Article can be accessed online at http://www.publishingindia.com

Accessing Data from Cloud Database using Tunneling

Arif Mohammad Abdul*, Sudarson Jena, M. Balraj*****

Abstract

Cloud services are playing an important role in day-to-day life because of rapid development in information technology. Every IT industry is showing interest to adopt the services of cloud. Cloud provides services on demand of users in cheapest price and without maintenance. The users are very much interested to place their sensitive data in cloud but agitate confidentiality cause of internal and external attacks owing to un-trusted public cloud database. In this paper, providing privacy to sensitive data by adding encrypting layer to data and processing requested queries over encrypted data at un-trusted server by using tunneling at client side has been proposed.

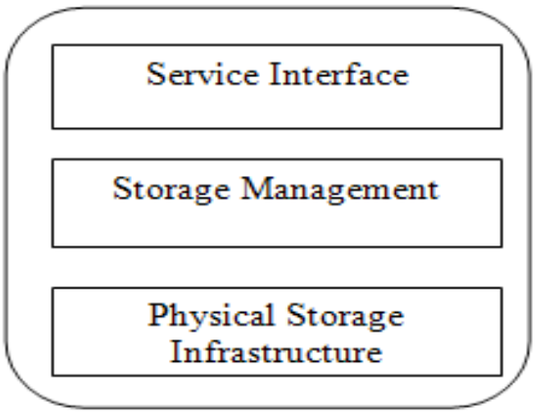
Keywords: Cloud Computing, Trusted System, Encryption Mechanisms, DBMS Server, Proxy Server, Tunneling, Client Site

Introduction

Cloud services are disposed by cloud computing, which includes software, product, infrastructure, shared resources etc. Rapid development in technology gave birth to cloud computing, which provides greater facilities to users like storage. The term "cloud computing" generally refers that applications and data are stored on shared servers. Storage is one of the most essential and demanding computing resource in present digitised era. This type of cloud service is known as cloud storage, every user and IT enterprises are showing interest to use of this service and the service provider provides this service in hired basis in their large-scale storage infrastructure to organisations and individuals. It has always been one of the famous services in cloud computing industry. Cloud storage has distinct characteristics including on-demand self-service, broadband network access, resource multiplexing, rapid elasticity and measured usage for utility billing. Besides the key advantages of cost saving, cloud storage can facilitate information sharing and task collaborating, promote portability and universal accessibility of data, as well as provide easy and convenient solutions to some other problems. For example, for disaster recovery purpose, organisations should maintain secondary off-premise data backups. Storage of sensitive data, such as financial, personal, or medical data is subject to more and more regulations and legal constraints. Cloud storage is offered by low maintenance of data and cost to owners from the complicated process. Cloud storage service named as Infrastructure as a Service (IaaS), deploys the quality of services and manages the storage devices of consumers in the cloud ensuring service level agreements. Amazon's Elastic Compute Cloud (ECC) is a ruling example, with other offerings such as ackspaces Mosso and GoGrid's ServePath (Mell & Grance, 2009). The basic architecture of a cloud storage system is composed of a storage resource pool, including the distributed file system, the Service Level Agreements (SLA), and service interfaces (Tim Jones, 2010; Zeng et al., 2009; Guttman & Roback, 1995).

* Department of Computer Science, GITAM University, Hyderabad, Andhra Pradesh, India. Email: arif@gitam.in
** Department of Information Technology, GITAM University, Hyderabad, Andhra Pradesh, India. Email: sjena2k5@rediffmail.com
*** Department of Computer Science, Avanthi's Scientific Technological and Research Academy, Hyderabad, India. Email: drrajucse@gmail.com

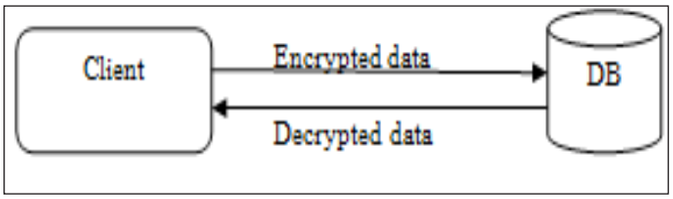
Figure 1: Basic Architecture of a Cloud Storage System



Users are in dilemma to store sensitive data on un-trusted server considering lack of security. Security issues have been reported as the biggest concern preventing enterprises and organisations from adopting cloud services according to recent researches (Chantry, 2009). The public cloud offers great services to users by delivery of computing resources over Internet. It offers scalability, cost effectiveness, increased reliability, and flexibility, as well as fast deployment however they fail to provide privacy, speed, and control. The security requirements for cloud storage differ with different users and applications; they distribute the same three basic objectives as any computer information systems (Guttman & Roback, 1995): confidentiality, integrity and availability. In order to access the data securely private cloud came into existence where the data is stored in the clouds either individual or group of an organisation, the confidentiality issue is better than in public cloud. In public cloud the organisations are much worried about the data leakage. This issue is solved by choosing a private cloud owing the greater level of security to store and/or process sensitive data but cons of private cloud are higher cost and maintenance. So, public cloud is focused on the confidentiality of data stored on servers outside of the company firewall by introducing data storage technique. This technique takes sensitive data as input and replaces it with a unique value called a token. De-tokenisation is the reverse process of replacing a token with its associated clear text value. Con of tokenisation is analysis of data is not done on tokenised data.

Cloud security providers offer ownership of data and some advanced encryption methods to ensure that data in clouds remains secure. Encryption is one strategy by which cloud service providers uses to protect enterprise cloud data from cybercriminals and any unauthorised access. Data encryption process uses an encryption algorithm to transform data. Encrypted values can be transformed back to the original value via the use of a key. It involves conversion of clear text data into cipher text, which cannot be access by unauthorised people.

Figsure 2: Encrypted Database



To secure the data in database cloud service provider has to secure the whole lifecycle of data.

Data in sleep mode: Data need to be secured in sleep mode using some encryption techniques for example AES (Advanced Encryption Standard) and RSA. AES is a symmetric-key algorithm also known as private key, which shares same key for both encryption and decryption of the data. RSA is an asymmetric-key algorithm also known as public key, which is used public key for encryption and private key for decryption.

Data in computation mode: Data should be secured while transferring from receptor to sender or sender to receptor. The techniques are SSL (Secure Socket Layer) and PGP (Pretty Good Privacy). SSL is primary used for web to secure the data while transferring between the web server and browser. PGP is mainly used for the electronic-mail systems.

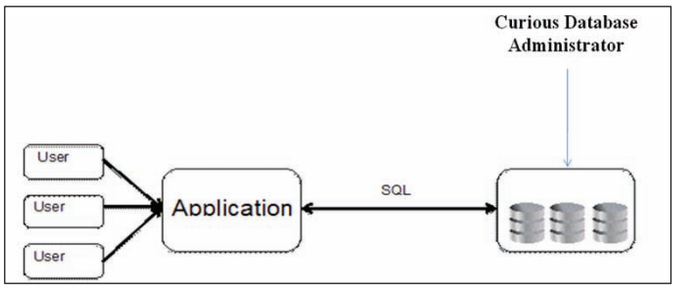
It is observed that one of the biggest problems is how to execute queries efficiently at the cloud resident un-trusted server while maintaining data security as a result of doing operations on data, data converts into plaintext at database then get a chance to unauthorised user can easily gain access.

Un-Trusted Servers

For providing confidentiality cloud offers most secure symmetric and asymmetric algorithms with large key size that is unable to be broken by unauthorised users. Users trust and store their sensitive data without bothering of

confidentiality. If server operator or administrator is compromised then there is no use of data to be encrypted because security level keys are held by administrator for verification of authorised users. Databases are maintained by administrators; obviously users trust the administrators and store their sensitive data in the encrypted format against the face of adversaries. What could happen when the administrators are compromised? Obviously the confidentiality is lost.

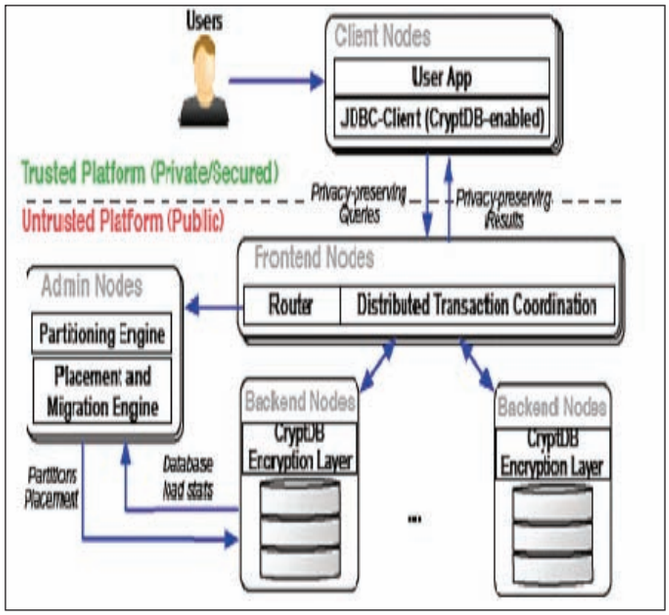
Figure 3: Curious Database Administrator



Database-as-a-Service (DBaaS) is one of the secure cloud service launched by relational cloud. A DBaaS promises to move much of the operational burden of provisioning, configuration, scaling, performance tuning, backup, privacy, and access control from the database users to the service operator, offering lower overall costs to users (Curino *et al*., 2011). Principles and implementation status of DBaaS have been built at MIT (http://relationalcloud.com). This system succeeds in efficient multi-tenancy, elastic scalability but fail to database privacy without cryptdb. Efficient multi-tenancy is achieved by DB-in-VM and elastic scalability is achieved by support of scale-out.

SUNDR (Li *et al*., 2004) is a network file system used to store data securely at un-trusted server by using fork consistency in the face of a malicious server. Fork consistency guarantees the sensitive data of users to be secured at malicious server site owing that security level maintained by client rather than server. The user generates pair of signature keys private and public. Public keys are managed by server at .sundr.users lists with more flexible certificate schemes and private keys are maintained by client security layer. To extract a file user must pass public key as a command to the client, client detects authorised or unauthorised users. If the user is authorised fetch reflects correctly with the property of fetch-modify consistency otherwise it violates fork consistency which means cryptography is in trouble. Formally specified fork consistency (Mazi`eres & Shasha, 2002a), and, assuming digital signatures and a collision-resistant hash function, proven SUNDR's protocol achieves it (Mazi`eres & Shasha, 2002b). SUNDR server consists of two types of programs to handle data – (1) blocks of data and (2) state (small amount of data). Drawback of this protocol is only detects attacks not to resolve, does not yet offers confidentiality to sensitive data.

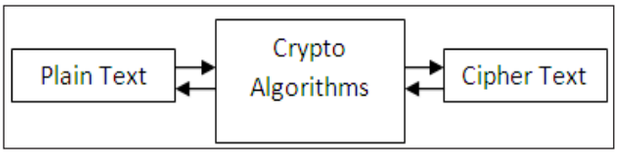
Figure 4: Architecture of DBaaS



Confidentiality can be achieved through encrypted storage, a widely studied problem (Blaze, 1993; Goh *et al*., 2003; Kallahalla *et al*., 2003; Wright *et al*., 2003). Security algorithms (Arora *et al*., 2012), shown in Figure 5, can be categorised into symmetric algorithms and asymmetric algorithms. A key is shared among sender and receptor to encrypt the plaintext into cipher text and decrypt the cipher text into plaintext known as symmetric key or single key or common key or shared key. Symmetric key algorithms are DES (Data Encryption Standard), AES (Advanced Encryption Standard), RC2, Blow-fish, RC4, IDEA; Two-fish algorithms are depends on single key encryptions (Goldreich, 2001; Agrawal *et al.*, 2004). Other algorithms include RSA, Diffie-Hellman, ECC which are based on key management techniques where the receiver's public key is used to encrypt the sensitive text and receiver's private key is used to decrypt the sensitive text. This technique is called as asymmetric key algorithm or two keys or secret key. These entire security prototypes involve generation of keys to encrypt the

sensitive text and decrypt the original message. Cipher text is sent through the network instead original text there is less chance of leaking the original message and other attacks are moderately prevented. Since this encryption revolves around keys for encryption and decryption, in order to decrypt the encrypted message by the receiver he must know the private/secret key which the sender has to send besides the cipher message. Simply one cannot send the key along with cipher text as attackers can easily decode it. Here, key exchange comes in play to exchange these secret keys in unreadable format which the attacker cannot understand (only the sender and receiver can understand). Key exchange algorithms are only meant for exchanging the keys among sender and receiver. Such algorithms are Diffie-Hellman, German Army Enigma, and Key Wrap etc.

Figure 5: Encryption and Decryption Scheme



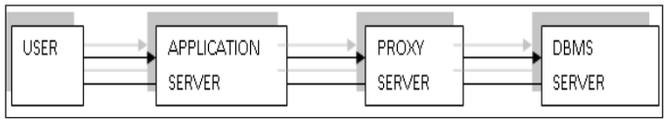
SPORC (Feldman *et al.*, 2010) and Depot (Mahajan *et al.*, 2010) extend SUNDR's design to build applications on top of an encrypted server. For example, SPORC uses operational transforms and fork* consistency to automatic recovery from the errors. SPORC's server concentrates on encrypted data and client server detects any divergence from operation like adding, modifying, dropping, or reordering. If any divergence is found it is recovered. SPORC allows concurrent, low-latency editing of shared state, permits disconnected operation, and supports dynamic access control even in the presence of concurrency (Feldman *et al.*, 2011). It gives confidentiality and security assurance to the user's cryptographic keys neither to compromise administrators nor software attacks and also its application logic is embedded into the client other than runtime.

Depot allows buggy or malicious clients and servers but provides protection and corrects them by using Fork-Join-Causal consistency (FJC). Fork-based techniques (Kallahalla *et al.*, 2003; Wright *et al.*, 2003; Arora *et al.*, 2012; Goldreich, 2001)provide safety in the absence of trust on server, servers may copout if the server is not reachable and clients in block state.

Secure Computation

Every organisation is adopting the services of cloud and getting benefited but remains worried about confidentiality because sensitive data are placed in the cloud, which is accessible to everyone including administrator. Technology has improved and data are protected from outsiders but what if administrator comprises. SUNDR is used to identify the malicious data but does not resolve the problem of attacking adversaries. CryptDB provides confidentiality from administrator by encrypting database and requested queries are executed over encrypted data without decrypting the data at administrator's end. However, many important applications process SQL queries over large quantities of data. If computation data are small then only trusted system is required, but if computed data are large then trusted system is not sufficient, so a trusted server is needed. CryptDB provides a separate trusted server named as Proxy server for computing the operations for large or small scale of data. It also acts as an interface between client and database.

Figure 6: Architecture of CryptDB



Architecture of CryptDB includes application server, proxy server, and database server. Application server receives queries from users and forwards to proxy server. Proxy server has the responsibility to provide the high confidentiality on data by encrypting and executing queries over encrypted data. Proxy server encrypts the query and sends to database server. Proxy server maintains secret bunch of keys called master keys about database schema and encryption keys of all columns of database. Proxy server has the capability to decrypt the data and exhibit the data to database server in the form of anonymized schema and encrypted data. Database server computes the queries without decrypting the data, transfers the encrypted result to proxy server. Proxy server decrypts the result and produces result to user. Plan executor executes encrypted queries same as plaintext. No extra effort is required but some of the operators like Selection, Projection, Joins, Aggregation, and Ordering require user defined functions to execute. This whole process reduces leakage of sensitive information by

executing SQL queries over encrypted data.

To execute queries over encrypted data, CryptDB points three ideas:

1. SQL Aware Encryption Strategy
2. Adjustable Query Based Encryption
3. Chain Encryption Keys to User Password

SQL aware encryption strategy uses the onion encryption technique to provide more confidentiality on the data against administrator and adversary. Onion encryption technique use different algorithms like RND, DET, OPE, HOM, JOIN, SEARCH. These algorithms are used for different operations.

Random (RND) algorithm is a secure algorithm, which provides maximum security. It uses different keys to encrypt similar plaintext, e.g Alice and Alice produce different cipher text even though they are same plaintext because RND uses different keys to encrypt. To get different result RND uses AES in CBC mode or Blowfish in CBC mode with IV. It is used to store information in a secure fashion not for perform any computation or operation.

Deterministic (DET) algorithm provides similar encrypted result for same plaintext still provides strong security and may be leaked because of same encrypted data value. This algorithm is mainly used for equality check.

e.g. select* from student where student_name='ALICE'

Plain Text

Roll_Number	Student_name	Subject	Result
1	ALICE	Networking	Pass
1	ALICE	Database	Fail
1	ALICE	Security	Pass
2	BOB	Networking	Pass

Cipher Text

Roll_Number	Student_name	Subject	Result
Axd1	qwqwer	sdda	fghgf
Axd1	qwqwer	hjkhj	rtuyy
Axd1	qwqwer	guiu	fghgf
TTRY	qwqwes	sdda	fghgf

Order Preserving Encryption (OPE) algorithm is used to maintain the order. It performs computation on encrypted data but is weaker than DET owing that maintaining its order curses this algorithm because it reveals the order.

Homomorphic encryption (HOM) serves some arithmetic operations like addition and multiplications. If execution is performed on encrypted data of two variables to get the result as sum then this algorithm effectuates as the product of two variables in support of paillier technique i.e., HOMK(x) ·HOMK(y) = HOMK(x+y)

Adjustable Query Based Encryption is used to decide on the bases of query type which algorithm is to be used. Onion technique provides strong security over column by column by providing different encryption keys. Proxy server analyzes the computation on a column and decides the suitable onion layer. Onion encryption is done by proxy server and decryption is done by database server. Joining of two columns with different keys is difficult because joining of two tables need same column name but onion encryption uses different keys for encryption. To reduce this mishap, a new technique is launched i.e., JOIN-ADJ. It is performed on two encrypted columns even though the columns encrypted with different keys and one column key is made equal to another column key using hashing technique.

Without using Homomorphic and Join it is difficult to perform addition and order comparisons. It is because different columns encrypt different keys and thus cannot perform addition or order comparison between values from two different columns.

Chain Encryption Keys to User Password issues key for each column as discussed above. Encryption is done column by column by applying different onion layers and is transformed into cipher text without any keys. To provide strong security to these encrypted columns this idea produces keys to each column with user's password.

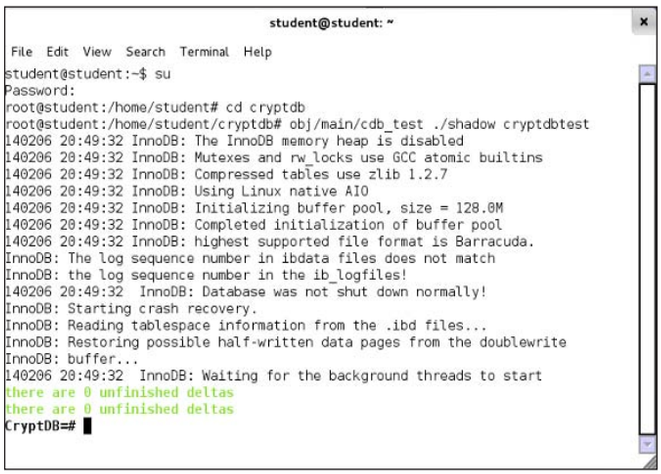
All public or private keys of users are maintained by proxy server but onion layer encryption keys are maintained by database server. Database only has the right to decrypt the onion layer encryption. So proxy does not have chance to enter into database nor does database have a chance to reveal the data.

Execution Process of CryptDB

Query processes method of CryptDB:

1. User issues a query, which is intercepted by intermediate server and re-writes the query in an encrypted form using a 'Key'.
2. The proxy server issues update command instead of issuing keys to call appropriate UDFs from DBMS server.
3. DBMS server receives updated command and adjusts the onion layer based on query demands and executes.
4. Result transfers to intermediate server in an encrypted format.
5. Intermediate server decrypts the result and returns plain text to the user.

Figure 7: CryptDB Initailisation

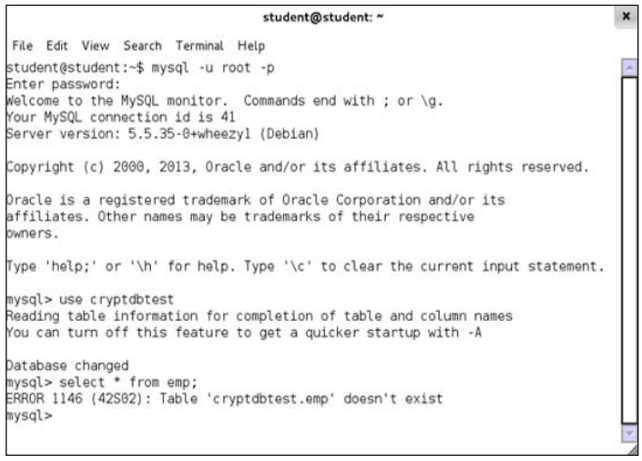


In order to launch the CryptDB, the user needs to login into the system as the root user by using his root password (i.e., by logging in as *sudo* user). The user needs to enter into the directory where the downloaded CryptDB is located, by *cd cryptdb* command. Figure 5 shows the downloaded software is in student directory and the command "*obj/main/cdb_test*" initialises the CryptDB and the second half of command "*./shadow cryptdbtest*" creates a new database in the mysql. Therefore the CryptDB is now ready to setup.

After successful installation of CryptDB we have created 'emp' table and inserted the columns 'eno' of number type and 'ename' of varchar type. When a query such as SELECT * FROM emp; is issued, it returns the encrypted query and equivalent result of that query, as shown in Figure 6. Only CryptDB users are able to visualise the encrypted and decrypted results whereas, the normal Database Administrator (DBA) can visualise only the encrypted data. When he attempts to access the emp table from MYSQL, CryptDB prevents him accessing the decrypted results.

Figure 8: Creating and Selecting a 'Emp' Table from CryptDB

Figure 9: Illegal Access of the Emp Table Data by DBA Using MYSQL



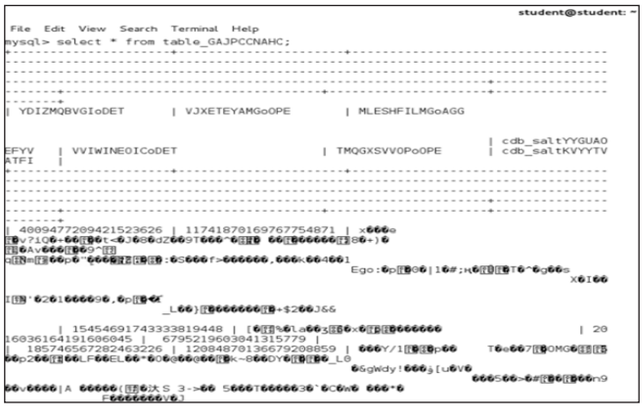
In order to guess the encrypted name of the 'emp' table by DBA, he can call any of the DET (Blaze, 1993) encryption method. Let us suppose that the 'emp' table is encrypted using AES encryption algorithm, one of the DET (deterministic) methods.

Select * from AES (emp);

Which requires a symmetric key , which can be generated through a Key Generator() (Goh *et al*., 2003). One cannot assure that the key generated in CryptDB is same as the key generated by the DBA manually, but there is a chance

of getting same key, by which the DBA can know the encrypted name of 'emp' table (Kallahalla *et al*., 2003). Suppose the DBA or attacker successfully knows the encrypted name of the 'emp' table, and if he issues a query from MYSQL, he encounters only cipher text, as shown in Figure 7, and plaintext is hidden from him. Thus the private data is secured even though the DBA knows the encrypted names.

Figure 10: Results Displayed to the Illegal User

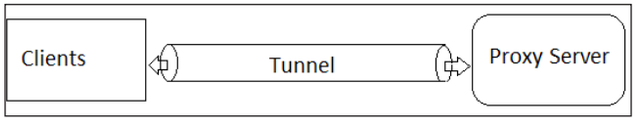


Above results and all the techniques of CryptDB serve confidentiality for those users' data who are logged out but not who are logged in because users execute query and get their results from proxy in plaintext not in encrypted format. Here malicious users and adversaries get a chance to attack and loose the confidentiality.

Tunneling

Using tunneling protocol, user can securely retrieve the plaintext result from intermediate server instead of providing some encryption techniques. This reduces cost for encryption and improves the protection to the plaintext. Tunneling refers to encrypting or wrapping the plaintext at the time of connection at source and decrypting at the destination. Secure movement of sensitive plaintext from one point to another point through a process is called encapsulation. The encapsulation methodology takes into account information parcels to show up as if they are of an open nature to an open system when they are really private information bundles, permitting them to pass unnoticed. It includes repackaging the movement information into an alternate structure, maybe with encryption as standard. A third utilise, or abuse, is to shroud the way of the activity that goes through the tunnel.

Figure 11: CryptDB with Tunneling



To make tunneling in the middle of server, the client must be designed to execute the same tunneling protocol.

Conclusion

CyptDB provides security to the sensitive data when the user logs out but while accessing the data it may lose confidentiality. To improve the confidentiality, tunneling protocol is used in this paper and to identify the attacks, Scyther tool is used with CryptDB in the proposed system.

References

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2004). *Order Preserving Encryption for Numeric Data*. In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data,Paris, France.

Arora, P., Wadhawan, R. C., Ahuja, S. A. (2012). Cloud computing security issues in infrastuture as a service. *International Journal of Advanced Research in Computer Science and Software Engineering*, January, 2(1), 117-125.

Blaze, M. (1993). *A Cryptographic File System for Unix*. In 1st ACM Conference on Communications and Computing Security, (pp. 9-16).

Cachin, C., Keidar., I., & Shraer, A. (2009). *Fail-Aware Untrusted Storage*.

Cachin., C., Shelat., A., & Shraer, A. (2007). *Efficient Fork-linearizable Access to Untrusted Shared Memory*. In Proceedings of 26th ACM Symposium on Principles of Distributed Computing.

Chantry, D. (2009). Mapping applications to the cloud. Technical Report. January 2009.

Curino, C., Jones, E. P. C., Popa, R. A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., & Zeldovich, N. (2011). *Relational Cloud: A Database-as-a-Service for the Cloud*. 5th Biennial Conference on Innovative Data Systems Research, (pp. 9-12), Asilomar, California.

Feldman, A. J., Zeller, W. P., Freedman, M. J., & Felten, E. W. (2010). *SPORC: Group Collaboration using Untrusted Cloud Resources*. In Proceedings of the

9th Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, Canada.

Goh, E. J., Shacham, H., Modadugu, N., & Boneh, D. (2003). *SiRiUS: Securing Remote Untrusted Storage*. In Proceedings of the Tenth Network and Distributed System Security (NDSS) Symposium, (pp. 131-145). Internet Society (ISOC).

Goldreich, O. (2001). *Foundations of Cryptography: Volume I Basic Tools*. Cambridge University Press.

Guttman, B., & Roback, E. A. (1995). Sp 800-12 an introduction to computer security: The NIST Handbook. Technical Report, Gaithersburg, MD, USA, 1995.

Kallahalla, M, Riedel, E., Swaminathan, R., Wang, Q., & Fu, K. (2003). *Plutus : Scalable Secure File Sharing on Untrusted Storage*. In 2nd USENIX Conference on File and Storage Technologies (FAST '03), San Francisco, CA.

Li, J., Krohn, M., Mazieres, D., & Shasha, D. (2004). *Secure Untrusted Data Repository (SUNDR)*. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI), pp. 91-106.

Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., & Walfish, W. (2010). *Depot: Cloud Storage with Minimal Trust*. In Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, Canada, Oct. 2010.

Mazi`eres, D., & Shasha, D. (2002a). *Building Secure File Systems Out of Byzantine Storage*. In Proceedings of the 21st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 108-117.

Mazi`eres, D., & Shasha, D. (2002b). Building secure file systems out of Byzantine storage. Technical Report TR2002–826, NYU Department of Computer Science.

Mell, P., & Grance, T. (2009). The NIST de2nition of cloud computing. Technical Report, July 2009.

Tim Jones, M. (2010). Anatomy of a cloud storage infrastructure. Technical Report, ,IBM, 2010.

Wright, C. P., Martino, M., & Zadok, E. (2003). *NCryptfs: A Secure and Convenient Cryptographic File System*. In Proceedings of the Annual USENIX Technical Conference, (pp. 197-210).

Zeng, W., Zhao, Y., Ou, K., & Song, W. (2009). *Research on Cloud Storage Architecture and Key Technologies*. In Proceedings of International Conference on Interaction Sciences: Information Technology, Culture and Human (pp. 1044-1048).