Secure Data Storage Architecture on Cloud Environments

Tran Thi Xuan Trang[1] and Katsuhisa Maruyama[2]
[1]*Graduate School of Information Science and Engineering, Ritsumeikan University, Kusatsu, Japan*
[2]*Department of Computer Science, Ritsumeikan University, Kusatsu, Japan*
trang@fse.cs.ritsumei.ac.jp, maru@cs.ritsumei.ac.jp

Keywords: Cloud Data Storage, Data Confidentiality, Data Partitioning, Query Translation, Migration Approach.

Abstract: Securing sensitive customer data outsourced to external servers in cloud computing environments is challenging. To maintain data confidentiality on untrusted servers, conventional data security techniques usually employ cryptographic approaches. However, most enterprises are unwilling to employ these approaches if they require high-performance client devices to cipher the entire data. In this situation, separating out the confidential data is beneficial since only the confidential data are encrypted or stored in trusted servers. Although this idea has already been proposed, its support is still insufficient. This paper proposes a secure data storage model in cloud computing environments that is based on the concept of data slicing and presents its prototype tool that supports the low-cost migration of existing applications. Our tool provides a structured query language (SQL) translation mechanism that provides transparent access to partitioned data without changing the original SQL queries. A simple case study shows how the proposed architecture implements secure data storage in cloud computing environments.

1 INTRODUCTION

Cloud computing has become popular in many areas of business over the last few years. Many enterprises are excited about the potential to reduce information technology (IT) costs and obtain advantages in scalability, availability, and performance. The service models of cloud computing are categorized into three types (Mell and Grance, 2011; Zhang et al., 2010): Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). While SaaS allows enterprise customers to use the provider's applications running on a cloud infrastructure, PaaS and IaaS allows them to develop, run, and manage applications on the cloud. In any model, the enterprises can utilize their business applications without building and maintaining their own infrastructure. Thus, some of them promote the introduction of cloud computing and the migration of legacy applications to cloud environments (Menychtas et al., 2013; Tak and Tang, 2014).

However, these enterprises also face difficulties deploying their existing applications to cloud environments since they must put their sensitive data on cloud servers (Catteddu and Hogben, 2009; Jansen and Grance, 2011; Wei et al., 2014; Fernandes et al., 2014). The salient risk is that attackers or collusive cloud insiders can easily steal these sensitive data if the data protection mechanism is vulnerable to various kinds of attacks. Therefore, data confidentiality raises a great concern for enterprises. Most enterprise managers or developers are unwilling to store sensitive data in cloud storages.

Hybrid cloud (Mell and Grance, 2011), which mixes private cloud and public cloud services, is a possible option for enterprises that have already invested in a private virtualization environment. It manages some data (or resources) in-house and places other data in a public area. For example, an enterprise might use a public cloud service, such as Amazon Simple Storage Service (Amazon S3), to store archived data for backup, but continue to maintain in-house storage for sensitive data.

Although this strategy is reasonable, another problem might arise. A primary difficulty of hybrid cloud deployment is how to minimize changes in the existing enterprise applications. No matter how similar the public and private cloud services are, implementation-level gaps between the original and revised applications are inevitable, and filling these gaps is usually expensive. For example, the separation of sensitive data requires many changes to the implementation code of the original application when it is revised on cloud environments using conventional data storage models. This complicates the migration of existing enterprise applications to hybrid cloud en-

Trang, T. and Maruyama, K.
Secure Data Storage Architecture on Cloud Environments.
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 39-47
ISBN: 978-989-758-194-6

vironments. All enterprises desire an environment that optimizes scalability and cost-effectiveness offered by a public cloud service if they could easily maintain the security of their sensitive data. In other words, they require a secure data storage model that emphasizes the protection of data confidentiality and a mechanism that supports the deployment of their existing applications based on this model.

This paper proposes a new data storage model that can protect sensitive data in cloud computing environments. The model is based on data slicing, which is the simple concept of separating the sensitive and non-sensitive data appearing in enterprise applications. The sensitive data must be stored into secure storages located on trusted servers, which can then be encrypted. On the other hand, non-sensitive data can be stored into public storages located on untrusted servers. In general, sensitive data is a small portion of the whole data volume and the sizes of such data are small. Since a large volume of non-sensitive data can be stored in public storages that are cheaper than secure storages, enterprises can save data storage costs while preserving the security level of their data.

Although this new secure storage model is beneficial from an economic perspective, the model, in general, requires changes in implementation code with respect to data access in existing enterprise applications, as mentioned before. The tasks achieving these code changes are time-consuming and error-prone. To avoid such troublesome tasks, we propose a mechanism that provides transparent access to data stored in two kinds of partitioned storages with SQL queries. The importance of transparent access was inspired by the research work (Ferretti et al., 2013). The mechanism automatically translates the original SQL queries into ones that are compatible with the new storage model. Therefore, the existing applications running on private environments can run on hybrid environments without altering any implementation code. The original data is automatically stored into the partitioned storages and can be accessed from them. The applications do not need to know that the data are actually partitioned. We implemented this translation mechanism in our prototype tool. To demonstrate its feasibility, we migrated an open source application, Plandora, to the cloud environment.

In our paper, we:

- Propose a new data storage model that can maintain data confidentiality when enterprises migrate their existing applications to cloud environments.
- Present an SQL translation mechanism that allows the applications to transparently access the data stored in the two partitioned storages on trusted and untrusted servers.
- Show a running implementation that employs the SQL translation mechanism and supports the building of the initial database scheme based on the new model.

The rest of this paper is organized as follows: Section 2 introduces related work. Section 3 explains the proposed data storage model. Section 4 elaborates the implementation of the model. Section 5 presents a case study of migration with a tool implementing the proposed architecture. Section 6 summarizes this paper and outlines our future work.

2 RELATED WORK

Several security issues and challenges on the adoption of cloud computing have been reported (Catteddu and Hogben, 2009; Jansen and Grance, 2011; Subashini and Kavitha, 2011; Ren et al., 2012; Hashizume et al., 2013; Fernandes et al., 2014). Some of the reports emphasize data confidentiality for (untrusted) cloud storage.

The systems DEPSKY (Bessani et al., 2011) and iDataGuard (Jammalamadaka et al., 2008) guarantee data confidentiality and integrity on multi-tenant cloud services. The DEPSKY employs a secret sharing scheme to avoid storing clear data in untrusted clouds. The iDataGuard is a middleware that transparently provides customers with secure file systems. These systems are both based on data encryption using cryptographic techniques. Moreover, cloud database services (Hacigümüs et al., 2002) employ an encryption facility that can answer queries over encrypted data (Ferrari, 2009; Weis and Alves-Foss, 2011).

Most existing approaches prevent the disclosure of sensitive data stored in untrusted storage servers with proper encryption. On the other hand, a few approaches exploit fragmentation to avoid data encryption (Aggarwal et al., 2005; Ciriani et al., 2011). The basic idea standing behind these approaches is to split data among multiple subsets, each of which is managed by an independent cloud provider. A small portion of the data is stored in local trusted storage, while the major portion of the data is outsourced to external storage servers. The concept of fragmentation was also adopted into a metadata based storage model (Subashini and Kavitha, 2012). In this model, data has to be segregated and further fragmented into smaller units until each fragment does not have any individual value.

Ideally, data encryption is an obvious solution to protect sensitive data. However, it is not feasible for

every enterprise since the execution of queries on encrypted data still requires the computational overhead although homomorphic encryption techniques (Gentry, 2009; Gomathisankaran et al., 2011; Yu et al., 2012), which enable processing encrypted data without decrypting it, are available. Therefore, a data fragmentation approach that can remove or reduce encryption costs is likely to become an affordable option to outsource data. Nevertheless, there are not sufficiently adequate models based on data fragmentation, which achieve both the goals of data confidentiality and transparent migration for existing applications to cloud environments. A new model that we propose guards against the disclosure of sensitive data and helps an existing application migrate the data to cloud storage without requiring changes to its implementation code.

3 DATA STORAGE MODEL

In this section, we propose our secure data storage model and present the overall architecture using this model.

3.1 Data Slicing

The main idea of the new data storage model is to split the original data into sensitive and non-sensitive data. This technique is basically called data slicing (Ferretti et al., 2013). The split data are stored into two independent areas: the (private) local storage and the (public) cloud storage.

Original:
Table A (PK, c1, c2, c3)
where: c1 is sensitive data column
c2, c3 are non-sensitive data columns

Sliced:
Local.A (PK, c1, link_key)
Cloud.A (link_key, c2, c3)

Figure 1: Data splitting example.

Figure 1 depicts an example of the model's data slicing. The original table A has a primary key (PK) and three columns: c1 containing sensitive data and both c2 and c3 containing non-sensitive data. The sensitive column c1 should be stored in the trusted area (in this case, Local.A table) along with PK, while the non-sensitive columns c2 and c3 can be stored in the untrusted cloud area (in this case, Cloud.A table). These two tables are linked together by a special key (link_key). In the proposed model, PK is not used to link the two partitioned tables since data slicing would be complicated if PK was a combination of many columns.

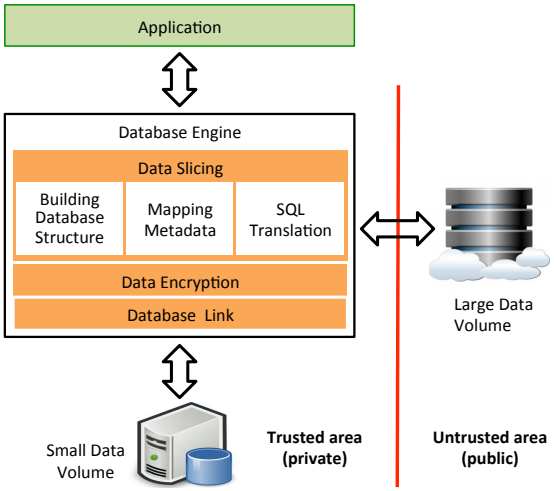


Figure 2: Overall architecture using our data storage model.

3.2 Architecture

Figure 2 shows the overall architecture adopting our secure data storage model on cloud computing environments.

Consider an explanatory application that has several functions to access enterprise data consisting of both sensitive and non-sensitive data. The application runs on either the trusted area or the untrusted area when it guarantees data confidentiality inside of itself. We are concerned with the confidentiality of the sensitive data. In this architecture, it is significant that all the sensitive data is always stored in the trusted area's private data storage. On the other hand, non-sensitive data can be stored in the untrusted area's public data storage.

To attain this secure data storage architecture, the database engine contains three modules: Data Slicing, Data Encryption, and Database Link. All data received from the application is sliced based on the attributes of individual data. The Data Slicing module splits the received data into data in the sensitive columns (labelled with "sensitive") and data in the non-labelled column, and separately passes them to the next modules. The data are encrypted and decrypted if needed by Data Encryption. Database Link is responsible for executing distributed SQL queries.

The Data Slicing module is explained in detail in the next three sections.

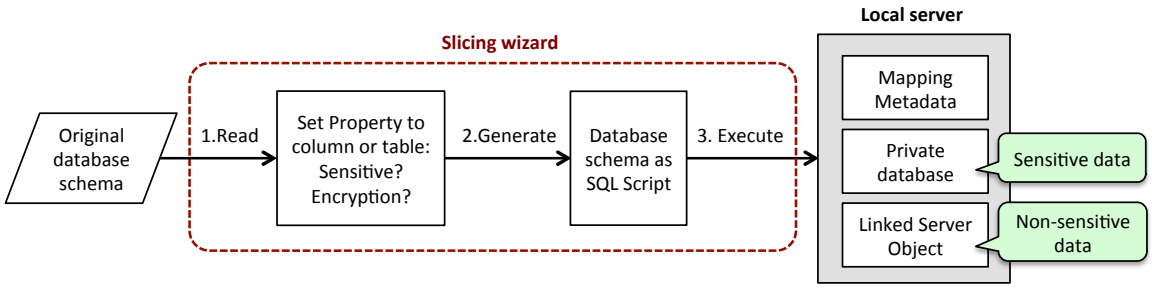


Figure 3: Database schema setup process.

3.2.1 Building Database Structure

Building Database Structure is used to set up the database schema for slicing data. It reads the original database schema and provides a series of steps to set sensitive and encryption properties for columns and tables. It behaves as a setup assistant that instructs database administrators (owners) on how to configure their database properties, as shown in Figure 3.

Based on a database administrator's setting, Building Database Structure generates and executes SQL scripts to build a schema for Mapping Metadata, a private database, and a public database. While generating SQL scripts for the private database schema, Building Database Structure also determines which columns need to be encrypted and encrypts data in those columns using Data Encryption.

The public database is connected to the private database through Linked Server Object on a local server. Distributed queries can be run against the Linked server and the queries can join tables from more than one data source.

3.2.2 Mapping Metadata

Mapping Metadata is a meta-table that contains the mapping information of the partitioned tables, and the sensitive and encrypted column settings. The two last attributes in the meta-table are col_type to distinguish between the link key and other columns and col_order to maintain the column order in the original table, as shown in Figure 4.

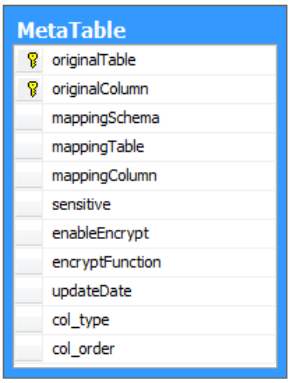


Figure 4: Mapping Metadata for slicing data.

3.2.3 SQL Translation

SQL Translation automatically translates SQL statements from the application into statements that can access data stored in two different storage areas. This means that SQL Translation allows users to access partitioned data transparently. SQL Translation parses the user's SQL statements and gets mapping names in the meta-table. Then, it converts the SQL statement into a new one with a JOIN clause between

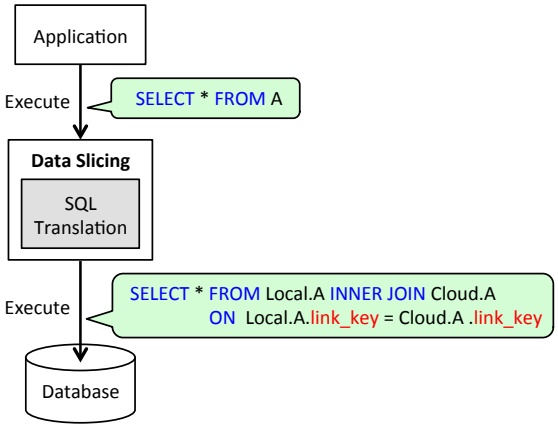


Figure 5: SQL translation example.

two databases through the link key (see Figure 5). Finally, the new SQL statement can be executed to retrieve data from the private and public storages just as the old one can.

SQL Translation also checks the encryption property of all columns and generates SQL scripts to encrypt or decrypt data for the corresponding columns. Based on the generated scripts, Database Engine encrypts and decrypts the data. The popular database servers support several cryptographic algorithms, including data encryption standard (DES), Triple DES, Rivest Cipher (RC2), 128-bit advanced encryption

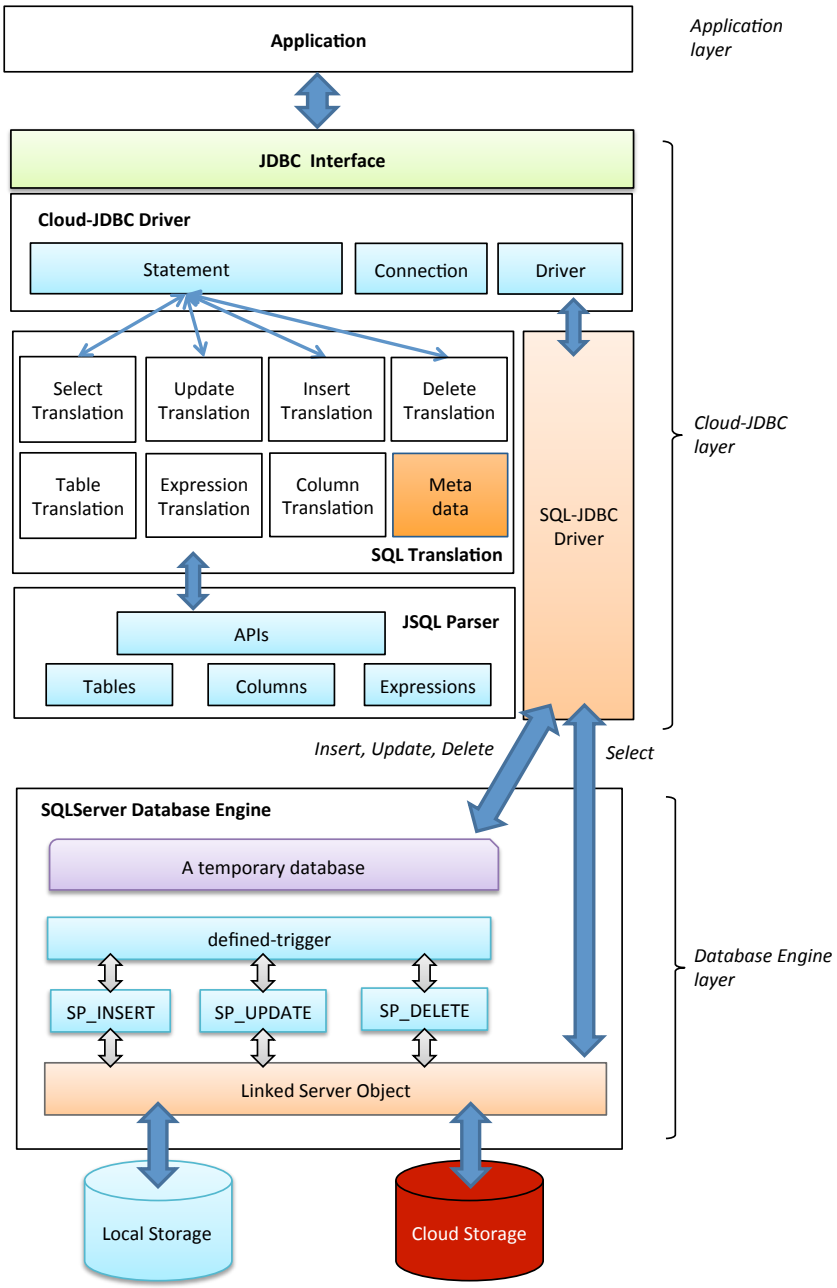


Figure 6: Detail architecture of the proposed model.

standard (AES), and 256-bit AES.

4 IMPLEMENTATION

We describe the detail architecture of the proposed model in this section. As shown in Figure 6, the architecture essentially consists of three layers: the Application layer, the Cloud-JDBC (Java Database Connectivity Interface) layer, and the Database Engine layer.

The Cloud-JDBC layer is composed of a JDBC interface, Cloud-JDBC Driver, SQL Translation, and SQL-JDBC Driver. The Database Engine layer contains several programmable objects (triggers and store procedures) and a Linked Server Object.

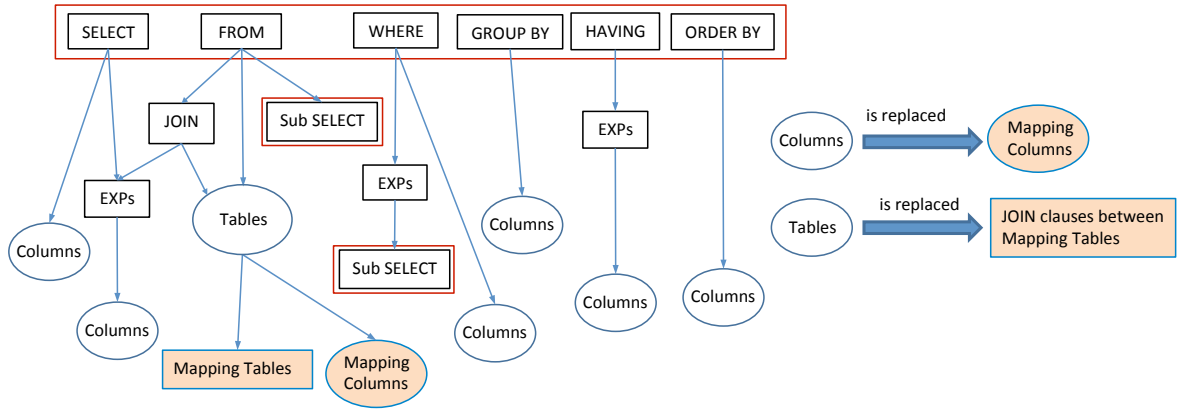


Figure 7: Mechanism of SELECT translation.

4.1 Cloud-JDBC Driver

Cloud-JDBC Driver is extended from the JDBC type 4 Driver to implicitly integrate SQL Translation into migrated applications. This integration permits no changes at the application logic level. This driver connects to the SQL Server in the same way as the JDBC driver. The only difference is the URL of connection that is changed from "`jdbc:sqlserver:`" to "`jdbc:cloudsqlserver:`". Cloud-JDBC Driver is packaged as a Java jar file and added into the migrated application.

4.2 SQL Translation

Java SQL (JSQL) Parser module is a parser of SQL statements, using JSqlParser [1]. This is called by SQL Translation to translate SELECT, UPDATE, INSERT, and DELETE statements. The translations for SELECT and UPDATE are explained in the next two sections. The translations for INSERT and DELETE statements are eliminated here since those are similar to that for the UPDATE statement.

(1) SELECT Statement Translation

The Visitor design pattern is used to translate a SELECT statement. Figure 7 shows the mechanism of SELECT translation. The main syntax of a SELECT statement is as follows.

SELECT *columns* FROM *tables*
WHERE *expressions* GROUP BY *columns*
HAVING *expressions* ORDER BY *columns*

The FROM clause is parsed to gather all data source information (including tables, columns, mapping columns, mapping tables, and aliases) based on Mapping Metadata. We use this data source as a dictionary for translation.

Both replacing the corresponding column names and inserting JOIN clauses between partitioned tables translates a SELECT statement. The SELECT, WHERE, HAVING, and ORDER BY clauses comprise the list of columns and expressions. In fact, each expression is a tree of columns and values. If the tree node is a column, the mapping column in the data source will replace it. Furthermore, the original tables are replaced by JOIN clauses between mapping tables. The sub-SECLECT clause is translated into a call by itself, recursively.

In addition, so that the return set of translated queries continues to have the same result name as the original queries, the tree-part column name (Local.A.c1) is written as an alias, like the original column c1.

(2) UPDATE Statement Translation

A temporary database is used to implicitly translate UPDATE, INSERT, and DELETE statements. This database has the same schema as the original database, except for two columns, link_key and RecordActionType, which are added to each table. The RecordActionType column has three values: U', 'I', and 'D' for update, insertion, and deletion, respectively.

Figure 8 shows the procedure of the translation for an UPDATE statement. SQL Translation translates this UPDATE statement into an INSERT INTO statement. The INSERT INTO statement inserts data into the temporary blank table A from both the Local.A table and the Cloud.A table. The criteria of the INSERT INTO statement is the same as that in the WHERE clause of the UPDATE statement. For the updated values of columns in a SET clause, the auto-generated unique numeric link_key and the RecordAc-

[1] JSqlParser translate SQL statements into a hierarchy of Java classes (`http://jsqlparser.sourceforge.net/`).

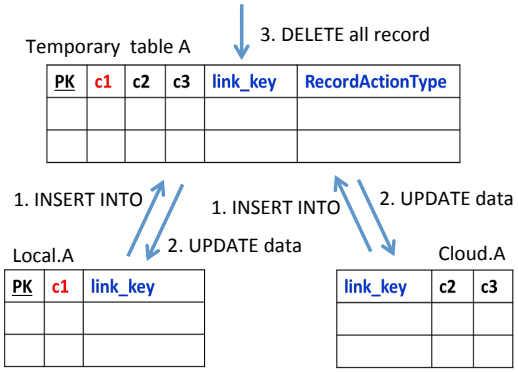


Figure 8: Procedure of UPDATE translation.

tionType having the U' value are also added into the SELECT clause of the INSERT INTO statement.

Running the INSERT INTO statement raises the author-defined-trigger, which is responsible for calling the corresponding stored procedures based on the RecordActionType column's values. In this case, the trigger will call the sp_UPDATE stored procedure to update both the Local.A table and Cloud.A table with data selected from the temporary table A, and delete all data in A.

5 CASE STUDY

To demonstrate the feasibility of the proposed architecture, we developed a prototype tool to implement it. With this tool, we migrated the Plandora [2] to the hybrid cloud environment to check whether it runs correctly. This means that the migrated Plandora returns the same result as the original Plandora when users select, update, insert, and delete some data.

5.1 Setup and Deployment

In this study, we specified information (provider name, server name, user name, password, and database name) to connect to the original database. Next, we chose the places for storages of the private and public data. Then, we created a table with the attributes (identification number, username, password, name, email, phone number, etc.) and stored data of fictitious persons. Finally, we marked the attributes that store sensitive data. The current version of the tool does not support data encryption.

After this setting up, the tool automatically constructs SQL scripts to create a Meta-table, a temporary blank database, local and cloud database schemata, and stored procedures and triggers for INSERT/UPDATE/DELETE statements. Figure 9 shows Mapping Metadata of one of the tables in the Plandora database. The project table is split into two tables saved in two database schemata, Local_Plandora and SERVER_LINK.

We deployed the Plandora to the cloud environment as follows: (1) add the Java jar file of CloudJDBC Driver that includes SQL Translation into the Plandora project; (2) build the database schemata in both the local and cloud storages by running the SQL scripts generated by the tool; and (3) change the connection URL from "`jdbc:sqlserver:`" to "`jdbc:cloudsqlserver:`". In the deployment, there is no change in the implementation code of Plandora.

Figure 10 shows an actual translation by SQL Translation for the SELECT statement in the tool. Access to the project table consists of the accesses to the two tables, Local_Plandora..project (an alias of the local project) and SERVER_LINK...project (an alias of the cloud project). The original SELECT statement retrieves data from four tables. On the other hand, the translated SELECT statement has three INNER JOINs in the FROM clause. Both statements output the same result, indicating that the translation is correct.

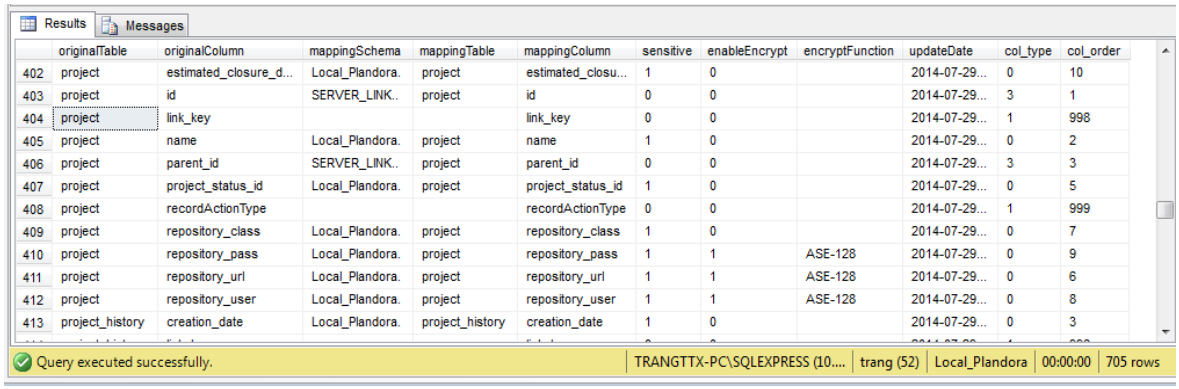
5.2 Experimental Results

Table 1 shows the experimental results for several SQL translations using Plandora. In the experiment, we prepared all SQL patterns from the full syntax of SQL data manipulation language for selecting, updating, inserting, and deleting data in the database. In Table 1, "Pass" represents the number of times the same result was obtained and "Total" represents the total number of trials. "NA" denotes no appearance in Plandora.

The experimental results reveal that the proposed SQL translation mechanism deals appropriately with many of the 20 prepared SQL query patterns. In 14 of the 15 patterns, except for "NA", their "Pass / Total" ratios exceed 95%.

In more detail, several test cases report errors in patterns No. 1 and No. 3. We confirmed that the errors in pattern No. 1 all derive from the inconsistent use of a word. The word "value" is a column name in the Plandora database, but it is a reserved word in JSqlParser. Therefore, JSqlParser failed to parse the "value" column. In pattern No. 3, the SELECT DISTINCT produced a "the text data type cannot be selected as distinct because it is not comparable" error in some cases. This error might result from a difference between database servers. The real Plandora is designed to run in the MySQL database server, while

[2]Plandora is an open source tool to manage the software development process (`http://www.plandora.org/`).



Results | Messages

	originalTable	originalColumn	mappingSchema	mappingTable	mappingColumn	sensitive	enableEncrypt	encryptFunction	updateDate	col_type	col_order
402	project	estimated_closure_d...	Local_Plandora.	project	estimated_closu...	1	0		2014-07-29...	0	10
403	project	id	SERVER_LINK..	project	id	0	0		2014-07-29...	3	1
404	project	link_key			link_key	0	0		2014-07-29...	1	998
405	project	name	Local_Plandora.	project	name	1	0		2014-07-29...	0	2
406	project	parent_id	SERVER_LINK..	project	parent_id	0	0		2014-07-29...	3	3
407	project	project_status_id	Local_Plandora.	project	project_status_id	1	0		2014-07-29...	0	5
408	project	recordActionType			recordActionType	0	0		2014-07-29...	1	999
409	project	repository_class	Local_Plandora.	project	repository_class	1	0		2014-07-29...	0	7
410	project	repository_pass	Local_Plandora.	project	repository_pass	1	1	ASE-128	2014-07-29...	0	9
411	project	repository_url	Local_Plandora.	project	repository_url	1	1	ASE-128	2014-07-29...	0	6
412	project	repository_user	Local_Plandora.	project	repository_user	1	1	ASE-128	2014-07-29...	0	8
413	project_history	creation_date	Local_Plandora.	project_history	creation_date	1	0		2014-07-29...	0	3

Query executed successfully. | TRANGTTX-PC\SQLEXPRESS (10.... | trang (52) | Local_Plandora | 00:00:00 | 705 rows

Figure 9: Mapping Metadata of Plandora database.

```
SELECT p.id, p.name, a.description, a.creation_date,
p.estimated_closure_date, p.project_status_id, p.parent_id,
ps.name AS PROJECT_STATUS_NAME,
p.can_alloc, p.repository_url, p.repository_class,
p.repository_user, p.repository_pass, p.allow_billable,
ps.state_machine_order
FROM project p, planning a, project_status ps, leader e
WHERE p.id = a.id AND ps.id = p.project_status_id AND
(p.can_alloc is null OR p.can_alloc='1')  AND
p.id = e.project_id AND p.id <> '0'
ORDER BY p.name
```

Original SELECT statement

```
SELECT Cloud_project.id,
Local_project.name,
Local_planning.description,
Local_planning.creation_date,
Local_project.estimated_closure_date,
Local_project.project_status_id, Cloud_project.parent_id,
Local_project_status.name AS PROJECT_STATUS_NAME,
Cloud_project.can_alloc, Local_project.repository_url,
Local_project.repository_class, Local_project.repository_user,
Local_project.repository_pass, Cloud_project.allow_billable,
Cloud_project_status.state_machine_order
FROM SERVER_LINK...project AS Cloud_project
INNER JOIN Local_Plandora..project AS Local_project
ON Cloud_project.link_key = Local_project.link_key,
SERVER_LINK...planning AS Cloud_planning
INNER JOIN Local_Plandora..planning AS Local_planning
ON Cloud_planning.link_key = Local_planning.link_key,
Local_Plandora..project_status AS Local_project_status
INNER JOIN SERVER_LINK...project_status AS Cloud_project_status
ON Local_project_status.link_key= Cloud_project_status.link_key,
SERVER_LINK...leader AS Cloud_leader
WHERE Cloud_project.id = Cloud_planning.id AND
Local_project_status.id = Local_project.project_status_id AND
(Cloud_project.can_alloc IS NULL OR
Cloud_project.can_alloc = '1') AND
Cloud_project.id = Cloud_leader.project_id AND
Cloud_project.id <> '0'
ORDER BY Local_project.name ASC
```

Translated SELECT statement

Figure 10: SELECT translation example.

we run it in the Microsoft SQL Server. Unfortunately, several data types converted from MySQL to SQL Server are incompatible.

Table 1: Experimental results for SQL translation.

No.	SQL query patterns	Pass / Total
1	SELECT *select_list* FROM *table_source*	482 / 500
2	SELECT TOP ⋯	NA
3	SELECT DISTINCT	76 / 100
4	SELECT INTO *new_table*	NA
5	WHERE *search_condition*	500 / 500
6	GROUP BY *expression*	78 / 78
7	HAVING *search_condition*	55 / 55
8	ORDER BY *order_expression*	46 / 46
9	OVER *clause*	NA
10	OPTION *clause*	NA
11	WITH *clause*	NA
12	Sub-query	200 / 200
13	SQL Aliases	20 / 20
14	SQL Expressions	70 / 70
15	SQL Operators	50 / 50
16	INSERT INTO *table_or_view* VALUES *clause*	18 / 18
17	SELECT INTO *statement*	25 / 25
18	INSERT INTO SELECT *statement*	200 / 200
19	UPDATE *statement*	50 / 50
20	DELETE *statement*	50 / 50

6 CONCLUSION

In this paper, we proposed an architectural model that guarantees data confidentiality while providing transparent migration of existing applications to cloud environments. Our proposed architecture successfully protects data confidentiality by employing a data slicing technique. Moreover, the running tool uses an SQL translation mechanism to attain the required level of transparency.

Our future work will address several issues. The proposed model only aims the guarantee of data confidentiality. Concerns about data integrity and availability remain as unsolved issues. For example, how

the model protects unauthorized modification of data stored in public storages? How it preserves the consistency of data stored in public and private storages when the failure of the connection to a cloud storage server occurs? These are popular questions but their solutions are not trivial.

With respect to data confidentiality of the model, we are expanding it to other relational database management systems, such as MySQL, PostgreSQL, Oracle, and BD2. We also plan to evaluate the security level and performance of the proposed architecture.

ACKNOWLEDGEMENTS

This work was partially sponsored by the Grant-in-Aid for Scientific Research (15H02685) from the Japan Society for the Promotion of Science (JSPS).

REFERENCES

Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., and Xu, Y. (2005). Two can keep a secret: A distributed architecture for secure database services. In *2nd Biennial Conference on Innovative Data Systems Research (CIDR 2005)*.

Bessani, A., Correia, M., Quaresma, B., Andr'e, F., and Sousa, P. (2011). DEPSKY: Dependable and secure storage in a cloud-of-clouds. In *6th Conference on Computer Systems (EuroSys'11)*, pages 31–46.

Catteddu, D. and Hogben, G. (2009). Cloud computing: Benefits, risks and recommendations for information security. Technical report.

Ciriani, V., di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2011). Selective data outsourcing for enforcing privacy. *Journal of Computer Security*, 19(3):531–566.

Fernandes, D. A., Soares, L. F., ao V. Gomes, J., Freire, M. M., and Inácio, P. R. (2014). Security issues in cloud environments: A survey. *International Journal of Information Security*, 13(2):113–170.

Ferrari, E. (2009). Database as a service: Challenges and solutions for privacy and security. In *Asia-Pacific Services Computing Conference (APSCC 2009)*, pages 46–51.

Ferretti, L., Colajanni, M., Marchetti, M., and Scaruffi, A. E. (2013). Transparent access on encrypted data distributed over multiple cloud infrastructures. In *4th International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 201–207.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pages 169–178.

Gomathisankaran, M., Tyagi, A., and Namuduri, K. (2011). HORNS: A homomorphic encryption scheme for cloud computing using residue number system. In *45th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5.

Hacigümüs, H., Iyer, B., and Mehrotra, S. (2002). Providing database as a service. In *18th International Conference on Data Engineering (ICDE'02)*, pages 29–38.

Hashizume, K., Rosado, D. G., Fernndez-Medina, E., and Fernandez, E. B. (2013). An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):1–13.

Jammalamadaka, R. C., Gamboni, R., Mehrotra, S., Seamons, K. E., and Venkatasubramanian, N. (2008). iDataGuard: Middleware providing a secure network drive interface to untrusted internet data storage. In *11th International Conference on Extending Database Technology (EDBT'08)*, pages 36–41.

Jansen, W. and Grance, T. (2011). Guidelines on security and privacy in public cloud computing. Technical Report SP 800-144.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report SP 800-145.

Menychtas, A., Santzaridou, C., Kousiouris, G., Varvarigou, T., Orue-Echevarria, L., Alonso, J., Gorronogoitia, J., Bruneliere, H., Strauss, O., Senkova, T., Pellens, B., and Stuer, P. (2013). ARTIST methodology and framework: A novel approach for the migration of legacy software on the cloud. In *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 424–431.

Ren, K., Wang, C., and Wang, Q. (2012). Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73.

Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of Networkand Computer Applications*, 34(1):1–11.

Subashini, S. and Kavitha, V. (2012). A metadata based storage model for securing data in cloud environment. *American Journal of Applied Sciences*, 9(9):1407–1414.

Tak, B. C. and Tang, C. (2014). Appcloak: Rapid migration of legacy applications into cloud. In *International Conference on Cloud Computing*, pages 810–817.

Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y., and Vasilakos, A. V. (2014). Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258:371–386.

Weis, J. and Alves-Foss, J. (2011). Securing database as a service: Issues and compromises. *IEEE Security Privacy*, 9(6):49–55.

Yu, A., Sathanur, A. V., and Jandhyala, V. (2012). A partial homomorphic encryption scheme for secure design automation on public clouds. In *21st Conference on Electrical Performance of Electronic Packaging and Systems*, pages 177–180.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.