Securing and Authenticating Access to Encrypted Cloud Data

Kunal Kashelani, Varshapriya Jyotinagar

Abstract— Sensitive information, when placed in wrong hands may lead to unwanted results. One way of protecting your data is to never use a cloud based service, but it carries dis-advantages of its own. So, how could it be possible to actually trust the cloud service provider with your sensitive information, without actually giving away the key to access your information?

Sensitive or not, a bulk amount of data gets stored on the cloud and it is also well protected by effective encryption mechanisms. But, the problem does not lie in mere storage, but in accessing and continuously updating of the data. The encrypted data, when gets accessed by the authentic user gets decrypted first and becomes vulnerable to attacks. Such data possesses a chance of being compromised and get placed in the wrong hands. However, if an effective system can be developed, where a user can access, update and perform operations on the data, without actually ever having to decrypt it, it could go a long way sealing the gate of vulnerabilities to such data and protecting key information of the user.

Keywords- cloud, security, confidentiality, database, database as a service.

I. INTRODUCTION

When considering security over cloud, third parties involved such as: cloud provider, intermediaries and internet cannot be considered as trusted. Only authentic users shall be trusted with the sensitive data and no other parties involved. Also the applications available online are vulnerable to theft. Sensitive information can be accessed by the adversaries by exploiting on the software bugs, also there is always the possibility of curious or malicious administrator leaking the private data. So, it can be considered that the privacy of information is a significant problem.

Enough security has been provided when considering the confidentiality of data for storage as a service paradigm, however while ensuring the security of data for database as a service, the paradigm is not considered secured enough.

We plan to design an architecture with a goal to allow multiple, independent clients, not necessarily belonging to the same geographic location, to execute concurrent operations on encrypted data. It should also include SQL statements that modify database structure, which would help in preserving the confidentiality and consistency of data at the client and the cloud level and also would eliminate intermediate servers, if any are present between the client and the cloud provider.

When we try to secure database as a service, our architecture does not introduce any intermediary proxy or a broker server between the cloud provider and the client. This elimination enables the secure database as a service to achieve same availability, elasticity level and reliability of a cloud DBaaS. Any proxy or intermediate server represents a single point of failure and hence were considered impracticable for a cloud based solution.

II. LITERATURE SURVEY

A number of successful attempts have been made in securing the sensitive data of the users. The attempts however may have been significantly different from each other in terms of features provided, they all lack one thing in common, which is providing database as a service. Since the subject was considered too tough to ask for, it was left open for research in pretty much every attempt was made at securing the sensitive information.

Our secure database as a service aims at providing such facility, keeping intact the features of original work, which may be given as:

- It maintains confidentiality of the data by allowing cloud database server to execute concurrent SQL operations over encrypted data.
- The data doesn't compromise on either elasticity, scalability or even availability of it, since it does not require an intermediate server.
- Time constraints required to perform the operation does not increase dramatically.

- *Kunal Kashelani is currently pursuing master's degree (M-Tech) program in Computer Engineering in Veermata Jijabai Technological Institute(VJTI), PH- +919028402049, E-mail: kunal.kashelani@gmail.com.*
- *Varshapriya Jyotinagar currently works as an Associate Professor in department of Computer Engineering, VJTI, Mumbai, India, E-mail: varshapriyajn@vjti.org.in.*

- Geographically distributed clients can concurrently access the database service independently.
- Tenant metadata stored by the cloud database are always encrypted, which allows it to work without the use of trusted proxy.

For a long time, a big problem in using encryption hasn't been if an attacker can crack or not a strongly encrypted file. The problem that persist is that to actually *do* anything with encrypted data—search it, sort it, or perform computation with it–that data must be decrypted and exposed to prying eyes. [7]

A. Crypt-DB:

A fully homomorphic system is the one, in which a user can encrypt data into indecipherable strings of numbers. We can then apply math on those strings to decrypt the result to get the same answer that we would have got had the data not been encrypted at all. Cryptographers have long sought to implement a system like that. Gentry solved that problem of fully homomorphic encryption with a brilliant new system. But his solution was theoretical and had a practical problem: It multiplied the time to perform the calculation by around a trillion. [7]

Crypt-DB tried to create a system that can manage to emulate a fully homomorphic system for as many functions of SQL as possible, where adding a mere 15% to 26% of added computing time to those applications. [2] [5] The only problem that lies with using crypt-DB is that they trust an intermediate proxy for their connection to the database server. Trusting an intermediate proxy is leaving open a single point of failure, which can turn into a system bottle-neck.

They divided their model into two kind of threats: [2] [5]

1. When the DBMS server is compromised, the attacker is supposedly passive and wants to learn about confidential data stored in the database. The threat included root access to DBMS machines, software compromises and even access to RAM of physical machines. CryptDB executes SQL queries over encrypted data to overcome this threat. Secret keys are used by proxy to encrypt all the data inserted or included in queries.
2. The second threat that they considered and meant to solve is arbitrary threat, where the adversary gets access to the keys that are used to encrypt the entire database. In this case, it is supposed that the adversary has gained complete control over the both, software and hardware of the application, DBMS servers and the proxy. CryptDB ensures in this case that the adversary does not gain access to the data of the users which are not currently logged in.

B. Securing DBaaS:

Securing database as a service gives a very good example, where Divyakant Agrawal and his colleagues tried to ensure privacy of data by splitting it between multiple hosts. The idea was such that the hosts won't be able to communicate with one another. The user on the other hand can only communicate with all the hosts, access the data from the respective service providers, combine it and get the original data. It would in such a case be impossible to access the original data from either piece for any malicious user. This method, however remain open to one concern. Metadata (such as field names) are not secured by their software and are open to attacks. This leaves a huge vulnerability in the security system, since Metadata possesses all the necessary information about the data present inside the servers. [4]

The internet over the past decade or so has replaced computer and external hard drives to become the natural place of storing a bulk amount of data, not just for big corporations but also the regular user. With the rise of the stored data over the internet to such a high extent, also considering the fact that the data is mostly private and sensitive, security of the data becomes a natural concern for the users. With the cloud providers providing storage of data in encrypted format may have encouraged us up a bit, but the concern doesn't fade away as the data we store still remains vulnerable to a lot of attacks.

One big concern that most corporations are worried about if the fact that although the data is protected while storage but is susceptible when tried to apply operations to it. Solving such an issue could go a long way maintaining confidentiality of the data considering the importance of protecting sensitive information of the user. Also add to the fact that big corporations such as 'Google' and 'MIT' have already buckled up trying to solve this issue at their own extent makes one understand the importance of such a project. They are calling their project 'Crypt-DB'.

The algorithm that we are planning to develop would be based on AES encryption mechanism and as a result of this, the cloud data on which we are planning to work this algorithm with would be expected to be encrypted with the same encryption mechanism.

C. Commonly used Encryption mechanisms:

Here is a list of some of the well-known corporations and what kind of encryption mechanism they use to store data over the cloud: [8]

- Amazon S3 uses a mechanism called as server-side encryption. They use one of the strongest block ciphers available i.e. 256-bit Advanced Encryption Standard (AES-256), to encrypt your data.
- Google encrypts and stores every cloud storage object's data and metadata under the 128-bit Advanced Encryption Standard, and each encryption

key is itself encrypted with a regularly rotated set of master keys.

- IDrive transfers and stores user's data using 256-bit AES encryption mechanism.
- Wuala uses 2048-bit RSA, 256-bit AES and SHA-256 algorithms for encryption and signatures and to keep integrity checks for your data.
- SpiderOak has a layered approach. It encrypts your data, using a combination of 256-bit AES and 2048-bit RSA.
- CloudSafe transfers your plaintext data using the highest possible SSL standard: EV SSL with AES-256-bit encryption.
- TeamDrive also uses 256-bit AES encryption algorithm to secure your data mainly while transferring or storing it.

These along with various other corporations generally use 128 bit or 256 bit key AES algorithm to encrypt their data and store it in the cloud. Safesync and SwissDisk also use 256 bit AES encryption while Crypto-Heaven uses end to end encryption up to 4096-bit RSA and AES-256 bit algorithm. [8]

It can be seen that most of the highly recognized cloud storage platforms use 128-bit key AES or 256-bit key AES, it can be very well said that our mechanism will be able to secure a very high percentage of cloud database present out there.

III. Design and Architecture

A. Technology:

Java Database Connectivity (JDBC), an API for java defines how a client may access a database. It is oriented towards relational database and works on Java standard edition platform. JDBC provides methods such as querying and updating of data in a cloud database. It is primarily used for making connection of a java program with the databases. J2SE (Java platform standard edition) is a platform, which is widely used for development and deployment of portable applications, basically for desktop and server environment. Being a part of Java software platform family which include (J2ME, J2EE), it includes specifications for Java language and Java virtual machine and also defines a wide range of general purpose API's. [9]

For the applications that are backed by SQL databases, our technique will provide practical and provable confidentiality, for attacks from curious or malicious administrators or where an adversary snoops in and exploits software bugs to gain access to private data. Our technique will work by executing SQL queries over encrypted data using a collection of SQL aware encryption schemes.

Once acquired a cloud database service, a tenant installs the software, which allows them to connect to cloud database as a service to administer it, read, write and modify database tables. A simple assumption made while creating this software is that the tenant is trusted, the network is untrusted while the cloud provider is honest but curious.

Our architecture does not introduce any intermediary proxy or a broker server between the cloud provider and the client.[1] This elimination enables the secure database as a service to achieve the same availability, elasticity level and reliability of a cloud DBaaS. Any proxy or intermediate server represents a single point of failure and hence were considered impracticable for a cloud based solution.

B. Approach and Mechanism:

Transparent data encryption (TDE): It is used to perform real-time encryption/decryption of data. During recovery, a database encryption key (DEK) is used, which is kept stored in database boot record. A certificate is stored in the master database which secures the symmetric key named as DEK. The data and log files are protected by TDE. Software developers uses TDE to encrypt data using AES and 3DES algorithms.

The database will be uploaded on the cloud. When using the DBaaS, the data will only be decrypted when the authenticated user needs access to the data. Certain updating operations will be performed on the database and on a copy of it through two means: once through DBaaS and once through regular way. When the updating is done through DBaaS, user won't be able to see any changes made in the database until he tries to retrieve the data, on the other hand, complete changes in the database may be seen, when the copy of database is updated through usual means. Finally, both the database and the copy of it are retrieved and compared.

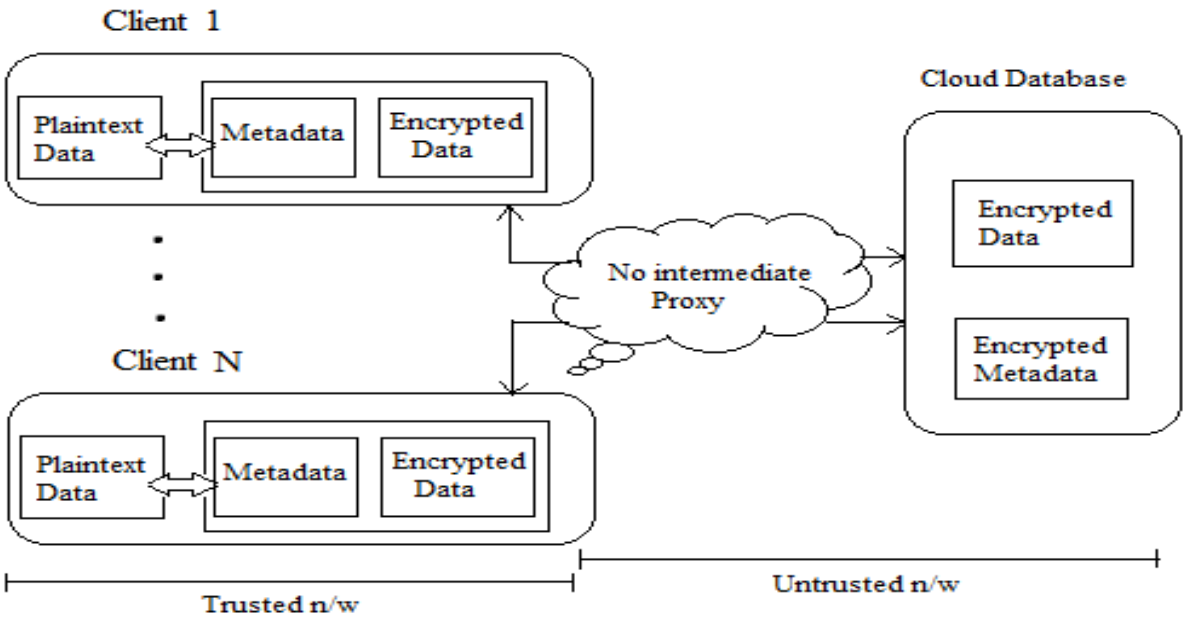

Fig 1: Securing and authenticating encrypted cloud data

If they show the same changes done through the process, the project can be called as a success. The algorithm that we are planning to develop would be based on AES encryption mechanism and as a result of this, the cloud data on which we are planning to work this algorithm with would be expected to be encrypted with the same encryption mechanism.

How-ever, this topic can be considered as a very good topic for the future scope of the project where either some algorithms would work with other well-known encryption mechanisms or possibly even a general purpose software can be built which would discard the dependency of the cloud data on the encryption mechanism and would work with data encrypted in any format.

C. Managing Data and Metadata over the cloud:

Taking a different approach here, our mechanism instead of storing the data over the cloud and metadata over the system of the client, it stores complete data and metadata over the cloud.

The information provided to encrypt and decrypt the plaintext data and other administrative information is called as metadata of the field. Adversaries gaining access to the metadata can use the information to lay a hand on the original plaintext data and hence it is very crucial from security point of view to secure the metadata of the field as well.

Securing database as a service gives a very good example, where Divyakant Agrawal and his colleagues tried to ensure privacy of data by splitting it between multiple hosts. Metadata (such as field names) are however not secured by their software and are open to attacks. This leaves a huge vulnerability in the security system, since Metadata possesses all the necessary information about the data present inside the servers. [4]

We store the plaintext data through secure tables, assuming they are saved in a relational database. While encrypting the tables, we use a same generalized key to store the table names, however a different randomly generated key is used to store each column name to not allow adversary to guess the columns with the same name, since keeping the generalized encryption key for each column would have kept same encrypted name for columns with the same plaintext name, which could have given attackers a certain idea about the relations among the tables.[1] However, in some cases take for example while using join query or the foreign key constraint, it is necessary to encrypt different columns with the same encryption key. This helps in allowing the remote processing of queries over encrypted data.

D. SQL operations:

When trying to run SQL queries over the encrypted data, the information provided by metadata terms really useful. Our architecture instead of leaving the metadata produced vulnerable to attacks, encrypts it and stores it too in the cloud database. One instance of metadata is stored for each database in the cloud and is called as *database metadata* while *table metadata* are the ones storing

information about encryption and decryption of plaintext data and are associated with secure table. Metadata in general contains the encryption keys used to encrypt the secure tables inside the database and hence is really crucial maintaining the confidentiality of the database. Different combination of data types and encryption types have different encryption keys associated with them.

A common master key is used to encrypt both the database metadata and table metadata. Authorized users only are given information about this master key, so they can use it to decrypt metadata also gain the information which will be helpful in encrypting and decrypting the tenant data.

The primary key of the metadata storage table is given to the user in the form of an 'ID' in order for the user to be able to retrieve it. A function named Message authentication code (MAC) is applied to the name of object, which can either be an entire database or a table. If a user knows the plaintext name of a table, it can use the ID to retrieve the metadata of the table.

IV. Conclusion

The proposed architecture, although being developed for just one encryption algorithm, it does not have any theoretical or practical limits attached to it. Our solution can be extended to work on other platforms and also can be included with new encryption algorithms. This topic can be considered as a very good topic for the future scope of the project where either some algorithms would work with other well-known encryption mechanisms or possibly even a general purpose software can be built which would discard the dependency of the cloud data on the encryption mechanism and would work with data encrypted in any format.

The impact of data encryption on the response time would be negligible and hence a very little delay in the performance is found out. This is because it is masked by network latencies, which are typical of cloud scenarios. Further optimizing the performance can also be considered as a possible future scope of this project for interested candidates.

References:

1. Luca Ferretti, Michele Colajanni, and Mirco Marchetti: '*Distributed, Concurrent, and Independent Access to Encrypted Cloud Databases'*, DOI no: 10.1109/TPDS.2013.154.
2. Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan MIT CSAIL: '*CryptDB: Protecting Confidentiality with Encrypted Query Processing*', ACM 978-1-4503-0977.
3. Dan Suciu: '*SQL on Encrypted Database*', ACM 0001-0782/12/09.
4. Joel Weis and Jim Alves-Foss: 'Securing Database as a service', 1540-7993/11/$26.00 © 2011 IEEE.
5. Raluca Ada popa, Catherine M.S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan: 'CryptDB: Processing queries on an encrypted database', 2012 ACM 0001-0782/12/09.
6. Zebra1024. (2014, July 31). Transparent Data Encryption(TDE). Retrieved from: https://msdn.microsoft.com/en-us/library/bb934049.aspx.
7. Andy Greenberg. (2011, December 19). An MIT magic trick: Computing on encrypted database, without ever decrypting it. Retrieved from: http://www.forbes.com/sites/andygreenberg/2011/12/19/an-mit-magic-trick-computing-on-encrypted-databases-without-ever-decrypting-them/.
8. Ashutosh KS. (n.d.). Top 10 online storage solutions with encryption [Blog post]. Retrieved from: http://www.hongkiat.com/blog/online-storage-with-encryption/.
9. Java Connectivity - JDBC (n.d.). Retrieved from: http://www.javadevelopmentindia.com/technology-amp-integration/technology-amp-integration/java-connectivity/jdbc/.