Security and Confidentiality Solutions for Public Cloud Database Services

Luca Ferretti, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti
Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia, Italy
e-mail: {luca.ferretti, fabio.pierazzi, michele.colajanni, mirco.marchetti}@unimore.it

***Abstract*—The users 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 comparions 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 computational costs. 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 in Fig. 1, 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 information about plain data, plain metadata, and

 ISBN: 978-1-61208-298-1

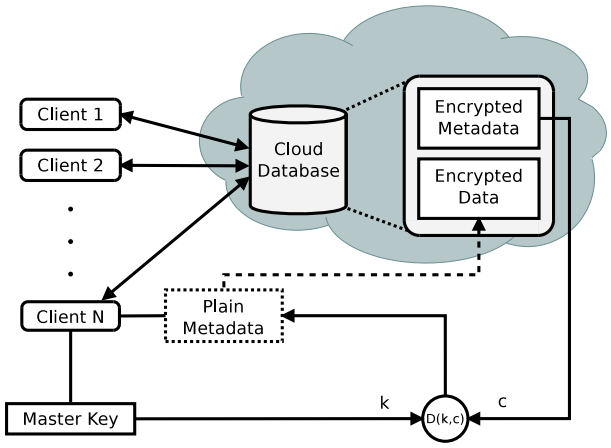


Figure 1. Cloud database architecture

the 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 (IND-CPA) 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 in Fig. 2.

Onion-Eq: 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 supports only a single Encryption Layer. It is recommended for columns in which operations to be supported are known at design time.

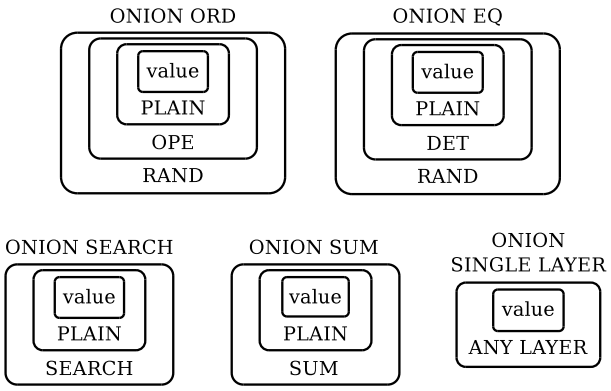


Figure 2. Onions and layers structure

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 databases, because our architecture can automatically choose

 ISBN: 978-1-61208-298-1

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.

Fig. 3 represents an example of the Onions in the structure of the encrypted database table. Each column's Onion corresponds to a different encrypted column. We have two columns: an integer column *id*, with Onion-Eq and Onion-Ord, and a string column *name*, with Onion-Search. We observe that the representation of the encrypted table in this figure is just for clarity, because in the real implementation the table and the columns names should be encrypted too.

Table 1 (Plain)

id	name
1	Mark
3	Luke
4	John

Table 1 (Encrypted)

id-OnionEq	id-OnionOrd	name-OnionSearch
x4cx52	x2bx3xc2	xz53j2hzfap3hx
xd2vsd	xbcv3b3f	x34k2x3243mgj3
x34ds2	x4nj3h3x	x45h23x3cxfhx2

Figure 3. Onions in the encrypted database

The main benefit of the Onions is to allow our architecture 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.

The two main phases involved in the column re-encryption operation are *re-encryption invocation* on the client side and the *re-encryption execution* on the cloud database side. We describe these two phases with reference to Fig. 4.

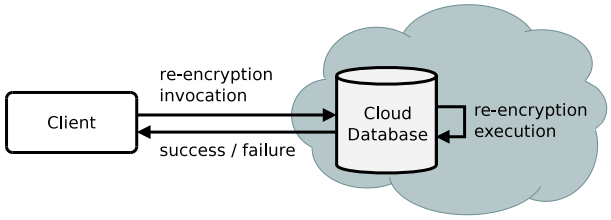


Figure 4. Automatic column re-encryption

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.

For example, let us consider a client that issues an SQL operation involving an equality check on a column that has only Onion-Eq with Actual Layer Rand. The Rand Encryption Layer does not support any operator, and therefore the equality check could not be performed on Rand. In the proposed architecture, a client is able to understand whether the issued query involves an equality check, and therefore it can automatically issue to the cloud database a request to lower down the Actual Layer of Onion-Eq from Rand to Det (re-encryption invocation), which supports the equality operator. After the re-encryption execution, the cloud database sends to the client the information about the outcome of the re-encryption.

We observe that the automatic column re-encryption mechanism is the most critical part of the proposal in terms of deployment and performance. The deployment problem is that the proposed solution requires to install some encryption libraries on the cloud database, in order to allow the stored procedures to be able to decrypt values with the proper encryption schemes (the same used by the clients). Compatibility with possibly any cloud database can be achieved by designing a column re-encryption mechanism that may be disabled anytime. This solution preserves compatibility at the price of losing adaptivity. The performance problem is that re-encryption increases the response time for operations requiring column re-encryption, especially if the column to be re-encrypted consists of a large number of rows. In highly dynamical contexts in which the SQL workload changes frequently, this overhead may be severe.

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

 ISBN: 978-1-61208-298-1

at design time which operations should be allowed on each 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. The prototype integrates the following encryption algorithms (see Section II).

- *Deterministic*: implemented through the standard AES algorithm [12] in CBC mode using a constant initialization value (PyCrypto 2.6 library);
- *Random*: implemented through the standard AES algorithm in CBC mode using a random initialization value (PyCrypto 2.6 library);
- *OPE*: implementation based on the OPE algorithm proposed by [4] and used by [8].

Hence, we implemented the following Onions: Onion-Eq, Onion-Ord and Onion-Single-Layer.

In the PostgreSQL database server, the stored procedures required for column re-encryption (see Section II) are implemented with the *PL/Python* and the *PL/pgSQL* languages. The PyCrypto 2.6 library has been installed in the database server machine in order to support Rand Layer decryption, by using the same functions defined in the client prototype.

In the experiments, we consider three TPC-C compliant databases having ten warehouses and a scale factor of five.

- *PostgreSQL Plain*. This database contains plaintext data.
- *Encrypted*. This configuration refers to an encrypted database where each column is defined as an Onion-Single-Layer. This configuration improves system performance, but does not guarantee adaptive encryption, because each data is encrypted through only one Encryption Layer. Plain, Det, OPE and Rand algorithms are used.
- *Adaptive Encryption*. This database integrates Onion-Eq, Onion-Ord and Onion-Single-Layer.

In the encrypted databases, each column is set to the highest Encryption Layer required to support the SQL operations from the TPC-C workload. In the adaptively encrypted database, we consider the default configuration in which each column is encrypted into all the Onions supported by its data type.

As expected, adaptive encryption schemes increment the overall encrypted database size, but we consider acceptable a larger database size if it guarantees data confidentiality in a cloud environment. In particular, the size of the plaintext database is 490MB, and the size of the adaptively encrypted database is about 4.3GB.

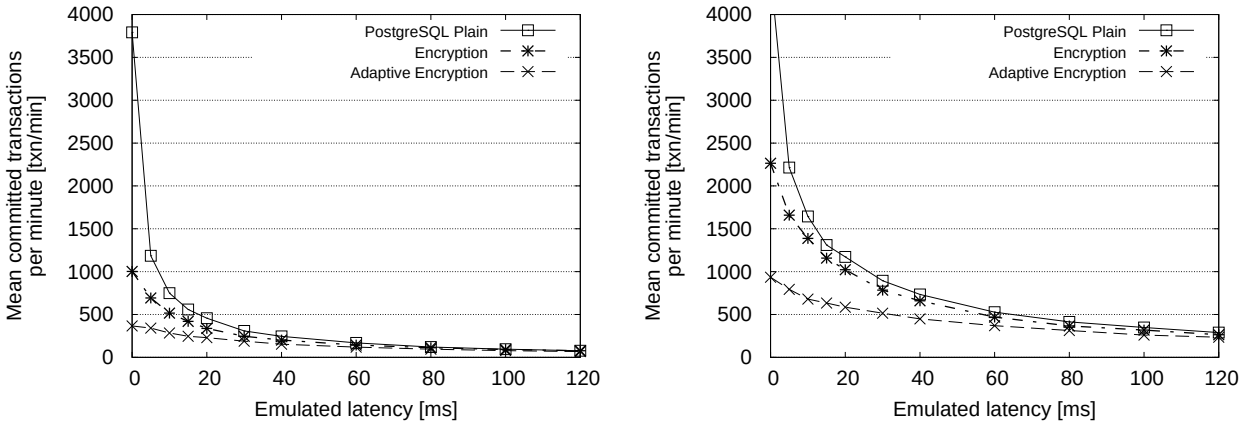
During each TPC-C test lasting for 300 seconds, we monitor the number of executed TPC-C transactions, and the response times of all the SQL operations from the standard TPC-C workload. We repeat the test for each database configuration (plain, encrypted, and adaptively encrypted), for increasing number of clients (from 5 to 20), and for increasing network latencies (from 0 to 120). In order to guarantee data consistency, the three databases have been set to repeatable read (snapshot) isolation level [13]. Since the standard TPC-C workload does not modify the database structure, the experiments consider a situation in which the Onions Actual Layers do not change dynamically at runtime.

B. Experimental results

The experiments aim to evaluate the overhead caused by adaptive encryption and network latencies in terms of system throughput and response time.

In Fig. 5, we report the number of committed TPC-C transactions per minute executed on the three cloud databases. The results refer to 5 and 20 active clients (Fig. 5a and Fig. 5b, respectively) for different network latencies. We can appreciate that in both cases (and in others not reported for space reasons) the throughput of the encrypted database is close to that of the plain database. Moreover, as the network emulated latency increases, even the performance of the database with adaptive encryption tend to that of the other two systems, and it is close to that for latencies higher than 60ms, which are realistic for typical cloud database scenarios. This is an extremely positive

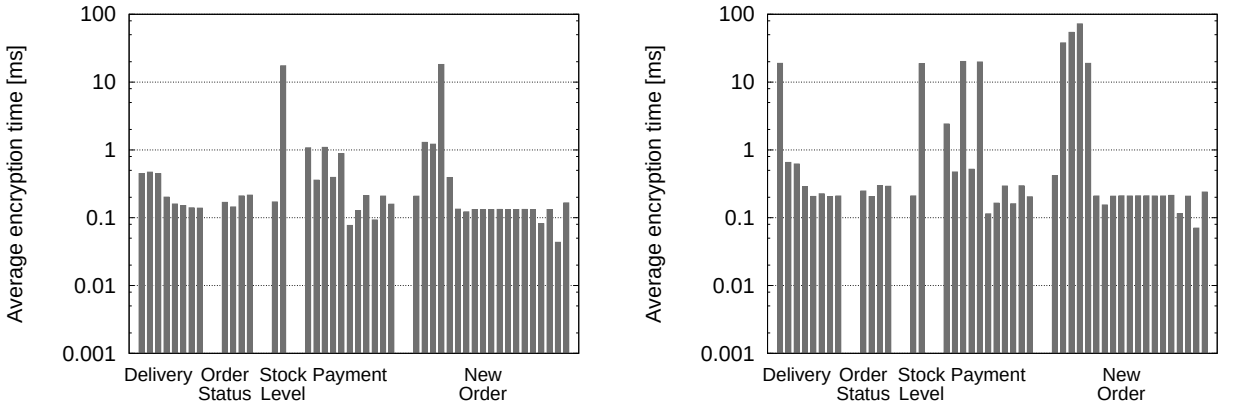
 ISBN: 978-1-61208-298-1



(a) Mean committed txn/m - 5 clients

(b) Mean committed txn/m - 20 clients

Figure 5. Throughput of the three types of cloud database



(a) Non adaptive configuration

(b) Adaptive configuration

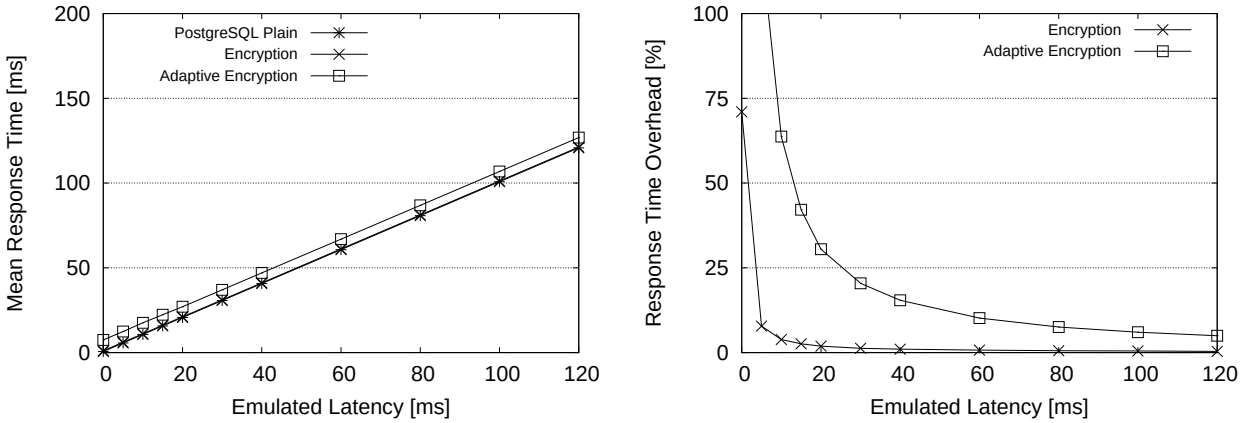
Figure 6. Average encryption times grouped by TPC-C transactions

result, because it demonstrates that adaptive encryption can be realistically used in cloud database services. We observe that the database throughput increases proportionally as a function of the number of clients, because in our experiments the clients do not saturate the database capacity.

In Fig. 6, we report two histograms in which we represent the mean encryption times required by each SQL operation composing the TPC-C workload. These results refer to the mean time that is required by a client to encrypt the parameters involved in each SQL operation of the TPC-C workload. The results are grouped on the basis of classes of TPC-C operations. The histogram in Fig. 6a considers the mean encryption times in the non adaptive configuration, while the other histogram in Fig. 6b considers the adaptive configuration. We can observe how some SQL operations in the adaptive configuration require higher encryption times with respect to those of the non adaptive case. Further analyses show that the two peaks in the non adaptive configuration (Fig. 6a) are related to SQL operations which require OPE encryption, and that most of the additional peaks in the adaptive configuration (Fig. 6b) are related to INSERT operations combined with OPE encryption. The OPE algorithm requires an encryption time that is two or three orders of magnitude higher than that related to Rand and Det algorithms [8]. In the adaptive encryption configuration, if the database administrator does not specify otherwise, every integer column is associated by default with Onion-Eq and Onion-Ord (which has an OPE Layer). Hence, every insertion of a value into an integer column requires an OPE encryption, and this increases significantly the overall encryption overhead. On the other hand, in the non adaptive configuration, the OPE Layer is associated only to the columns in which the database administrator explicitly requires the support for order comparison operations. However, we observe that most of the TPC-C queries encryption times have not been affected by the introduction of adaptive encryption methods (Fig. 6).

In the following set of experiments, we investigate the impact of network latency on the SQL operations response time,

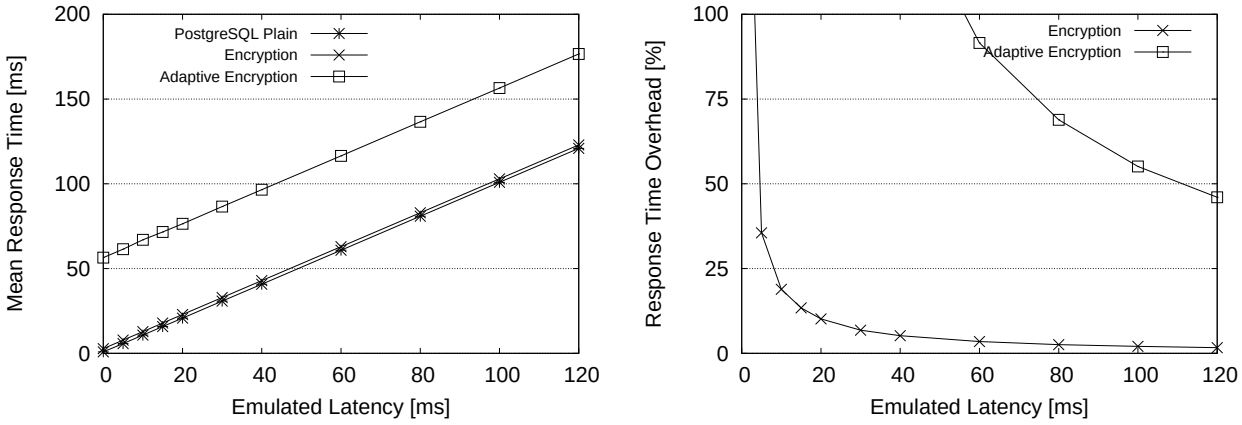
 ISBN: 978-1-61208-298-1



(a) SELECT - 10 clients

(b) SELECT overhead - 10 clients

Figure 7. Response time and overhead of the chosen SELECT operation (10 clients)



(a) INSERT - 10 clients

(b) INSERT overhead - 10 clients

Figure 8. Response time and overhead of the INSERT operation (10 clients)

with regard to the overhead caused by adaptive encryption. For this reason, we evaluate the response time of the most popular SELECT, DELETE, INSERT and UPDATE operations chosen from the TPC-C workload.

In Fig. 7a and Fig. 7b, we report the mean response times and the overhead of the chosen SELECT operation for 10 active clients as a function of increasing network latencies. The overhead of encryption and adaptive encryption is measured with respect to the plain database response time. These two figures confirm that the response time is quite similar for any type of database, and that the overhead of adaptive encryption tends to be masked for latencies higher than 60ms. This is an important conclusion because the results of the chosen SELECT query are also representative of the performance related to the chosen DELETE and UPDATE operations. Completely different results are obtained for the chosen INSERT operation in Fig. 8. In such a case, the adaptively encrypted version of the cloud database has response time and overhead much higher than those of the encrypted and plain databases. While for the encrypted database we can have similar conclusions to those achieved for the other SQL operations, the cost of the chosen INSERT combined with adaptive encryption is very high, and it remains high even for latencies superior to 120ms.

In order to understand the motivation of the overhead of this adaptive INSERT, in Table I, we report the number of encryptions (kE) required to encrypt the parameters of the operations, and the number of decryptions (kD) required to decrypt the results (if any), for each of the two configurations of the cloud database: encrypted and adaptively encrypted. We use the symbol E* to denote the number of OPE encryptions in Table I, because the encryption time required by the OPE algorithm is two or three orders of magnitude higher than that related to Rand and Det algorithms [8]. From this table, we can see that the high overhead of the INSERT in the adaptively encrypted configuration is caused by the higher number of encryptions required by the adaptive architecture to encrypt all parameters of the different Onions, and by the very high computational cost characterizing the OPE encryption.

 ISBN: 978-1-61208-298-1

TABLE I. NUMBERS OF ENCRYPTIONS (kE) AND DECRYPTIONS (kD) INVOLVED IN THE CHOSEN SELECT AND INSERT OPERATIONS FOR THE TWO ENCRYPTED DATABASE CONFIGURATIONS

Query	Layers	Non adaptive configuration	Adaptive configuration
SELECT	DET	2E	2E + 3D
	OPE	none	none
	RAND	3D	3D
INSERT	DET	2E	6E
	OPE	none	3E*
	RAND	7E	7E

C. Main conclusions of the performance study

All experimental results show that network latencies higher than 60 ms, which are typical of most cloud database environments, make the adaptive encryption overhead almost negligible when considering the overall set of operations of the TPC-C standard benchmark. However, in the adaptively encrypted database configuration, for some SQL operations involving the OPE encryption or for the encryption of a high number of parameters through several Encryption Layers (e.g., INSERT), the impact on the response time is visible even for network latencies higher than 120 ms. We can conclude that the proposed approach must be improved for contexts characterized by a large number of these operations. We are working on parallelized encryption schemes through multithreading over different cores.

If we refer to a scenario for increasing numbers of clients, we can observe that the (adaptive) encryption overheads (e.g., Fig. 7b and Fig. 8b) remain constant, while the cloud database throughput (e.g., Fig. 5a and Fig. 5b) increases proportionally as a function of the number of active clients.

It is worth to observe that the adaptively encrypted configuration represents a *worst case scenario*, in which each database column is encrypted into all the Onions supported by its data type (each Onion is assumed at its highest Encryption Layer). This is a scenario in which no user manually configure any encryption detail, and our architecture automatically chooses for each column all the supported Onions in order to guarantee adaptivity. This completely adaptive configuration requires a high number of encryptions and decryptions per query that may affect system performance especially for some SQL operations. On the other hand, the non adaptively encrypted configuration represents a *best case scenario*, in which the user manually defines which encryption scheme to use on each database column (see Section II). This configuration guarantees best performance, but no adaptivity. As a consequence, we can claim that realistic workloads are characterized by performance results falling between the two extreme scenarios presented in this paper.

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. Hacigümüş, 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 residuosity classes,” 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.

[10] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications. Cambridge university press, 2004.

[11] 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,” in Proc. of the 5th USENIX Conference on Operating Systems Design and Implementation, December 2002, pp. 255–270.

[12] J. Daemen and V. Rijmen, The design of Rijndael: AES – the advanced encryption standard. Springer, 2002.

[13] A. Fekete, D. Liarokapis, E. O’Neil, P. O’Neil, and D. Shasha, “Making snapshot isolation serializable,” ACM Transactions on Database Systems, vol. 30, no. 2, June 2005, pp. 492–528.

 ISBN: 978-1-61208-298-1