_j^*(D), K_{U_j})$. The user decrypts ciphertext as $\hat{e}_2 \cdot (\hat{e}_1)^{-x_{j1}} = g^{r'_D x_{j1}} D \cdot (g^{r'_D})^{-x_{j1}} = D$.

- $Revoke(i)$. The Access Manager on the CS runs this algorithm to revoke user i access to the data. Given i, the Access Manager removes K_{S_i} from the Key Store as follows: $K_S \leftarrow K_S \backslash K_{S_i}$, where K_S represents the Key Store and is initialised as $K_S \leftarrow \phi$.

7. SECURITY ANALYSIS

In this section, we provide an evaluation of the security of our proposed scheme. To support complex queries, we extended the multi-user SDE scheme from [19], which only supports keyword search. We represent the search query as an access tree [34, 37, 38] that can provide support for complex queries, such as disjunctions and conjunctions of equalities, inequalities and negations. The SDE construction is INDistinguishable under Chosen Plaintext Attack (IND-CPA) under the assumption the Decisional Diffie-Hellmann (DDH) problem is hard relative to the group $\mathbb{G}$ (defined in Section 6.2). We encrypt the access tree and the data stored in the database using SDE, hence our scheme is also IND-CPA secure. In the following, we provide a sketch of the proof that both variants of SDE including KE and PE are secure.

THEOREM 1. *If the DDH problem is hard relative to $\mathbb{G}$, then the keyword encryption scheme KE is IND-CPA secure against the server S, i.e., for all Probabilistic Polynomial Time (PPT) adversaries $\mathcal{A}$ there exists a negligible function negl such that:*

$$Succ^{\mathcal{A}}_{KE,S}(k) = Pr\left[\begin{array}{l|c} b' = b & \begin{array}{c} (Params, MSK) \leftarrow Init(1^k) \\ (K_{U_i}, K_{S_i}) \leftarrow KeyGen(MSK, i) \\ kw_0, kw_1 \leftarrow \mathcal{A}^{User\text{-}KW\text{-}Enc(\cdot, K_{U_i})}(K_{S_i}) \\ b \xleftarrow{R} \{0,1\} \\ c_i^*(kw_b) = User\text{-}KW\text{-}Enc(kw_b, x_{i1}) \\ b' \leftarrow \mathcal{A}^{User\text{-}KW\text{-}Enc(\cdot, K_{U_i})}(K_{S_i}, c_i^*(kw_b)) \end{array} \end{array}\right] < \tfrac{1}{2} + negl(k) \quad (1)$$

Proof. See Theorem 2 in [19].

THEOREM 2. *If the DDH problem is hard relative to $\mathbb{G}$, then the proxy encryption scheme PE is IND-CPA secure against the server S, i.e., for all PPT adversaries $\mathcal{A}$ there*

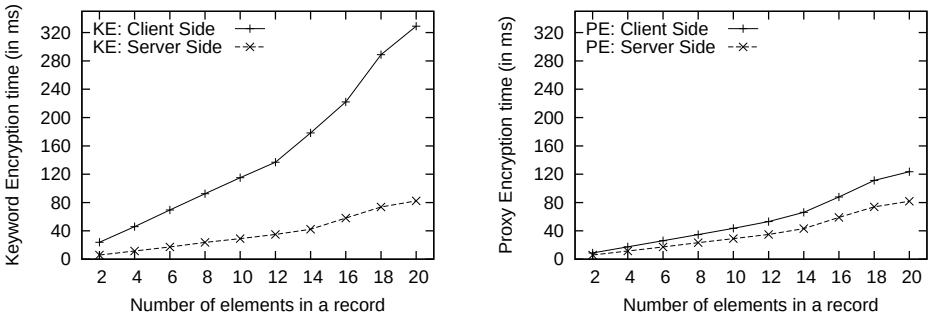

Figure 4: INSERT Query: Effect of number of elements in a record on (a) KE and (b) PE running time

exists a negligible function negl such that:

$$Succ^{\mathcal{A}}_{PE,S}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow Init(1^k) \\ (K_{U_i}, K_{S_i}) \leftarrow KeyGen(MSK, i) \\ d_0, d_1 \leftarrow \mathcal{A}^{User\text{-}P\text{-}Enc(\cdot, K_{U_i}}(K_{S_i}) \\ b \xleftarrow{R} \{0,1\} \\ c_i^*(d_b) = User\text{-}P\text{-}Enc(d_b, x_{i1}) \\ b' \leftarrow \mathcal{A}^{User\text{-}P\text{-}Enc(\cdot, K_{U_i})}(K_{S_i}, c_i^*(d_b)) \end{array}\right]$$

$$< \tfrac{1}{2} + negl(k) \tag{2}$$

Proof. See Theorem 1 in [19].

8. PERFORMANCE EVALUATION

In this section, we provide an evaluation of the overhead introduced by our encryption scheme. We implemented our scheme in Java. We tested the implementation of our scheme on a single node based on an Intel Core i7 2.67 GHz processor with 4 GB of RAM, running Microsoft Windows 7 Professional version Service Pack 1. In the following, all the results are averaged over 100 runs.

8.1 INSERT Query

In our proposed scheme, an INSERT query is encrypted in two steps, each step involving two encryption schemes including KE and PE. Figure 4 illustrates performance overhead of INSERT query. In our experiment, we observed effect of number of elements on KE and PE schemes on both client and server sides. As we can notice in Figure 4, the encryption time grows linearly with increase in number of elements in the record. In both KE and PE schemes, the client side incurs more overhead than that of the server side. The main reason is complex operations that are performed on the client side including random number generation. We increased number of encrypted elements from 2 to 20 and measured running time of encryption in milliseconds (ms). To encrypt 20 elements, the KE scheme takes approximately 330 ms (i.e., an average of 16.5 ms per element) and 82 ms (i.e., an average of 4.1 ms per element) on the client and server sides as shown in Figure 4(a), respectively. Similarly, we analysed the behaviour of the PE scheme. To encrypt 20 elements, the PE scheme takes approximately 123 ms (i.e., an average of 6.15 ms per element) and 81 ms (i.e., an average of 4 ms per element) on the client and server sides as shown in Figure 4(b), respectively. Each string field is represented as a single encrypted element while each numerical field of size s bits are represented with s encrypted elements. That is, one numerical field of size s bits is equivalent to s string fields. Asymptotically, INSERT query takes $\Theta(n + ms)$ on both client and server sides, where n and m represent number of string and numerical fields, respectively while s indicates size of each numerical field.

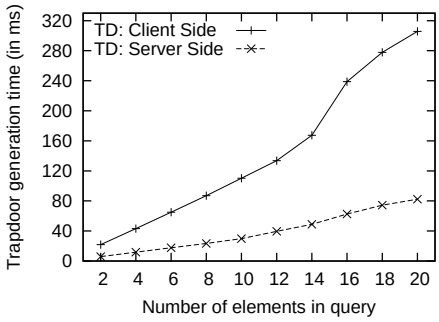

Figure 5: SELECT Query: Effect of number of elements in query on the trapdoor generation time

8.2 SELECT Query: Trapdoor Generation

A client's query is transformed into trapdoors. The WHERE-clause of the query is represented as a complex query tree, illustrated in Figure 3. Besides the table name and fields appearing with WHERE-clause, the query is dependent on number is elements in the leaf node of the WHERE-clause tree. Figure 5 illustrates the case when number of elements in query are increased from 2 to 20. For generating 20 trapdoors, the client and server sides take 305 ms (i.e., an average of 15.25 ms per element) and 83 ms (i.e., an average of 4.15 ms per element), respectively. Like the PE and KE schemes, trapdoor generation takes more time on client side than that of the server. The algorithmic complexity of trapdoor generation on both client and server sides is: $\Theta(n + ms)$.

8.3 Policy Enforcement

The server makes authorisation check before executing the requested query. For a requested client, the server lookups the corresponding (SELECT/INSERT) access control ma-

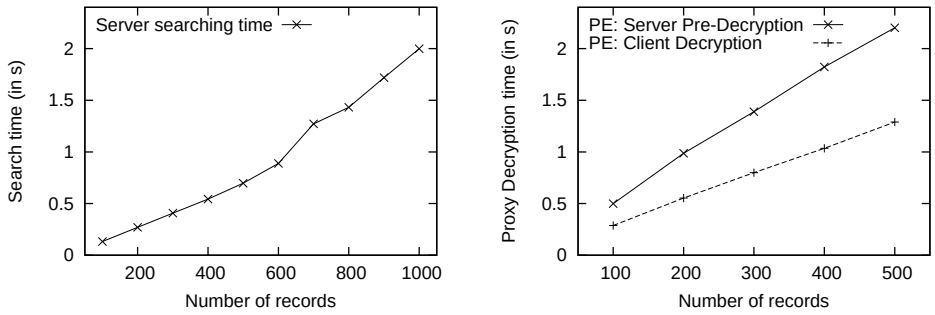

Figure 7: SELECT Query: Effect of number of records on (a) search and (b) proxy decryption time

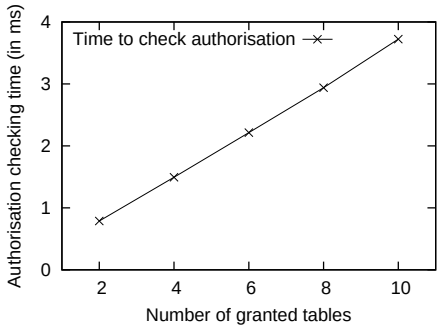

Figure 6: Effect of granted tables (in the access control matrix) on authorisation checking time

trix. The server matches encrypted trapdoor of table name in query with any encrypted table name for which the client has been granted access. If there is a match, the access is granted. Figure 6 shows overhead of encrypted matching between table name in query and table name in the access control matrix. In our experiment, we assumed 10 table names in the access control matrix and it took approximately 3.7 ms, an average of 0.37 ms per comparison. Overall, the time for checking the access rights is very small compared to other operations.

8.4 SELECT Query: Server Matching

The query matching is the most critical operation performed by the server. The server tests the encrypted query against an encrypted record stored in the database and then decrypts the matching records. Figure 7 demonstrates performance of query search and decryption. For this experiment, we consider query illustrated in Figure 3 (i.e., 5 comparisons per row in the worst case and 1 comparison in the best case, which is due to short-circuit evaluation of the query) and consider 1000 random records in the database. To assess the performance of our search operation, we matched query with records in the database. As we expected and can see in Figure 7(a), the searching time grows linearly with increase in number of records. For 1000 records, it took almost 2 seconds (s). For each string field in query, the server takes $\Omega(r)$, where r is number of records in the table. However, a numeric field in query is computationally extensive. For a numeric field of size s bits, the server takes $O(r \cdot s^2)$. The number of equality checks depends on the complexity of the query and the distribution of the data in the database. If there are n string and m numeric fields in query, the server will take $O(r \cdot (n + ms^2))$ in the worst case. In the best case, the server has to traverse r records, thus incurring $\Omega(r)$ overhead.

8.5 SELECT Query: Result Decryption

After finding out all matched records with query, the server pre-decrypts results and sends to the client, which finally retrieves them after performing decryption. The decryption time depends mainly on number of fields in a record and number of records. Figure 7(b) shows performance of result decryption. Assuming one field (i.e., name) in SELECT query as shown in Figure 3), the decryption time on the server and client sides is 2200 ms and 1290 ms for 500 records, respectively. In the worst case, the result decryption incurs: $O(r \cdot (n + m))$.

Table 7: Summary of complexity of executing queries on both client and server sides, where n represents number of string elements; m indicates number of numerical elements each of size s bits; and r represents number of records in the database

Query Type	Description	Best Case	Worst Case
INSERT	Record encryption	$\Theta(n + ms)$	
SELECT	Trapdoor generation	$\Theta(n + ms)$	
SELECT	Server matching	$\Omega(r)$	$O(r \cdot (n + ms^2))$
SELECT	Result decryption	$\Omega(1)$	$O(r \cdot (n + m))$

Table 7 summarises algorithmic complexities of both INSERT and SELECT queries. We can notice that the server side search operation is computationally extensive. In reality, we have implemented short-circuit evaluation of query that can speed up the performance in practical scenarios.

9. DISCUSSION

9.1 Fine-grained Access Control

In our proposed scheme, we have defined access control at the table level. Without loss of generality, we can extend access control at column level. However, the access control matrix will grow depending on the number of columns. As

a result, the CS will take more time to perform the authorisation check, which is the trade-off for supporting more fine-grained access policies.

9.2 Collusion Attacks

We assume that DBUs can collude; however, they cannot gain more than what each DBU can access individually because each one has her own private key and combination of those keys do not reveal any further information. On the other hand, a DBU (or a DBA) and the CS can collude together to gain unauthorised access to the data by combining their keys, where they can recover the master secret. For withstanding against this kind of collusion, one possibility is to assume multiple instances of the CS and split the server side key such that each instance gets one share. The main drawback of this approach is that it cannot work if all instances of the CS are compromised. Another approach is to provide protection with an extra layer of encryption say by employing Key-Policy ABE (KP-ABE) [34], which is collusion-resistant. We are planning to investigate this approach in future.

10. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the first multi-user scheme that supports complex SQL-like queries for encrypted databases stored in the cloud. State-of-the-art multi-user schemes are limited to keyword searches, making them not practical for real deployments. By representing queries as a tree, our scheme is able to support conjunctions and disjunctions of equalities and inequalities. Moreover, we support enforcement of access policies by the cloud server without leaking information about the data being protected. We have implemented and evaluated the overhead of our scheme. As future work, we plan to improve the performance of complex queries, where numerical fields are used. Another direction to improve the performance is by building indexes. After performance optimisations, we plan to apply the technique on very large databases.

11. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their feedback. This work has been partially supported by the TENACE PRIN Project 20103P34XC funded by the Italian MIUR.

12. REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pp. 44–55, 2000.

[2] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, (New York, NY, USA), pp. 216–227, ACM, 2002.

[3] E.-J. Goh, "Secure indexes." Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/`.

[4] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security* (M. Jakobsson, M. Yung, and J. Zhou, eds.), vol. 3089 of *Lecture Notes in Computer Science*, pp. 31–45, Springer Berlin Heidelberg, 2004.

[5] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security* (J. Ioannidis, A. Keromytis, and M. Yung, eds.), vol. 3531 of *Lecture Notes in Computer Science*, pp. 442–455, Springer Berlin Heidelberg, 2005.

[6] H. Wang and L. V. S. Lakshmanan, "Efficient secure query evaluation over encrypted xml databases," in *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pp. 127–138, VLDB Endowment, 2006.

[7] C. Bösch, R. Brinkman, P. Hartel, and W. Jonker, "Conjunctive wildcard search over encrypted data," in *Secure Data Management* (W. Jonker and M. Petkovic, eds.), vol. 6933 of *Lecture Notes in Computer Science*, pp. 114–127, Springer Berlin Heidelberg, 2011.

[8] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, (New York, NY, USA), pp. 85–100, ACM, 2011.

[9] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology - EUROCRYPT 2004* (C. Cachin and J. Camenisch, eds.), vol. 3027 of *Lecture Notes in Computer Science*, pp. 506–522, Springer Berlin Heidelberg, 2004.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, (New York, NY, USA), pp. 79–88, ACM, 2006.

[11] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography* (S. Vadhan, ed.), vol. 4392 of *Lecture Notes in Computer Science*, pp. 535–554, Springer Berlin Heidelberg, 2007.

[12] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Computational Science and Its Applications - ICCSA 2008* (O. Gervasi, B. Murgante, A. Laganà, D. Taniar, Y. Mun, and M. L. Gavrilova, eds.), vol. 5072 of *Lecture Notes in Computer Science*, pp. 1249–1259, Springer Berlin Heidelberg, 2008.

[13] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology - EUROCRYPT 2008* (N. Smart, ed.), vol. 4965 of *Lecture Notes in Computer Science*, pp. 146–162, Springer Berlin Heidelberg, 2008.

[14] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems and Software*, vol. 83, no. 5, pp. 763 – 771, 2010.

[15] B. Zhu, B. Zhu, and K. Ren, "Peksrand: Providing predicate privacy in public-key encryption with

keyword search," in *Communications (ICC), 2011 IEEE International Conference on*, pp. 1–6, 2011.

[16] Y. Yang, H. Lu, and J. Weng, "Multi-user private keyword search for cloud computing," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 264–271, 2011.

[17] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pp. 383–392, 2011.

[18] Y. Hwang and P. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Pairing-Based Cryptography - Pairing 2007* (T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, eds.), vol. 4575 of *Lecture Notes in Computer Science*, pp. 2–22, Springer Berlin Heidelberg, 2007.

[19] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," in *Data and Applications Security XXII* (V. Atluri, ed.), vol. 5094 of *Lecture Notes in Computer Science*, pp. 127–143, Springer Berlin Heidelberg, 2008.

[20] F. Bao, R. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Information Security Practice and Experience* (L. Chen, Y. Mu, and W. Susilo, eds.), vol. 4991 of *Lecture Notes in Computer Science*, pp. 71–85, Springer Berlin Heidelberg, 2008.

[21] J. Shao, Z. Cao, X. Liang, and H. Lin, "Proxy re-encryption with keyword search," *Information Sciences*, vol. 180, no. 13, pp. 2576 – 2587, 2010.

[22] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: management of access control evolution on outsourced data," in *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pp. 123–134, VLDB Endowment, 2007.

[23] M. Raykova, H. Zhao, and S. Bellovin, "Privacy enhanced access control for outsourced data sharing," in *Financial Cryptography and Data Security* (A. Keromytis, ed.), vol. 7397 of *Lecture Notes in Computer Science*, pp. 223–238, Springer Berlin Heidelberg, 2012.

[24] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pp. 720–731, VLDB Endowment, 2004.

[25] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, no. 3, pp. 333–358, 2012.

[26] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *INFOCOM, 2011 Proceedings IEEE*, pp. 829–837, 2011.

[27] Y. Lu and G. Tsudik, "Enhancing data privacy in the cloud," in *Trust Management V* (I. Wakeman, E. Gudes, C. Jensen, and J. Crampton, eds.), vol. 358 of *IFIP Advances in Information and Communication Technology*, pp. 117–132, Springer Berlin Heidelberg, 2011.

[28] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, (New York, NY, USA), pp. 965–976, ACM, 2012.

[29] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, (New York, NY, USA), pp. 563–574, ACM, 2004.

[30] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Advances in Cryptology - EUROCRYPT 2009* (A. Joux, ed.), vol. 5479 of *Lecture Notes in Computer Science*, pp. 224–241, Springer Berlin Heidelberg, 2009.

[31] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Advances in Cryptology - CRYPTO 2011* (P. Rogaway, ed.), vol. 6841 of *Lecture Notes in Computer Science*, pp. 578–595, Springer Berlin Heidelberg, 2011.

[32] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT 1999* (J. Stern, ed.), vol. 1592 of *Lecture Notes in Computer Science*, pp. 223–238, Springer Berlin Heidelberg, 1999.

[33] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Theory of Cryptography* (O. Reingold, ed.), vol. 5444 of *Lecture Notes in Computer Science*, pp. 457–473, Springer Berlin Heidelberg, 2009.

[34] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, (New York, NY, USA), pp. 89–98, ACM, 2006.

[35] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *Advances in Cryptology - CRYPTO 2009* (S. Halevi, ed.), vol. 5677 of *Lecture Notes in Computer Science*, pp. 108–125, Springer Berlin Heidelberg, 2009.

[36] S. D. C. di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Enforcing dynamic write privileges in data outsourcing," *Elsevier Computers & Security (COSE)*, 2013.

[37] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pp. 321–334, 2007.

[38] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, (New York, NY, USA), pp. 195–203, ACM, 2007.