International Journal of Research
International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

Accessing Confidential Data in Public Clouds

[1]Namala Swathi& [2]Veerendra
India

Abstract—

The user's perception that the confidentiality of their data is endangered by internal and external attacks is limiting the diffusion of public cloud database services. In this context, the use of cryptography is complicated by high computational costs and restrictions on supported SQL operations over encrypted data. In this paper, we propose an architecture that takes advantage of adaptive encryption mechanisms to guarantee at runtime the best level of data confidentiality for any type of SQL operation. We demonstrate through a large set of experiments that these encryption schemes represent a feasible solution for achieving data confidentiality in public cloud databases, even from a performance point of view.

Keywords-Cloud; Database; Confidentiality; Adaptivity; Encryption

I. INTRODUCTION

The Database as a Service (DBaaS) [1] is a novel paradigm through which cloud providers offer the possibility of storing data in remote databases. The main concerns that are preventing the diffusion of DBaaS are related to data security and confidentiality issues [2]. Hence, the main alternative seems the use of cryptography, which is an already adopted solution for files stored in the cloud, but that represents an open issue for database operations over encrypted data. Fully homomorphic encryption theoretically supports any kind of computation over encrypted data [3], but it is computationally unfeasible, because it increases the computational cost of any operation by many orders of magnitude. Other schemes which allow the execution of computations over encrypted data limit the type of allowed operations (e.g., order comparison in [4], sums in [5], search in [6]). Although these methods were successfully deployed in some DBaaS contexts [7], they require the anticipatory choice of which encryption scheme can be used for each database column and for a specific set of SQL commands. In this paper, we propose a cloud database architecture based on adaptive encryption techniques [8] that encapsulate data through different layers of encryption. This adaptive encryption architecture is attractive because it does not require to define at design time which operations are allowed on each column, and because it can guarantee at runtime the maximum level of data confidentiality for different SQL operations. Unfortunately, this scheme is affected by high computationalcosts. However, through a prototype implementation of an encrypted cloud database, we show that adaptive encryption can be well applied to a cloud database paradigm, because most performance overheads are masked by network latencies. This study represents the first performance evaluation of adaptive encryption methods applied to cloud database services. Other experiments [8] assumed a LAN scenario and no network latency. The paper is structured as follows. Section II describes the proposed adaptive encryption scheme for cloud database architectures. Section III presents the results of the experimental evaluations for different network scenarios, workload models and number of clients. Section IV outlines main conclusions and possible directions for improvement.

II. ARCHITECTURE

We describe the architecture we propose to guarantee data confidentiality through adaptive encryption methods in cloud database environments.

A. Architecture model

We refer to the distributed architecture represented where we assume that independent and distributed clients (Client 1 to N) access a public cloud database service [9]. All information (i.e., data and metadata) is stored encrypted in the cloud database. The proposed architecture manages five types of information.

• plain data: the informative content provided by the client users.

• encrypted data: the encrypted data that are stored in the cloud database.

• plain metadata: all the information required by the clients to manage encrypted data on the cloud database.

• encrypted metadata: the encrypted metadata that are stored in the cloud database.

• master key: the encryption key of the encrypted metadata. We assume that it is distributed to all legitimate clients. A legitimate client can issue SQL operations (SELECT, INSERT, UPDATE, DELETE) to the encrypted cloud database by executing the following steps. It retrieves encrypted metadata, and obtains plain metadata by decrypting them through the master key. The metadata are cached locally in a volatile representation that is used for improving performance. Then, the client can issue SQL operations over the encrypted data (i.e., the real informative content), because it is able to encrypt the queries, their parameters, and decrypt their results by using the local plain metadata. This architecture guarantees confidentiality of data in a security model in which the WAN network is untrusted (malicious),while client users are trusted, that is, they do not reveal any informationabout plain data, plain metadata, andthe master key. The cloud provider administrator is semi- honest [10] (also called honest-but-curious), because he could try accessing information stored in the database, but he does not modify internal data and SQL operations results.

B. Adaptive encryption techniques

We consider SQL-aware encryption algorithms that guarantee data confidentiality and allow the cloud database server to carry out a large set of SQL operations over encrypted data.Each algorithm supports a specific subset of SQL operators. This paper refers to the following encryption schemes. Deterministic (Det): it deterministically encrypts data, so that the encryption of an input value always guarantees the same output value. It supports the equality operator. Order Preserving Encryption (OPE) [4]: this encryption scheme preserves in the encrypted values the numerical order of the original unencrypted data. It supports the following SQL operators: equal, unequal, less, less or equal, greater, greater or equal. Sum: this encryption algorithm is homomorphic with respect to the sum operation: summing unencrypted data is equivalent to multiplying the correspondent encrypted values. It supports the sum operator between integer values. Search: it supports equality check on full strings (i.e., the LIKE operator) that do not include fragments of words. Random (Rand): it is a semantic secure encryption (INDCPA) that does not reveal any information of the original plain value. It does not support any SQL operator. Plain: a special kind of “encryption” that leaves values unencrypted. It supports all SQL operators, and it is included to store publicly available data, or some anonymous values that do not require any data confidentiality. If each column data was encrypted through only one of these algorithms, the database administrator would have to decide at the design time which operations must be supported on each database column. This assumption is impractical in most cases. Hence, we need to define adaptive schemes that allow our architecture to support at runtime the SQL operations issued by the clients, while preserving a high level of confidentiality on the columns that are not involved in any operation. For this reason, we organize the encryption schemes into structures called Onions. Each Onion is composed by different encryption algorithms, called (Encryption) Layers, one above the other. Outer Layers guarantee higher data confidentiality and lower number of allowed operations, and each Onion supports a specific set of operators. When additional SQL operations are to be executed on a column, the outer Layers are dynamically decrypted. In this paper, we consider and design the following Onions, which are also represented. it manages the equality operator. Onion-Ord: it manages the following operators: less, less or equal, greater, greater or equal, equal, unequal. Onion-Sum: it manages the sum operator. Onion-Search: it manages the string equality operator. Onion-Single-Layer: a special type of Onion that support only a single Encryption Layer. It is recommended for columns in which operations to be supported are known at design time. In our architecture, each plain database column is encrypted into one or more encrypted columns, each one corresponding to a different Onion, depending on the SQL operations that must be supported on that column. The most external Encryption Layer of an Onion is called Actual Layer, which by default corresponds to its strongest encryption algorithm. Each data type is characterized by a default set of supported Onions, depending on the operations supported by the data type and the compatibility between the encryption algorithms and the data type itself. Each database column can be defined through three parameters: column name, data type, and confidentiality parameters. The confidentiality parameters of a column define the set of Onions to be associated with it, and their starting Actual Layers. The Onions associated to a column must be compatible with the column data type. For example, integer columns can be associated to Onion- Eq, Onion-Ord and Onion-Sum, because integer values support equality checks, order comparisons and sums, but they cannot be associated to Onion-Search, which manages the string equality operator. At the time of a table creation, the database administrator (DBA) has the possibility to specify only a column’s name and data type, as in normal relational Data bases, because our architecture can automatically choose the default set of Onions with regard to the column data type. However, the DBA can also manually specify the confidentiality parameters of a column, when the SQL operations to be supported on the column are known at design time to adapt the level of data confidentiality to the current SQL workload by decrypting an encrypted column’s outer Layer(s). In such a way, it supports at runtime any SQL operation issued by a user. We refer to the Onion adaptation process as the automatic column re-encryption. The proposed architecture is designed so that the column re-encryption is executed on the cloud database through User Defined Functions (i.e., stored procedures) that, when required, are automatically invoked by the clients. Only trusted clients that know decryption keys can invoke column re-encryption. For security reasons, they cannot request any column re-encryption that would expose the Plain Layer of an Onion. Hence, all information stored in the cloud database is always encrypted, and the cloud provider does not have access to plain data. In the re-encryption invocation phase, the client examines the plaintext query issued by the user (which can also be an external application) and evaluates whether the involved SQL operators (e.g., equality checks and order comparisons) are supported with respect to the Actual Layers of the Onions available on the involved columns. If it is necessary to adjust the Actual Layer of one or more Onions in order to support the operators, the client issues a request for re-encryption to the cloud database through a stored procedure invocation. Only trusted clients know the decryption key that is required by the stored procedure to decrypt the outer Layer of the Onion. The invocation phase is repeated for each column that requires re-encryption. In the re-encryption execution phase, the cloud database engine executes a properly defined stored procedure that diminishes the Actual Layer of an Onion by decrypting its row values one by one. After the stored procedure execution, the cloud database sends the information about its outcome (success or failure) to the client that issued the request for re encryption. We observe that any new execution of the same SQL operator on the column does not require to invoke the re-encryption process again, because the cloud database does not encrypt the Onion back to the upper Layer.

C. Discussion

The proposed data confidentiality architecture is inspired by the solutions presented in [8] and [7]. Nevertheless, this is the first that allows to leverage adaptive encryption mechanisms while avoiding the use of any intermediate (trusted) proxy server to manage encryption details. There are several benefits characterizing the proposed architecture. It guarantees confidentiality of information stored in the cloud database, while allowing the execution of SQL operations over encrypted data. It simplifies database configuration, because it does not require to manually define column. It guarantees best level of data confidentiality for any SQL workload, thanks to the automatic column re-encryption mechanism. It does not require any intermediate (trusted) proxy to manage encryption details. We observe that adaptive encryption is also affected by two major drawbacks. The first problem is that each plain column must be encrypted into one or more encrypted columns (Onions), thus increasing the overall database size up to one order of magnitude. This cost may be considered acceptable, or it can be reduced by the database administrator through a suitable tuning of the confidentiality parameters. The second problem is the performance overhead characterizing adaptive encryption,that has to encrypt all parameters and decrypt the results of every SQL operation through all the Encryption Layers of each involved Onion. These costs prevent the use of adaptive encryption methods on most real contexts. However, in Section III we show that this overhead becomes less significant when an encrypted database is used in the cloud, because in these scenarios realistic network latencies tend to mask the CPU time of expensive operations.

III. PERFORMANCE EVALUATION

A. Experimental Testbed

We design a suite of performance tests in order to evaluate the impact of adaptive encryption methods on response times and throughput for different network latencies (from 0 to 120 ms) and number of clients (from 5 to 20). The experiments are carried out in Emulab [11], which provides us with a set of machines in a controlled LAN environment. The TPC-C standard benchmark is used as the workload model for the database services. In Emulab, we design and implement the testbed as a simulated network that connects 20 clients with one server. Each client machine runs the Python client prototype of our architecture on a pc3000 machine having single 3GHz processor, 2GB of RAM and two 10,000 RPM 146GB SCSI disks. The server machine hosts a database server implemented in PostgreSQL 9.1 on a d710 machine having a quad-core Xeon 2.4 GHz processor, 12GB of RAM and a 7,200 RPM 500GB SATA disk. Each machine runs a Fedora 15 image. The current version of the prototype supports the main SQL operations (SELECT, DELETE, INSERT and UPDATE) and the WHERE clause expressions.

IV. CONCLUSION AND FUTURE WORK

We proposed an architecture that supports adaptive data confidentiality in cloud database environments without requiring any intermediate trusted proxy. Adaptive encryption mechanisms have two main benefits: they guarantee at runtime the maximum level of data confidentiality for any SQL workload, and they simplify database configuration at design time. However, they are affected by high computational costs with respect to non adaptive encryption schemes. This paper demonstrated that applying adaptive encryption methods to cloud database services is a suitable solution, because network latency masks the overhead caused by adaptive encryption for most SQL operations. If we consider the overall set of queries belonging to the TPC-C standard benchmark, the overhead becomes negligible for network latencies that are typical of most intra-continental distances, and lower than those of inter-continental distances that often characterize cloud services. Our results also show that the overhead of some SQL operations requiring more encryption steps and more parameters are not masked by Internet latencies. If the workload is characterized by many similar operations, the present alternative is to accept this cost when data confidentiality is more important than performance. As a future solution, we are also studying encryption parallelization solutions that can leverage multithreading over different processor cores.

REFERENCES

[1] H. Hacig¨um¨us¸, B. Iyer, and S. Mehrotra, “Providing database as a service,” in Proc. of the 18th IEEE International Conference on Data Engineering, February 2002, pp. 29–38.

[2] T. Mather, S. Kumaraswamy, and S. Latif, “Cloud security and privacy: an enterprise perspective on risks and compliance”. O’Reilly Media, Incorporated, 2009.

[3] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in Proc. of the 41st annual ACM symposium on Theory of computing, May 2009, pp. 169–178.

[4] A. Boldyreva, N. Chenette, and A. O’Neill, “Order-preserving encryption revisited: Improved security analysis and alternative solutions,” in Proc. of the Advances in Cryptology – CRYPTO 2011. Springer, August 2011, pp. 578–595.

[5] P. Paillier, “Public-key cryptosystems based on composite degree residuosityclasses,” in Proc. of the Advances in Cryptology – EUROCRYPT99. Springer, May 1999, pp. 223–238.

[6] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proc. of the IEEE Symposium on Security and Privacy, May 2000, pp. 44–55.

[7] L. Ferretti, M. Colajanni, and M. Marchetti, "Distributed, concurrent, and independent access to encrypted cloud databases," IEEE Transactions on Parallel and Distributed Systems, vol. 99, no.PrePrints, 2013.

[8] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in Proc. of the 23rd ACM Symposium on Operating Systems Principles, October 2011, pp. 85–100.

[9] L. Ferretti, M. Colajanni, and M. Marchetti, "Supporting security and consistency for cloud database," in Proc. of the 4th International Symposium on Cyberspace Safety and Security. Springer, December 2012, pp. 179–193.