DD-PLAC: Preserving Privacy with Encrypted Cloud Database

Shital H. Dinde
PG scholar
PVPIT, Pune.

Arati M. Dixit, PhD
Professor
PVPIT, Pune
Dept. of Technology, Savitribai Phule, University of Pune.

ABSTRACT
Cloud database services assure the high availability and scalability, but there are many issues about data privacy and confidentiality. The online applications are also vulnerable to attack that gain access to the sensitive data as the attacker can easily exploit software bug. Hence the security and privacy of the sensitive data stored on the cloud is the biggest challenge today. Storing critical and sensitive data in the hands of cloud service provider will not guarantee the privacy of data. Several ways are available for storage services , but the data privacy and confidentiality solutions for cloud database are still in research. The data privacy and confidentiality can be maintained by combining encryption of data with SQL operations. The application that uses SQL database can be secured by using DD-PLAC architecture which provides confidentiality of the data stored on cloud. DD-PLAC (Distributed Database with Proxy-less architectures that store meta data in the cloud) architecture combines data encryption, key management and access control policies which addresses the issues related to typical threat for cloud database.

Keywords
Security, Privacy, cloud database, Symmetric key algorithm (SKA), access control, DD-PLAC (Distributed Database with Proxy-less architectures that store meta data in the cloud)

1. INTRODUCTION
Cloud computing refers to both the application delivered as services over the internet and the hardware and system software in the data centers that provide those services. Most of the cloud users outsource their sensitive information; security is one of the most major objections to cloud computing. Cloud users face security threats both from outside and inside the cloud. Many of the security concerns involved in protecting clouds from outside threats are similar to those already facing large data centers. Cloud based solutions for database services are now considered as an alternative to scalability and availability. In Cloud computing, sensitive information or data is kept in the hands of untrusted third parties as cloud service providers. The original plain text data must be accessible by trusted and authenticated third parties only, this will not include cloud provider, intermediaries, and Internet. While outsourcing sensitive data to untrusted cloud providers still poses many security and privacy concerns. As the cloud provider is responsible for physical security, like enforcing external firewall policies. Cloud user is responsible for the application level security. So, the goal is to allow users to take advantage of cloud infrastructure and services with data confidentiality by avoiding cloud service providers may access user's data.

The client can access DD-PLAC after the authentication process. After the authentication, a user interacts with the cloud database. DD-PLAC analyzes the original operation to identify which tables are involved and retrieve their meta data from the cloud DBMS. The meta data are decrypted through the unique master key and the information is used to translate the original plain SQL into a query that operates on the encrypted database.

2. RELATED WORK
DD-PLAC provides some features that are 1) allow concurrent SQL operations over encrypted data. 2) Multiple clients can access concurrently and independently a cloud database securely. 3) It does not require intermediate proxy servers. 4) It provides the same availability, scalability as this is compatible with most relational databases. Cryptographic file systems and secure storage will guarantee confidentiality of the data and the integrity of data which will be stored on untrusted cloud. DBMS engines offers encryption of data using Transparent Data Encryption (TDE) [3].It possible to build a trusted DBMS over untrusted storage by using this technique. But, in the DBaaS context the DBMS engine is not trusted because it is controlled by the cloud provider; hence the TDE approach is not suitable for the cloud database services. An approach to preserve data confidentiality in scenarios where the DBMS is not trusted. However it requires a modified DBMS engine that is not compatible with commercial and open source DBMS software adopted by cloud providers. On the other hand, the proposed architecture is compatible with standard DBMS engines, and allows customers to make a secure cloud database by leveraging cloud DBaaS readily available. The architecture given in [4] uses encryption to control access to encrypted data stored in a cloud. This solution is not suitable to usage contexts in which the structure of the database changes, and does not support concurrent access from multiple clients possibly distributed on a geographical scale.

The following are three types of architectures are defined to preserve the privacy.

1. Proxy Based Architecture: The proxy-based architectures[5] do not satisfy our design requirements because the proxy is a bottleneck and a single-point-of-failure that limits availability, scalability and elasticity of the cloud DBaaS. Since the proxy must be trusted, it cannot be outsourced to the cloud and has to be deployed and maintained locally.

2. Proxy-less architectures that store meta data in the clients (PLA):The Proxy-less architecture that store meta data in the clients[4] does not use an intermediate proxy and meta data are stored at the client side. So the clients can connect directly to the cloud database, this architecture provides availability, scalability and elasticity. So, each client has its own encryption engine and manages a local copy of meta data. So, this solution can represent a sub-case of the proxy-based architecture, in which a different proxy is deployed within each client. A similar architecture for cloud accesses would suffer from the same consistency issues of proxy-based architectures.

3. Proxy-less architectures that store meta data in the cloud database (PLAC):The third architecture is proxy-less architectures[6] that store meta data in the cloud database. In this the meta data is stored to the cloud database, but the encryption engine is executed by each client. As meta data are not shared among all the clients there is no need of synchronization. Client machines execute a client software component that allows a user to connect and issue queries directly to the cloud DBaaS. This software component retrieves the necessary meta data from the untrusted database through SQL statements and makes them available to the encryption engine at the client. Multiple clients can access the untrusted cloud database independently, with high availability, scalability and elasticity. The drawback of this architecture is bottleneck and the single point of failure.

3. THE DESIGN OF DD-PLAC

Protecting privacy is very much difficult in a computerized world where individuals, devices, and sensors are associated and information is created, accessed and shared widely with one another. To ensure the client's security, governments additionally came up with legitimate measures, for example, the US federal law called The Secret Data Assurance and Measurable Productivity Act (CIPSEA). In the same endeavors, organizations have utilized different information de-ID routines, for example, pseudonymization, encryption and so on to remove/hide any data that recognizes people. However these de-ID strategies have not been totally ready to secure the client's protection. If anyone wants to store the personal or confidential data in the cloud, these are securely encrypted before storing them to the cloud. Encrypting the data will safeguard the privacy of your data; especially important when you are storing sensitive corporate data or personal information that should never fall into the wrong hands.

A client organization in which a trusted Database Admin machine hosts the DD-PLAC client, which is the application for the creation and management of the encrypted database. All database users can issue SQL operations directly to the cloud database even from geographically distributed locations by executing a DD-PLAC client on their machines. The entire set of data is stored in an encrypted form in the cloud database. Due to the SQL-aware encryption strategies, the cloud database engine can execute queries on encrypted data without accessing any decryption keys. Even meta data that are necessary to manage encryption strategies are considered critical information, hence DD-PLAC stores them encrypted in the cloud database: the Database Admin and the users can efficiently retrieve meta data through standard SQL queries.

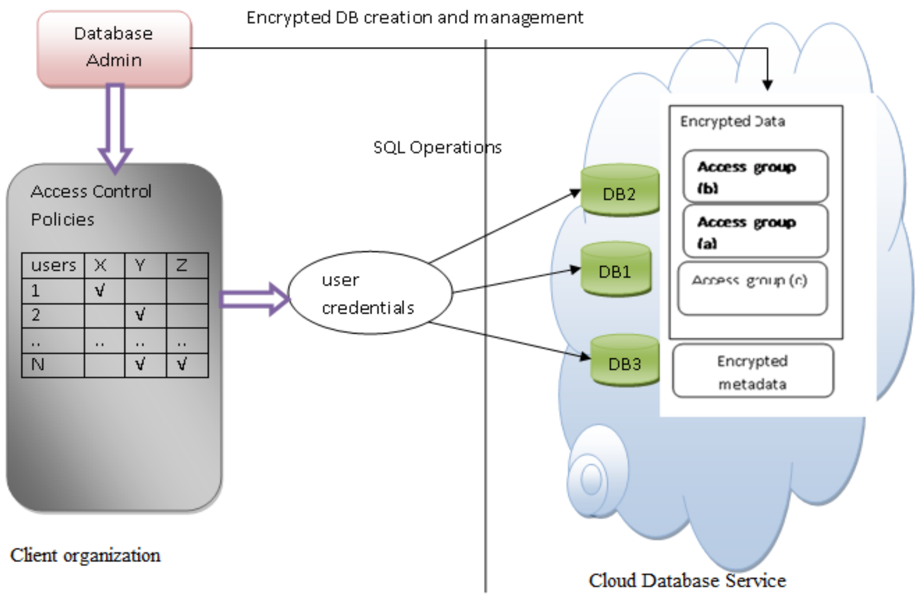


Figure 1: DD-PLAC Architecture

It is assumed that Database admin is the only who has root credentials for the client application and no internal or external attackers able to access, or crack the credentials. Database Admin will manage the user accounts and the access control policies. The access control matrix is mostly used for describing access control policies[15], [16]. Each row in the matrix is a database user and each column is a structure (e.g., column, table, database). Each cell of the access control matrix defines whether a user can access the corresponding data or not. These policies will include set of rules to define which user can access which subset of database. In DD-PLAC the access control is implemented as follows. Each user is provided with a set of user credentials including all information that allows him/her to access all and only the authentic data. Due to this access control policies the data isolation is achieved.

The DD-PLAC architecture shown in Fig.1 is same as proxy-less architectures that store meta data in the cloud database but to provide the more availability and to improve the performance the distributed cloud database is used where the data is distributed over the cloud, which will allow the databases to truly support the elastic requirements of cloud computing applications. Databases have been distributed in terms of instances running on servers that have access to a high-speed network for a while.

In the encryption algorithm, the encryption key is to be designed based on the data to be stored in to the database. Unlike PLAC architecture for every record the different encryption key is being used and that data is encryption key, database name, table name and unique record id is to be stored in again encrypted format in meta data.

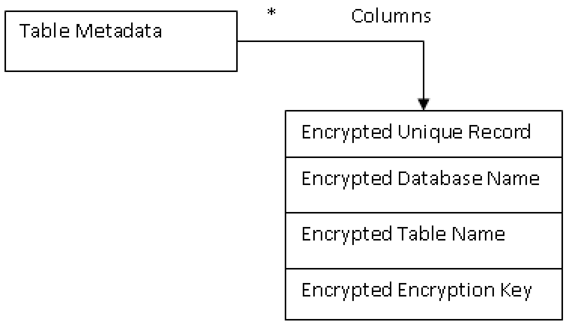


Figure 2: Metadata Structure

The structure of meta data is given in Fig. 2. Using this meta data it is easier to find the encryption key of record being stored in database for the application. Concurrent read and write operations that do not modify the structure of the encrypted database. This encryption algorithm ensures the data privacy and the distributed cloud database will ensures the continuous service from the cloud service provider as well as confirms the data is safe in hands of cloud database.

4. IMPLEMENTATION DETAILS

The implementation of DD-PLAC architecture is done using the case study 'E governance'. The Jelastic public cloud is used to deploy the application and this cloud provides the database storage as Mysql server.

The DD-PLAC architecture is divided into following four parts:

1. Developing application
2. Creating distributed database
3. Implementation of Security algorithm
4. Generating Metadata

1. Developing application:
The web application is developed using PHP, includes the following details of human being:

• Personal Information
• Professional Details
• Banking Details
• Insurance Details

2. Creating distributed database:
The vertical partitioning technique is used to distribute the database over the network.

3. Implementation of Security algorithm:
The AES symmetric key algorithm is used for maintaining security in application. The Encryption key is generated based on the information entered by the user.

4. Generating Metadata:
Metadata stores the Encryption Key, Unique Registration Id in encrypted format.

4.1 Algorithms

K is a random secret generated by the application from the personal details of user at the time of registration. The length of K is 256 bits, as is recommended by most of the standards regarding key length for symmetric key algorithms (SKAs). However, the length of the key can be altered depending on SKA. K is obtained in a two-step process. In the first step, a random number R of length 256 bits is generated such that R = {0, 1}256. In the next step, R is passed through a hash function that could be any hash function with a 256-bit output. In this case, secure hash algorithm 256 (SHA-256) is used. The second step completely randomizes the initial user-derived random number R. The output of the hash function is termed as K and is used in symmetric key encryption [e.g., the Advanced Encryption Standard (AES)] for securing the data.

Encryption Algorithm

Input: Plaintext data P, SKA, 512 bit hash function H

Output: Encrypted data C

Steps:

```
1. Read the Plaintext data(P).
2. Generate the random key(R) from the personal details of user.
   R = {Personal Details}512
3. Generate the symmetric key by applying hash function.
   K = H(R)
4. Encrypt the data by using the key K.
   C = SKA(P, K)
5. Store the C in particular table from the database.
6. Store the key K and random number R in meta data in encrypted format by using the encryption of meta data.
```

For encryption algorithm, input is plain text (like email-id, phone number as well as birth time, birth place and one random string) to generate the encryption key. From the plain text generate some random key R. By using SHA-512, generate the hash values based on input R. The generated 128 bit hash value is used as encryption key for AES algorithm.

Decryption Algorithm

Input: Encrypted data C, SKA, 512 bit hash function H

Output: Plaintext data P

Steps:

```
1. Retrieve the key K from the meta data.
2. Decrypt the stored data by using the key K.
   P=SKA(C,K)
3. Get the original plain text, do the operation on it.
```

In decryption algorithm, first retrieve the key from meta data with respect to particular record. Decrypt the data by using AES and the key and use it.

4.2 Generation of Secret Key
For encryption algorithm, input is plain text (like email-id, phone number as well as birth time, birth place and one random string) to generate the encryption key. From the plain text generate some random key R. By using SHA-512, generate the hash values based on input R. The generated 128 bit hash value is used as encryption key for AES algorithm.

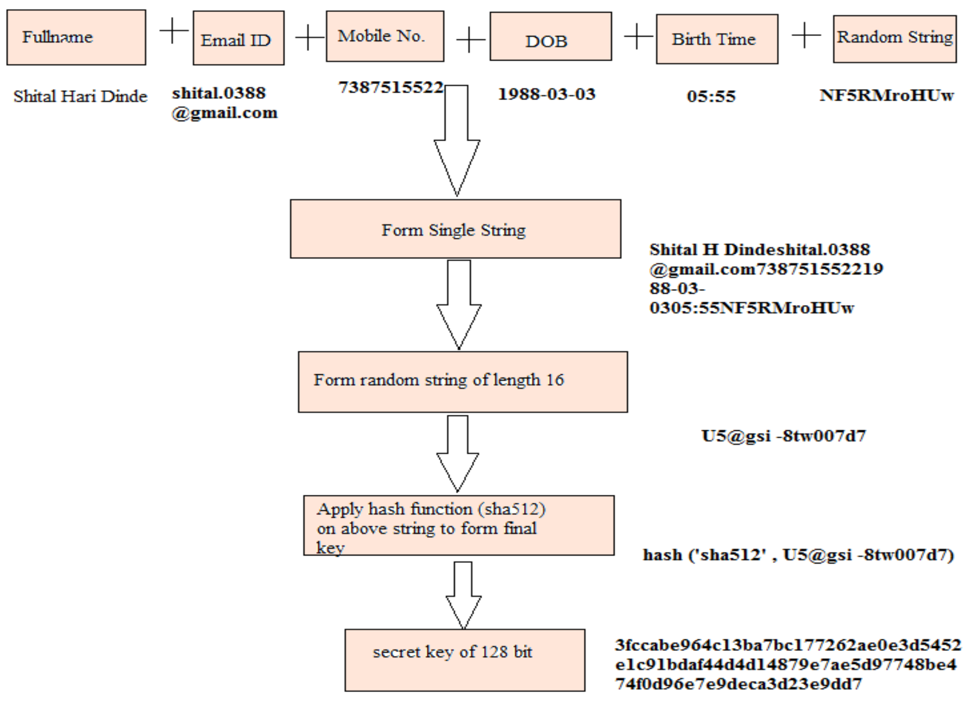

Fullname
Email ID
Mobile No.
DOB
Birth Time
Random String
Shital Hari Dinde
shital.0388 @gmail.com
7387515522
1988-03-03
05:55
NF5RMroHUw
Form Single String
Shital H Dindeshital.0388 @gmail.com738751552219 88-03- 0305:55NF5RMroHUw
Form random string of length 16
U5@gsi -8tw007d7
Apply hash function (sha512) on above string to form final key
hash ('sha512' , U5@gsi -8tw007d7)
secret key of 128 bit
3fccabe964c13ba7bc177262ae0e3d5452 e1c91bdaf44d4d14879e7ae5d97748be4 74f0d96e7e9deca3d23e9dd7

Figure 3: Secret Key Generation

4.3 Queries over encrypted data
DD-PLAC architecture enables the DBMS server to execute SQL queries on encrypted data same as if it were executing the same queries on plain text data. Existing applications do not need to be changed. The DBMS's query plan for an encrypted query is typically the same as for the original query, except that the operators comprising the query, such as selections, projections, joins, aggregates, and orderings, are performed on cipher texts, and use modified operators in some cases. meta data stores a secret master key MK, the database schema.

Processing a query in DD-PLAC given in fig.4 involves three steps:

1. The application issues a query, which the encryption engine intercepts and rewrites it.

2. Then it forwards the encrypted query to the DBMS server, which executes it using standard SQL.

3. The DBMS server returns the (encrypted) query result, which the encryption engine decrypts it and returns to the application.

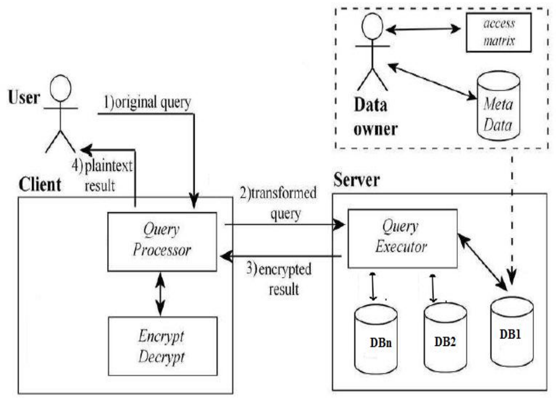
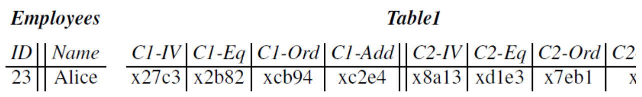


Figure 4: Query Processing

4.3.1 *Read Query*
To understand the query execution over cipher text, consider the following schema given in Fig.5 To illustrate consider the query: SELECT ID FROM Employees WHERE Name = `Alice, The encrypted query is

SELECT C1-Eq, C1-IV FROM Table1 WHERE C2-Eq = x7..d where column C1-IV corresponds to ID, and where x7..d is encryption of "Alice".



Employees

ID	Name
23	Alice

Table1

C1-IV	C1-Eq	C1-Ord	C1-Add	C2-IV	C2-Eq	C2-Ord	C2
x27c3	x2b82	xcb94	xc2e4	x8a13	xd1e3	x7eb1	x

Figure 5: Employee Schema

4.3.2 *Write Query*

For the INSERT, DELETE, and UPDATE queries, the architecture applies the same processing to the predicates (i.e., the WHERE clause) as for read queries. DELETE queries require no additional processing. For all INSERT and UPDATE queries that set the value of a column to a constant, it encrypts each inserted column's value by using the encryption algorithm.

4.4 Distributed Cloud Database

Cloud uses database partitioning for two purposes: 1. If the load of a single machine exceeds its capacity, then scale the database to multiple machines, and (2) for load balance on the back-end machines. To scale these workloads, partition the data such that it minimizes the number of mulch-node transactions and then place the different partitions on different nodes. The aim is to minimize the number of cross-node distributed transactions, which incur overhead both because of the extra work done on each node and because of the increase in the time spent holding locks at the back-ends.[10]

It is obvious that replicating data to an extent will increase the read capacity of the system. However, after a certain replication factor, it might be difficult to maintain consistency even if eager replication and synchronous update processing are used. On the other hand, write capacity can be scaled through partial replication where only subsets of nodes are holding a particular portion of the database. Thus, write operations can be localized and the overheads of concurrent update processing can be reduced. Sharding is a technique to split data into multiple partitions (i.e., Shards). There are two basic ways of partitioning data [13]

1. Vertical Partitioning: by splitting the table attributes (i.e., columns) and thus creating tables with small number of attributes. The main idea is to map different functional areas of an application into different partitions. Both the datasets and workload scalability are driven by different functional aspects of an application. Thus, it is necessary to pick up the right tables and column(s) to create the correct partition, because the 'join' operations in a relational database will now need to be performed within the application code. Vertical partitioning is strictly done for performance reasons.

2. Horizontal Partitioning: by splitting the tuples (i.e., rows) across different tables. It allows scaling into any number of partitions. The tuples are partitioned based on a key which can be hash based, range based or directory based. Join operations are similarly discouraged to avoid cross-partition queries.

For the DD-PLAC architecture the vertical partitioning is used because vertical partitioning is possible in all versions of MySQL and requires no special functionality on the part of the MySQL server and the proper use of vertical partitioning can lead to performance increases of up to 90%. Vertical partitioning lets queries scan less data. This increases query performance. For example, a table that contains seven columns of which only the first four are generally referenced may benefit from splitting the last three columns into a separate table.

5. RESULT ANALYSIS

5.1 Performance Analysis

The performance and the scalability of the DD-PLAC architecture, implemented in PHP, is measured on the cloud database using workloads based on the standard database benchmark TPC-C where concurrent clients are geographically distributed. . The database used for this architecture is MySql Server which supports the main data manipulation (SELECT, INSERT, UPDATE, DELETE) and data definition (CREATE, DELETE) operations of the SQL language with no required modification of the cloud database service, and it can be ported to any relational DBMS and to any commercial cloud database service. The current implementation of the DD-PLAC architecture includes all the encryption algorithms that are necessary to support each SQL operation of the TPC-C workloads on the encrypted database columns. The client load for MySql server is measured by using mysqlslap benchmarking tool which reports the timing of each stage. mysqlslap can emulate a large number of client connections hitting the database server at the same time. The load testing parameters are fully configurable and the results from different test runs can be used to fine-tune database design or hardware resources.

To evaluate encryption costs, the execution time of the SQL commands of the TPC-C benchmark is measured. For each geographically distributed client, average Round Trip Time with respect to the cloud database server (RTT in ms), and the average time required for an AES encryption (ENCT in ms) is measured given in Fig 6. From this figure, the encryption time is below 1 ms for the majority of operations.

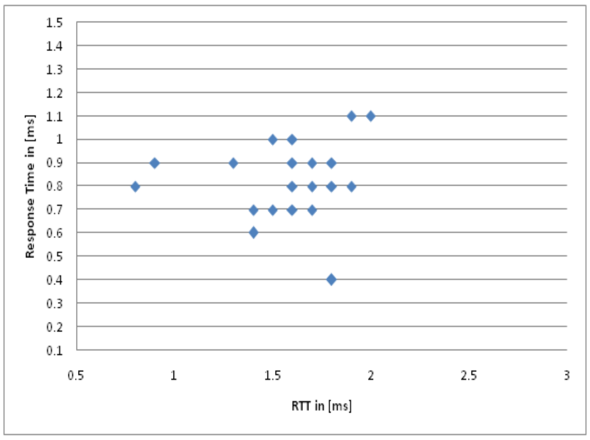


Figure 6: Distribution of the RTTs and ENCT times for the 20 clients

To evaluate the performance overhead of encrypted SQL operations, the focus is on the most frequently executed SELECT, INSERT, UPDATE, and DELETE commands of the TPC-C benchmark. The performance of the DD-PLAC architecture which is evaluated using standard TPC-C benchmark's mysqlslap tool on realistic SQL operations is compared with SecureDBaaS[6] .The graph based on this comparison is plotted in Fig. 7.

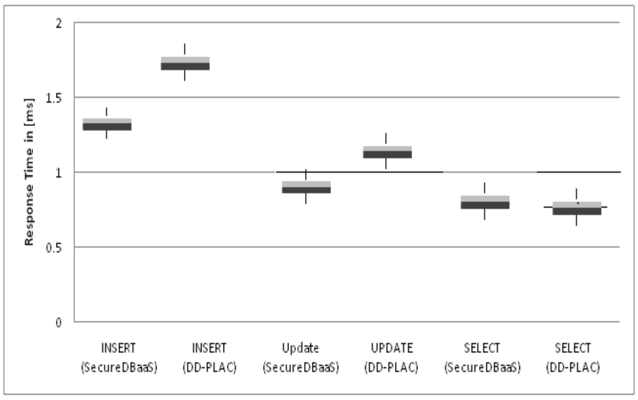


Figure 7: Comparison of DD-PLAC and SecureDBaaS w.r.t Response time

The Y -axis reports the boxplots of the response times expressed in ms (at a different scale), while the X-axis identifies the SQL operations. For SELECT and UPDATE operations, the response times of DD-PLAC SQL commands are slightly greater w.r.t. SecureDBaaS[6], as compared to INSERT operation. INSERT command has to encrypt all columns of a tuple, while an UPDATE operation encrypts just one or few values.

5.3 Comparison of DD-PLAC with Secure DBaaS

Based on analysis done in section 5.1 and 5.2, the conclusion can be done that the performance of DD-PLAC architecture is lower than Secure DBaaS architecture due to the use of distributed database which takes more time to execute the queries over encrypted data.

But the security provided by DD-PLAC is higher than the security provided by SecureDBaaS due the encryption technique used in DD-PLAC

6. CONCLUSION

The DD-PLAC architecture provides a strong level of security and privacy. All the data which is stored on the cloud provider are encrypted through cryptographic algorithms which allow the execution of standard SQL queries on encrypted data. This architecture is also providing direct, independent and concurrent access to the cloud database. It does not rely on a trusted proxy that represents and also avoids the single point of failure and a system bottleneck, which in turn increases the availability and scalability of cloud database services.

7. REFERENCES

[1] Daniel J. Abadi, Data Management in the Cloud: Limitations and Opportunities, IEEE Data Engineering Bulletin, Volume 32, March 2009, 3-12.

[2] James Broberg, Rajkumar Buyya, Zahir Tari, MetaCDN: Harnessing Storage Clouds or high performance content delivery, Journal of Network and Computer Applications, 1012–1022, 2009.

[3] Oracle corporation: Oracle advanced security (October 2012), http://www.oracle.com/technetwork/database/options/advanced-security

[4] Damiani, E., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Metadata Management in Outsourced Encrypted Databases. In: Jonker, W., Petkovi´c, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 16–32. Springer, Heidelberg (2005)

[5] Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty- Third ACM Symposium on Operating Systems Principles, SOSP 2011, pp. 85–100. ACM, New York (2011)

[6] Luca Ferretti, Michele Colajanni, and Mirco Marchetti: Distributed, Concurrent, and Independent Access to Encrypted Cloud Databases. IEEE Transactions On Parallel And Distributed Systems, Vol. 25, No. 2, February 2014

[7] “Xeround: The Cloud Database,” Xeround, http://xeround.com, Apr. 2013.

[8] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M.Newbold, M. Hibler, C. Barb, and A. Joglekar, “An Integrated Experimental Environment for Distributed Systems and Networks,” Proc. Fifth USENIX Conf. Operating Systems Design and Implementation, Dec. 2002.

[9] Tim Mather, Subra Kumaraswamy, and Shahed Latif ”Cloud Security and Privacy” Published by O’Reilly Media, Inc.

[10] Carlo Curion, Evan P.C.Jones, Hari Balkrishna, Nirmesh Malviya, "Relational Cloud: A Database as a Service for the Cloud"

[11] ”Addressing Data Security Challenges in the Cloud” A Trend Micro White Paper | July 2010

[12] tpcc-mysql: Simple usage steps and how to build graphs with gnuplot, by Michael Rikmas. https://www.percona.com/blog/2013/07/01/tpcc-mysql-simple-usage-steps-and-how-to-build-graphs-with-gnuplot/

[13] How To Measure MySQL Query Performance with mysqlslap: https://www.digitalocean.com/community/tutorials/how-to-measure-mysql-query-performance-with-mysqlslap

[14] Manual Reference Pages - \FBMYSQLSLAP\FR (1) http://manpages.sgvulcan.com/mysqlslap.1.php

[15] M. R. Asghar, G. Russello, B. Crispo, and M. Ion, “Supporting complex queries and access policies for multi-user encrypted databases,” in Proc. ACM Workshop Cloud Comput. Secur., Nov.2013, pp. 77–88

[16] R. S. Sandhu and P. Samarati, “Access control: Principle and practice,” IEEE Commun. Mag., vol. 32, no. 9, pp. 40–48, Sep. 1994.