ISSN:0975-9646

State-of-art in Storage Model using Encryption Technique for privacy preserving in Cloud Computing

V.K.Saxena[1] and Shashank Pushkar[2]

[1] *School of Engineering & Technology, Vikram University, Ujjain, M.P., India*
[2] *Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India*

***Abstract*: Now-a-days cloud computing is showing consistent growth in the field of computing. Users can utilize these services on pay-per-use basis. When data is exchanged in cloud, there exists the problem of disclosure of privacy. The idea is to build privacy preserving storage model where data sharing services can update and control the access and limit the usage of their shared data. Preserving privacy is an important issue for cloud computing and it needs to be considered at every phase of design. This paper proposes a metadata based data segregation and storage methodology along with an encryption technique to provide additional security. This would serve as a helping note in the progress of strengthening the privacy preserving approaches in cloud computing.**

Keywords: Cloud computing, data privacy, data security, data storage

I. INTRODUCTION

In large data centre, cloud computing moves the application software and databases, where the management of data and services are not reliable. This unique attribute poses many security challenges [15]. To realize the tremendous potential, business must address the privacy questions raised by the new computing model [4].

The metadata based storage model is based on the information which is valuable only as long as the fragments of the information are related to each other. For example, credit card information without its corresponding information like card holder name, Card Verification Value (CVV) and validity date is invaluable. The information becomes valuable only when these fragments of information are mapped. The mapped information about elements is required only for authenticated users and owners of the relevant information. In recent times, a well known instance of intrusion of user information is recorded by Sony PS Network [8].

In this situation, there is no necessity that data should be stored in a mapped manner, but the mapping is needed at the point of usage. Juels et.al., [10] described a formal "Proof of Retrievability" (POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error-correcting code to ensure both possession and retrievability of files on archive service systems. The time of usage of the information is apparently very less in comparison to the time that data is present at the storage location. Thus two types of security concerns arise. One concern is during data usage, i.e. during transmission and secondly, static phase of the data, i.e. during residing at storage centers. With respect to the data security during transmission in the cloud, Subashini S.,et.al.,[13] proposed a layered framework to deliver security as a service in cloud environment. This framework consists of a security service which provides a multi-tier security based on the need of the transaction. The framework provides dynamic security to users based on their security requirements, thus enabling localized level of security and thereby reducing the cost of security for applications requiring less security and providing robust security to applications. Hose et.al.[9] proposes a model to fragment data horizontally or vertically with relation to the tuples so that data can be accessed or updated in an optimized manner.

Subashini S., et.al.[14], proposed the model in which the data has to be segregated and further fragmented into smaller units until each fragment does not have any value individually. In addition to the fragmentation, we propose an encryption technique which provides additional security. This encryption allows only to data that is fragmented as 'sensitive' by the data migration environment.

Although existing privacy preserving query processing approaches, such as [1], [2], [3], [7], [11], [12], [16], can evaluate a query on randomized data, none of them can handle a series of queries, where some queries need other queries results as input. In [5], a symmetric searchable encryption scheme and an asymmetric searchable encryption scheme are proposed to store user's data in a third party. This paper proposes a secure query plan executor which can execute query plans without additional information about the data of data sharing services.

The rest of the paper is organized as follows: Section II presents system architecture, section III analyzes metadata based storage model, section IV provides the methodology, section V provides privacy preserving query plan with data storage, Section VI analyzes our approach for privacy preservation, section VII analyzes the stirring example for data privacy, and section VIII concludes the paper.

II. SYSTEM ARCHITECTURE

Now-a-days in existing data integration systems, it is understood that there is a central and trusted authority collecting all data from data sharing services and computing integration results for users based on the collected data. We assume that our data storage will correctly construct the query plans for user's requirements, decompose query plans, discover and fetch data from distributed data sharing services, amalgamate all data together, and, finally, return the final results to users. Further, we assume that our data storage is granted the access to the shared data by all data sharing services, and all shared data is well protected [17]. Our data storage consists of two components: the query plan wrapper and the query plan executor. The query plan wrapper is responsible for analyzing integration requirements and constructing query plans for the query plan executor. Since the wrapper development and optimization have been extensively studied [6], [18]. we assume that the query plan wrapper can select data sharing services and construct a query plan graph (Fig.3) from user's integration requirements. The query plan executor is responsible for executing query plans to fetch data from data sharing services and producing the final results.

III. Data Storage Model Based On Metadata

This paper describes the model which only deals with the data security at the storage centers. This in turn has two concerns: one issue is about the actual physical unit where the data is stored and the other one is the intrusion into the information. Our model is mainly focused in providing security in avoiding intrusion.

Here Data has to be segregated into Public Data Segment (PDS) and Sensitive Data Segment (SDS). The SDS has to be further fragmented into smaller units until each fragment does not have any value individually. Here the fragmentation need not be of multiple levels. Instead, effort has to be put in to identify the key element that makes the data sensitive and should be fragmented separately. Data fragmentation is shown in figure 1.

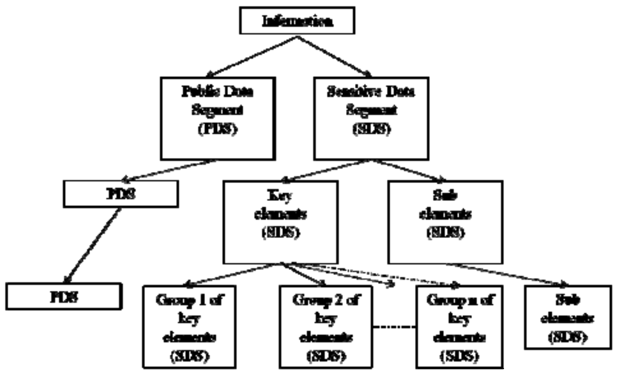
Information
Public Data Segment (PDS)
Sensitive Data Segment (SDS)
PDS
Key elements (SDS)
Sub elements (SDS)
PDS
Group 1 of key elements (SDS)
Group 2 of key elements (SDS)
Group n of key elements (SDS)
Sub elements (SDS)

Fig.1 Data Fragmentation

In this process, the value of the information is actually destroyed, but as and when fragmentation is done, the mapping data required to re-assemble the information should also be generated parallel. This can be done for database that is being designed from scratch. But, this is not effective for enterprises who want to move their existing data to the cloud. As a measure of migration of data from existing environment to cloud, the migration should be done appropriately. This can be made feasible by this model. For achieving this, we need a Data Migration Environment (DME) which does this job. The input to DME should be the existing schema of the database and additional information about the sensitive part of the schema should be given as Metadata to the DME. Then the DME can fragment the data into pieces based on the level of security needed. It will also prepare a mapping table to re-assemble the data.

IV. The Methodology

We can consider an example related with a customer database in a bank consisting of customer's information along with his credit card information. The schema for storing such information will be in the form of tables. Some tables containing personal information of the user and some tables containing information regarding to credit cards and will be mapped using their ids. This particular information can be stored in a **bankDB** database as follows:

Customertable (<u>CustomerId,</u> CustomerName, CustomerAddress, CustomerPhone, CustomerDOB)
Membershiptable (<u>**CustomerId**</u>, Password, PasswordQuestion, PasswordAnswer)
Creditcardtable (<u>CardId</u>, CreditcardNo, CardExpiryDate, CVVNo)
Customer_Creditcardtable containing (<u>CustomerId</u>, <u>CardId</u>)
Where the primary keys are underlined and foreign key are bold.

If an intruder wants to access this information, he can exploit particular database because all related information are stored at the same location. In this example, the Customer table contains data which is not of much importance. The Membership table taken individually does not have any value but along with the Customer table data, it is vital information for an intruder. The Credit card table is a sensitive data with high value because though there is no mapping done with the Customer table, for the intruder, it is a high potential target. For example, an online transaction can be done successfully with this data alone. The information of Customer table and Customer_Creditcard table taken together indicates heavy losses for the bank. Usually, the entire data is stored in a single database and on the same hardware resource.

This model enforces that the related data should be stored at different locations and should be mapped runtime either during update or query. Paper considers this entire model, which is to be migrated to our proposed model through the DME. The user has to supply the schema information together with its metadata of these tables to the DME. Let us consider only three categories of metadata for this example. The data which is having low value is considered as 'Normal'. The data which is having high value is considered as 'Critical' and the data which has value when mapped with other data are considered as sensitive. The data which maps 'Sensitive' or 'Critical' data to 'Normal' data is also considered 'Sensitive'. The metadata for our example are shown in Table 1.

Table 1: Metadata information

Table	**Metadata**
Customer	Normal
Membership	Sensitive
Credited	Critical
Customer_Creditcard	Sensitive

The work of DME starts now. It has to fragment this data. For this fragmentation the DME should be able to be configured or customized with respect to the level of security required. In our example, DME provide medium level security and it should fragment only data which are of 'Critical' criteria. For high level security, it should fragment data present in both 'Critical' and 'Sensitive' criteria. The DME is not aware of the actual data residing within these tables. Hence along with the metadata of the tables, the primary key name should be provided in addition to it. This is easily available with the schema information of the database tables. Now DME configured the different levels of security needed and their corresponding metadata. For medium security in our database, the DME can fragment only the data that is 'Critical'. In our example, we have one 'Critical' data set. The corresponding table is Credit card table and the primary key of this table is CreditcardId. In the first step of DME it fragments this table as below:

DME_Creditcardtable(<u>SensitiveId</u>, CreditcardNo, CardExpiryDate)
DME_Creditcard_Senstivetable (<u>**SensitiveId**</u>, CVVNo)
DME_Creditcard_Mappertable (<u>CreditcardId</u>, SensitiveId)

Here in above two tables, primary and foreign keys are created by DME.
The data of the above three tables will fall under the 'Sensitive' category of metadata. In the current situation Table 2 shows the metadata.

Table 2: Metadata information after fragmentation

Table	**Metadata**
Customer	Normal
Membership	Sensitive
DME_Credited	Sensitive_DME
Customer_Creditcard	Sensitive
DME_Creditcard_Sensitive	Sensitive_DME
DME_Creditcard_Mapper	Sensitive_DME

The DME segregates the schema by separating out the data modified by DME, 'Originally Sensitive' data and 'Normal' data as shown in Table 3 after fragmentation is completed.

Table 3: Segregated schema

Normal	**Originally sensitive**	**Sensitive DME**
Customer	Membership	DME_Creditcard
	Customer_Creditcard	DME_Creditcard_Sensitive
		DME_Creditcard_Mapper

The 'Sensitive DME' data is then split into Actual Data (AD) and Mapper Data (MD):
Sensitive DME

- Actual Data (DME_Creditcard, DME_Creditcard_Sensitive)
- Mapper Data (DME_Creditcard_Mapper)

Here the DME then moves the 'Normal' data to one database and 'Originally Sensitive' data to another database, and AD of 'Sensitive DME' data to another database at different location and MD of 'Sensitive DME' to the database with 'Normal' data. If DME creates its own table with respect to the AD, then this table will be the most sensitive data and will be stored in a different location. Here different location means the different server at the same geographical location or at different geographical location. Now one more mapping is required for mapping the original table with the fragmented data set. This fragmented data set can be stored in a separate table. Now the fields of our database are looks like the following:

Server 1

- bankDB
  Customertable(<u>CustomerId</u>, CustomerName, CustomerAddress, CustomerPhone, CustomerDOB)
- bankDB_DME
  Membershiptable(<u>**CustomerId**</u>, Password, PasswordQuestion, PasswordAnswer)
  Customer_Creditcardtable (<u>CustomerId</u>, <u>CardId</u>)
  DME_Creditcard_Mappertable (<u>CreditcardId</u>, <u>*SensitiveId*</u>)
  DME_Mappertable (<u>OriginalTableName</u>, <u>NewTable Name</u>)
  Where sensitiveId is created by DME

Server 2

- DME_Creditcardtable(<u>*SensitiveId*,</u> CreditcardNo, CardExpiryDate)
  Where sensitiveId is created by DME

Server 3

- DME_Creditcard_Senstivetable (<u>**SensitiveId,**</u> CVVNo)
  Where sensitiveId primary and foreign key created by DME

After this separation of the fields, the DME_Mapper table is shown in Table 4.

Table 4: DME_MAPPER Table

OriginalTableName	**NewTableName**
Creditcard	DME_Creditcard
Creditcard	DME_Creditcard_Sensitive
Creditcard	DME_Creditcard_Mapper

Here each database contains data which does not have value in itself. The entire mapping is done only during runtime and the value is built up temporarily during access and update and later its value is destroyed. During the static phase of the life cycle of the data, when an intruder wants to get access to the data, he can't use the data to exploit the information by any way. The integrity between the original schema and the new schema can be taken care by deploying a database runtime migration environment which will deploy all the logics required for the runtime generation of schema.

V. PRIVACY PRESERVING QUERY PLAN WITH DATA STORAGE

In proposed system, as in the Fig. 2, data storage will collect only the data required for user's requests. To formulate the privacy preserving data integration across data sharing services in the cloud, it is needed to define the query plan [17]:

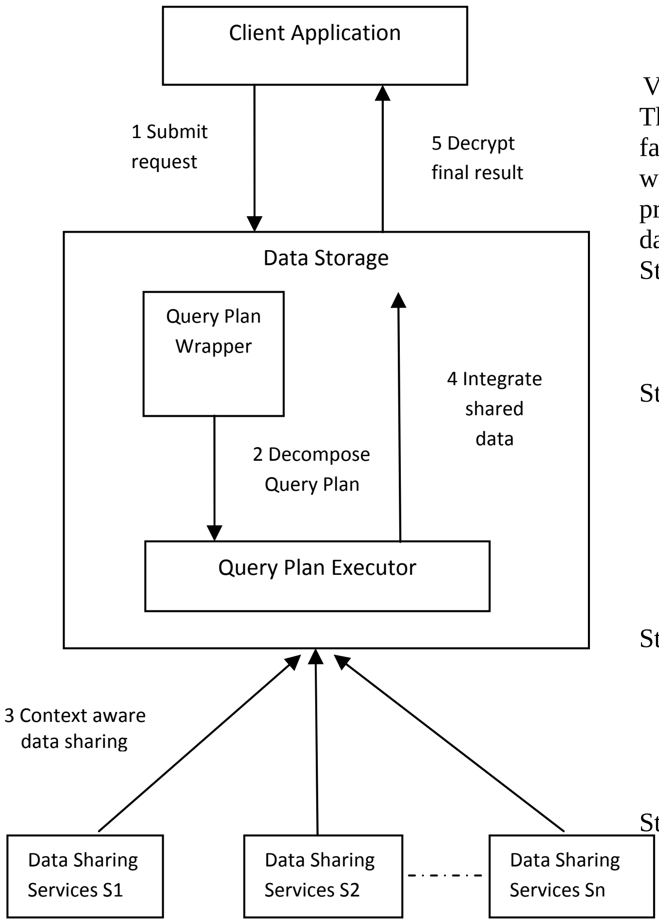

Fig.2 Our privacy preserving repository for data integration across data sharing services

Definition of Query Plan: A query plan P is a partially ordered set of queries (p_1; p_2; …..; p_m) with two properties:

1. Each p_i can be evaluated only after all of its precedent queries have been evaluated.

2. The data from the data sharing services can be directly used by each p_i or the precedent queries' outputs can be used as inputs.

The output of p_i with no succeeding queries is the final result of P, and all other queries' outputs are intermediate results. From the above definition, it indicates that a query plan P has a much richer structure than a single query or a set of independent queries. First, there is a partial order relation among queries in P. Second, only the outputs of queries in P without successive queries constitute the final result and all other intermediate results should be protected. Consequently, we have the following definition:

Definition of Privacy Preserving Data storage: For a query plan P = (p_1; p_2; ….; p_m) and a data storage STOR, where STOR is a privacy preserving data storage for data integration, if STOR executes P in a privacy preserving manner as follows:

- STOR has only P's final result encrypted with user's public key and has no information on P's intermediate results
- STOR cannot use the data shared for P to evaluate any other queries.

VI. PROPOSED APPROACH FOR PRIVACY PRESERVATION

Then objective of the paper is to build up data storage to facilitate the data integration and sharing across cloud along with preservation of data confidentiality. We present the process of the data integration via our privacy preserving data storage STOR. The process is as follows:

Step 1: The user sends his/her public key P_k and the requirements about data integration to our repository STOR.

Step 2: The query plan wrapper of STOR analyzes the user's integration requirements and converts them to a query plan graph G, and then decomposes G to a set of sub-graphs (G_1;G_2; ….;G_m) and sends the sub-graphs to the query plan executor. Every sub graph G_i represents the context of one data sharing service for conducting context-aware data sharing.

Step 3: For every G_i, the query plan executor looks for the corresponding data sharing service S_i and sends G_i to S_i, which prepares the data using the Context-Aware Data Sharing concept and returns all randomized data to the query plan executor.

Step 4: The query plan executor integrates all returned data to execute the G and outputs the results FResult of user's request, which is encrypted with the user's public key P_k.

Step 5: STOR sends Final Result to the user who then decrypts it with his/her secret key S_k.

This proposed scheme is secure under the standard security model. This scheme is also able to support user accountability. Whether we are assembling, managing or developing on a cloud computing platform, we need a cloud compatible database. It also supports other cloud objectives such as lower costs for hardware, maintenance, tuning and support.

VII. AN STIRRING EXAMPLE FOR DATA PRIVACY

Here, we consider an example of customer database in a bank which include customer, membership and credit card information and a location database. We assume that database C1 (CNAME, CNO) storing customer's name and corresponding credit card no, database C2 (MNO, CNO) storing the corresponding membership no. and credit card no., database C3 (MNO, CPLACE, PID) storing customer's credit card details with password-id and a location database C4 (CNAME,CLOC) storing the location of customers. The database schema and data are shown in Table 5. This example is explained in terms of four SQL queries shown in Table 6, where Q1, Q2, and Q3, generate three temporary tables, Tmp1, Tmp2, and Tmp3 respectively, and the last query, Q4 outputs the final results.

Table 5 Database in the example (C1, C2, C3, and C4)

Customer table
C1(CNAME, CNO)

CNAME	CNO
Mukto	P1
Mukto	P2
Parth	P3
Abhi	P4

Membership table
C2(MNO,CNO)

MNO	CNO
MNO1	P1
MNO2	P1
MNO3	P2
MNO4	P3

Credit card table
C3(MNO,CPLACE,PID)

MNO	CPLACE	PID
MNO1	P1	ID1
MNO2	P1	ID2
MNO3	P2	ID3
MNO4	P3	ID4

Location table
C4(CNAME, CLOC)

CNAME	CLOC
Mukto	IND
Mukto	POL
Parth	USA
Abhi	BRZ

Table 6 Queries required by the example

Q1–>Tmp1
SELECT C1.CNO
FROM C1
WHERE C1.CNAME= "Mukto"
	Q2–>Tmp2
	SELECT C2.MNO
	FROM Tmp1, C2
	WHERE Tmp1.CNO= "C2.CNO"

Q3–>Tmp3
SELECT C4.CLOC
FROM C4
WHERE C4.CNAME= "Mukto"
	Q4
	SELECT C3.PID
	FROM Tmp2, Tmp3, C3
	WHERE C3.MNO= Tmp2.MNO
	AND C3.CPLACE=Tmp3.CLOC

All the queries are executed by Table 5. However, our data storage is allowed to collect only the needed information. On the other hand our data storage will randomize Q1's result and make the randomized result still usable for Q2 because the data storage needs some extra information to execute queries, such as Q1's result, which is needed by Q2 as an input.

Here, in this example, to protect Q1's result i.e. {P1; P2} without disabling Q2, {P1; P2} is replaced by {H(P1);H(P2)}, where H is a hash function. Because the hashed patterns will usually remain unique, the data storage can evaluate Q2 by comparing H(Tmp1;CNO) and H(T2;CNO). This simple hash solution can avoid the need for our data storage to know Q1's results, but still keep the mapping relation between names and customer's CNOs. Since H(P3) does not appear in the Q1's hashed result {H(P1);H(P2)}, our data storage can find that the customer with CNO4 is not having name Mukto. To protect the privacy of such information, the concept of Context-Aware Data Sharing is used to randomize Q1's result. The context awareness implies that when a bank shares its database C1 with our data storage, it should know that its customer number (CNO) data will be used to match the customer number data from C2. While the simple hash solution only randomizes the items in Q1's result (i.e., P1; P2). Our Context-aware data sharing concept randomizes all CNOs in C1, but ensures that only P1 and P2 can be used to evaluate Q2. Hence, the mapping between names and CNOs are well protected. In the context of the above example we illustrate how our data storage is being used for developing of various credit cards at different locations and how the data cannot disclose any additional information about the data of databases C1, C2, C3, and C4.

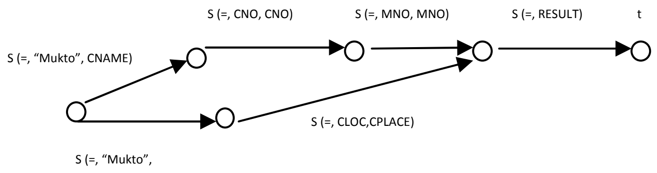

Fig.3 The Query Plan graph of example

VIII. Conclusion

The paper presented a privacy preserving data storage to integrate data from various data sharing services. In contrast to existing data sharing techniques, our data storage only collects the minimum amount of information from data sharing services based on user's integration requests, and data sharing services can restrict our data storage to use their shared information only for user's integration requests. Metadata based model will take some quantifiable effort to be implemented in real time, it provides necessary solution for an environment like cloud computing. This paper is extended the outcome of Metadata based model. After fragmentation in this model, by applying an encryption technique, the privacy of the data can be preserved more efficiently.

References

[1] Agrawal, R., Evfimievski, A.V., and Srikant, R., Information Sharing across Private Databases, Proc. ACM SIGMOD, International Conf. Management of Data, SIGMOD'03, pp.86-97, 2003.

[2] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y., Order-Preserving Encryption for Numeric Data, Proc. ACM International Conf. Management of Data, SIGMOD'04, pp.563-574, 2004.

[3] Bellare, M., Boldyreva, A., and O'Neill, A., Deterministic and Efficiently Searchable Encryption, Advances in Cryptology, CRYPTO'07, pp.535-552, 2007.

[4] BNA, Privacy and Security Law Report, The Bureau of National Affairs Inc., 2009.

[5] Boneh, D., Sahai, A., and Waters, B., Functional Encryption: Definitions and Challenges, In proc.of TCC'2011, LNCS 6597, pp.253-273, 2011.

[6] Fischer, K.P., Bleimann, U., Fuhrmann, W., and Furnell, S.M., Security Policy Enforcement in BPEL-Defined Collaborative Business Processes, Proc. 23rd International Conf. Data Eng. Workshop, pp.685-694, 2007.

[7] Ge, T., and Zdonik, S.B., Answering Aggregation Queries in a Secure System Model, Proc. 33rd International Conf. Very Large Data Bases, VLDB'07, pp.519-530, 2007.

[8] Goodin, D., User data stolen in Sony PlayStation Network hack attack, Ars Technica, 2011.

[9] Hose, K. and Schenkel, R., Distributed Database Systems Fragmentation and Allocation, Distributed Database Systems, 2010.

[10] Juels, A. and Kaliski, B.S., Pors: Proofs of retrievability for large files, Proceedings of the 14th ACM Conference on Computer and Communications Security, ACM Press, USA, pp.584-597, Oct. 28-31, 2007.

[11] Lebanon, G., Scannapieco, M., Fouad, M.R., and Bertino, E., Beyond k-Anonymity: A Decision Theoretic Framework for Assessing Privacy Risk, Transactions on Data Privacy Journal, 2:3, pp.153-183, 2009.

[12] Lindell, Y., and Pinkas, B., Secure Multiparty Computation for Privacy-Preserving Data Mining, Journal of Privacy and Confidentiality, 1(1), pp.59-98, 2009.

[13] Subashini, S. and Kavitha, V., A survey on security issues in service delivery models of cloud computing, J. Netw. Comput. Applications, 34: pp.1-11, 2011.

[14] Subashini, S. and Kavitha, V., A metadata based storage model for securing data in cloud environment, American Journal of Applied Sciences, 9(9), pp.1407-1414, 2012.

[15] Wang, C., Wang, Q., Ren, K., and Lou, W., Ensuring data storage security in cloud computing, Proceedings of the 17th International Workshop on Quality of Service, Jul. 13-15, IEEE Xplore Press, Charleston. SC, pp.1-9, 2009.

[16] Xiong, L., Chitti, S., and Liu, L., Preserving Data Privacy for Outsourcing Data Aggregation Services, ACM Trans. Internet Technology, vol. 7. No. 3, pp. 17-45, 2007.

[17] Yau, S.S., and Yin, Y., A Privacy Preserving Repository for Data Integration across Data Sharing Services, IEEE Transactions on Services Computing, Vol 1, No.3. July-September, 2008.

[18] Yu, B., Li, G., Sollins, K.R., and Tung, A.K.H., Effective Keyword-Based Selection of Relational Databases, Proc. ACM International Conf. Management of Data, SIGMOD'07, pp.139-150, 2007.