Secure Data Storage in Distributed Cloud Environments

Renata Jordão, Valério Aymoré Martins, Fábio Buiati, Rafael Timóteo de Sousa Júnior, Flávio Elias de Deus
Electrical Engineering Department, University of Brasilia,
Campus Universitário Darcy Ribeiro, Asa Norte, 70910-900 Brasilia, DF, Brazil
e-mail: {renata.jordao, fabio.buiati } @redes.unb.br
{valeriomartins, desousa, flavioelias} @unb.br

***Abstract*— In order to prevent the attack and discovery of sensitive data, a model is presented to protect the confidentiality of data even when the attackers have access to all data from the server: using systems that perform queries and inserts on encrypted data without the decryption key. The model set is applied in a real scenario: a distributed system hosted by Google (Encrypted BigQuery), which includes the use of encryption in data stored in non-relational databases for massive data processing. As a result, it is shown that this system supports a variety of applications with low overhead.**

Keywords - confidenciality; cloud; Encrypted BigQuery; big data, NoSQL.

I. INTRODUCTION

The concern with the design and maintenance of safe environments became the main occupation of network administrators and developers, wherein the majority of reported attacks, information theft and unauthorized access, are done directly on data source [14]. For instance, next to their databases, or by persons belonging to the organization, or the lack of mechanisms to prevent direct remote reading of databases by third parties.

Considering these security issues, this work proposes the implementation of a secure data storage model in distributed cloud environments where there is risk of direct access to the data by other unauthorized access (e.g. Cloud). The main goal of this work is to guarantee confidentiality and integrity to data hosted in cloud databases systems with highly sensitive content through cryptographic adjustable models applied in fields of tables that allow queries operations over encrypted data.

We evaluate the proposed model using a real scenario: a distributed system hosted by Google (Encrypted BigQuery[1]) that allows developers to rewrite the queries in order to access the encrypted data directly without allowing the server to expose their real content.

This paper is structured as follows. In section II, we review the main concepts, including massive databases and technologies related to Hadoop and MapReduce as well as security in the storage and data transmission in cloud environments. In section III, we describe the EBQ tool (Encrypted BigQuery) as a system for conducting SQL-like queries over an encrypted database with the set of encryption schemes used in the editable queries. Then we detail the proposed model, the implementation rolled out and the results obtained in section IV. The final section presents the conclusions and points out future work.

[1] https://code.google.com/p/encrypted-bigquery-client/

II. MASSIVE DATA STORAGE AND PROCESSING

A. Massive Data Storage based on NoSQL

The large volume of data generated by Web applications along with the different requirements of their applications, such as scalability on demand and the high degree of availability, have contributed to the emerging of new paradigms and technologies. In this context a new category of database called NoSQL (Not Only SQL) [3], was proposed in order to meet the requirements of managing large volume of data, either semi-structured or unstructured. Proposing the break of ACID paradigm [3], NoSQL storage models present a systematic, which groups them into four basic types, namely:

- Key-Value Stores: A simple model that allows database visualization as a large hash table. E.g: Dynamo DB (Amazon), Voldemort, Berkeley DB, Redis and Scalaris.
- Column-oriented (BigTable style databases): databases like Google BigTable model, that priorize data partitioning with strong consistency. E.g: Cassandra, Google BigTable and Hyperbase.
- Document-oriented (Document databases): It holds collections of documents, i.e., objects with unique identifiers and a set of fields (strings, lists or nested documents), similar to the key-value model, but each document has a set of field-keys and the values of these fields. E.g: CouchDB and MongoDB.
- Graphs-oriented (Graph databases): It features three basic components (nodes, relationships, properties) allowing the database management system (DBMS) to be seen as a labeled directed multigraph, in which each pair of nodes can be connected by more than one edge. E.g: Neo4j, the InfoGrid and HyperGraphDB.

B. Massive Data Processing

Hadoop was developed by the Apache Software Foundation to run in a large number of machines that do not share memory or disks. The Hadoop Distributed File System (HDFS) is its distributed file system. The system divides the data in blocks, which are propagating through different servers. Due to the large number of repositories, data stored on a server, which may disconnect or die, can be recovered from a copy from the others servers. The reason why the system can answer computationally complicated questions is that it has all these processors, working in parallel. The mechanism responsible for this parallel computing is MapReduce, which is a computational framework for parallel processing created by Google. It abstracts the difficulties of working with distributed data, so that each node is independent and it consists the following functions [5]:

• Map: it takes a list as input, applies a function to process the distributed file system in parallel, and generates as output a new list usually other key/value pairs.

• Shuffle: it is responsible for arranging the output of the Map function assigning to each Reduce input all the values associated with the same key.

• Reduce: it receives the result of Map function as input, applies a function reducing the input to a single value in the output.

The set of requirements defined for this work - a model for massive distributed partition-tolerant storage, with direct research on restricted sets of information – column-oriented distributed storage with typical Map/Reduce processes present themselves as better-proposed models of column-oriented NoSQL storages that are target of this work, especially the implementation of Google: the BigQuery/Dremel.

C. BigQuery/Dremel

BigQuery [2] is the external implementation of Dremel tool, used internally by Google for the query service on large data sets. It came up from the need for a tool that could handle queries and return results faster due to the amount of data that Google has to deal with daily.

The performance advantage of BigQuery comes from its parallel processing architecture. Thousands of servers in a multi-level tree structure, aggregating the results at the root, compute the SQL-like query. The BigQuery stores the data in a column format, so that only the data of the columns being queried are read. This configuration provides significant performance gains in I/O compared to the traditional model of relational database/data storage since there is no need to read the entire record for each query. In this way, the combination of a column-storage with a tree structure gives Google BigQuery its performance gain.

[2] https://developers.google.com/bigquery/

D. Security

To design a secure system, one can follow the security definitions of Stallings [12], taking in account the principles of availability, integrity, confidentiality, authenticity and nonrepudiation.

This work focuses on addressing the issues of confidentiality regarding to the content of the message [7], and issues of authenticity, where only persons, entities and authorized processes, can access data stored and/or transmitted [12].

To this purpose, a basic service provided by encryption techniques has the ability to send information between two participants in a manner that prevents others from reading it. Using a key, a plaintext is encrypted by an encryption algorithm resulting in a ciphertext. It is important that an encryption algorithm is reasonably efficient so that it can be calculated. Encryption schemes are not impossible to be broken without the key. The security level of a scheme depends on how much effort or resources are allocated by the attacker to break it [8].

Within the context proposed for this work, the storage and data transmission should occur in environments where data can be provisioned in an infrastructure with low security requirements, such as cloud computing. The access is made directly by trusted clients interface or proxies' services containing specific trust mechanisms between clients and proxies. One possible interesting solution is to encrypt data from the client before host them on some storage service, so that only the client has knowledge of the content hosted and the querying also does not leak information.

Thus, the relevant information can be encrypted before being sent to the servers for processing and analysis. However, to process queries on the encrypted data is difficult. One approach is to design a scheme that can perform arbitrary functions on encrypted data as whether it was running on the plaintext. A fully homomorphic encryption [4] is a theoretical concept that meets these goals in ensuring confidentiality, but it is considered impractical at the time.

III. SECURE DATA STORAGE MODEL

Several approaches propose security mechanisms to deal with the problem in data storage. Song et al. [11] describe cryptographic algorithms to perform a search for keywords on encrypted data using an insecure server. Boneh & Waters [2] present a model with public-key schemes for comparison, checking subsets of conjunction and querying on encrypted data, but these schemes have exponential cipher size in relation to the plaintext, restraining its practical application.

Regarding existing implementations, CryptDB [10] provides confidentiality based on SQL database applications using a proxy to intercept the coming customer queries,

encrypting all communication. Arasu et al. [1] present another implementation: the Cipherbase, a complete SQL database system that allows organizations to use the benefits of cloud computing while maintaining confidentiality of sensitive data.

This paper employs the implementation of Google BigQuery, secure column-oriented storage model. Google provides this mechanism through an encrypted access interface, Encrypted BigQuery (EBQ). EBQ is an interface that allows encryption of data from the client and then uploads to BigQuery [13]. Thus, in theory, there is no possibility of any BigQuery administrator gaining access to data. The EBQ operation begins with the construction of the schema file indicating the field's structure at the table to be loaded. EBQ includes an extra field (encryption method), which sets the types of cryptographic schemes that are used. Thus, data is loaded in a similar way to BigQuery, specifying the key file used and following encryption schemes defined at encrypt field. The data is then stored in Google's cloud, more precisely in the segment Google Cloud Storage.

To make sense to the BigQuery database, the queries are rewritten in a proper way before sent. Thus, queries made using EBQ undergo a sort of interpreter adapting encryption parameters to be pursued, so that the BigQuery database does not get knowledge of the returned results. Likewise, the results received are decrypted with the key used by EBQ and stored on the client. Then, it is delivered to the application, as shown in Figure 1. All access to data stored in a distributed manner is done by integration with the Hadoop at Google BigQuery along with the storage platform.

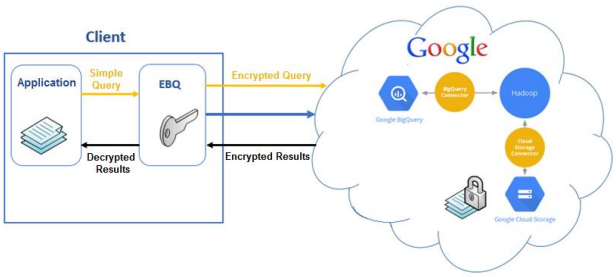

Figure 1. EBQ's Architecture.

In EBQ, the raw data is encrypted in different ways before being inserted into a database. Depending on the type of query to be performed in a certain field, a different technique is used from among the following:

- Probabilistic algorithm: A ciphertext is denoted probabilistic when there are different values for different ciphertexts with high probability [8]. For an efficient design, a block cipher such as AES in CBC mode with an initialization vector (IV) randomly generated is used. Despite ensuring confidentiality and integrity of data in a database, it is not possible to manipulate the encrypted data, except the SELECT command.
- Deterministic algorithm: it generates the same ciphertext for the same plaintext. The algorithm is done with the AES block cipher in CBC mode with a VI of zeros. Thus, the scheme performs a deterministic pseudo-random permutation [6] by using cipher block, which represents a decrease in security level: the opponent can have the knowledge of the encrypted values, which correspond to the same value in the plaintext. The choice of this scheme allows equality queries, i.e., can perform the SELECT command with equality predicates, joins equality, and COUNT [10].
- Searchwords: This algorithm is not necessarily an encryption scheme. The hash function is computed, normally SHA-1, in all possible word sequences. Then, the hashes are kept in a field and separated by spaces. It can be used as an attribute to the WHERE clause with content checking (using CONTAINS) with complete keywords, but does not appear in simple SELECT queries.
- Probabilistic searchwords: it searches for a word W and return all positions where W appears in the plaintext [11]. The main idea of the algorithm is to encrypt a text performing the XOR bitwise operation between the plaintext and a pseudo-random sequence of bits with a specific structure. This configuration provides isolated queries for searches, i.e., unsecured server cannot learn anything from the plaintext beyond searching results. It allows SELECT queries for content checking, using the WHERE clause and CONTAINS or LIKE attribute.
- Homomorphic encryption: Encryption that allows you to handle specific types of calculations in ciphertexts and generates a result also encrypted that, when decrypted, corresponds to the result of operations conducted on plaintexts. Since the fully homomorphic encryption is infeasible, an efficient alternative to be chosen is a partially homomorphic encryption, as Paillier [9], which is an IND-CPA secure scheme. Thus, due to the properties of adding a partially homomorphic encryption, queries that handle with numerical manipulations like sum and average between columns are admitted.

Table I presents a summary of the EBQ´s encryptions schemes where it is noted that the cryptosystem that presents higher level of security is the Probabilistic scheme, followed by Homomorphic, Probabilistic Searchwords, Searchwords and Deterministic encryption.

TABLE I. SUMMARY OF EBQ'S ENCRYPTION SCHEMES.

Cryptosystem	Build	Function	SQL Syntax
Probabilistic	AES in CBC mode	Static data	SELECT, UPDATE, DELETE
Homomorphic	[9]	Sum	SUM, +
Searchwords	Hash SHA-1 in words	Search in keywords	SELECT – WHERE, CONTAINS
Probabilistic Searchwords	[11]	Search in keywords	SELECT – WHERE, CONTAINS, ILIKE
Deterministic	AES in CBC mode with null IV	Equality	=,!=, IN, COUNT

IV. RESULTS AND ANALYSIS

This section aims to apply cryptographic models outlined in Table I, evaluating the performance and overhead added by the cryptosystem when applied in a massive column-oriented distributed data storage system, such as EBQ on Google BigQuery.

The results are based on measurements of execution time of certain queries presented in graphs and tables along with their description and analysis.

A. *Implementation Environment*

Analyses of performance and overhead are done using a computer with a processor core i5 1.8GHz with 6GB of RAM memory and Linux Ubuntu 14.0.4, running EBQ tool that is connected to the internet via Google BigQuery.

EBQ runs through a python base that accesses BigQuery using the command line tool. All transactions are made through this medium access adapting requests made and invoke the functions of insertion and removal query created by BigQuery.

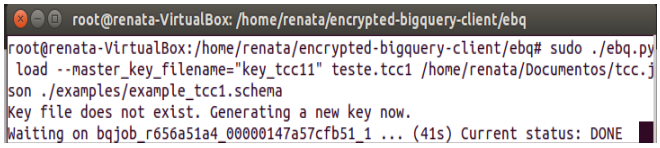

Figure 2. Loading data using command line tool.

Figure 2 illustrates the operation of loading the database. As seen in the first two lines, the user invokes the EBQ tool and fill in the fields indicating which key is used for encryption, the recipient table, the file to be loaded and which schema is followed. In case there is no key, a new key is created and stored with the client. Immediately after that, the request becomes a job to be accomplished by BigQuery.

B. *Implementation Scenario*

For application and demonstration of encryption models, we use a scheme in order to present the application of the use of column families. The example set has as its theme a personal civil registration, which has an identification number and column families divided into registration, biometrics, public, private and secret structures, according to Table II.

Table II also shows the encryption type used for each table columns. The definition of the appropriate cryptosystem for each column strongly influences the performance and content of the query, since certain queries have restrictions depending on the security level chosen for that column.

TABLE II. CRYPTOSYSTEMS APPLIED IN COLUMN FAMILIES

Registration	
Column	**Types of Cryptosystem**
UFcad (Registration State)	Deterministic
Biometrics	
Column	**Types of Cryptosystem**
Dtcad (Registration Date)	Deterministic
Fingerprint	Deterministic
Face	Deterministic
Public	
Column	**Types of Cryptosystem**
Letter	Deterministic
Name	Probabilistic Searchwords
Gender	Searchwords
UFnasc (State of Birth Location)	Deterministic
Anonasc (Birth Year)	None
Dtnasc (Birth date)	Deterministic
Private	
Column	**Types of Cryptosystem**
ID	Searchwords
Job Title	Probabilistic Searchwords
Qtfilhos (Number of children)	Probabilistic
Secret	
Column	**Types of Cryptosystem**
Salary1	Homomorphic
Salary2	Homomorphic

With the schema file containing the structure data we performed the analysis of the impact of querying a large volume of encrypted datasets with input volumes of 5000, 50000 and 100000 records. To perform this operation, adjustments in the code had to be made due to EBQ tool errors that did not allow to enter data in JSON format (format inspired by JavaScript), which enables the exchange of data following a nested structure using collections of name/value pairs, as an object. This format is applied in the case of personal records.

After inserting the data, the Google BigQuery Web interface (Figure 3) offers the information that the data is stored in an encrypted way and that there is no information about the inserted content. The interface also offers a description of the table containing the hash key and the encrypted schema itself, an identification code of the table, the table size and the number of lines.

It is noteworthy that all kind of tables, flat or nested structure, are presented in a unique way, a flat display. The user can perform queries through the interface, however, using the EBQ extent, the composition of the query should be based on the encoded values of the fields as attributes.

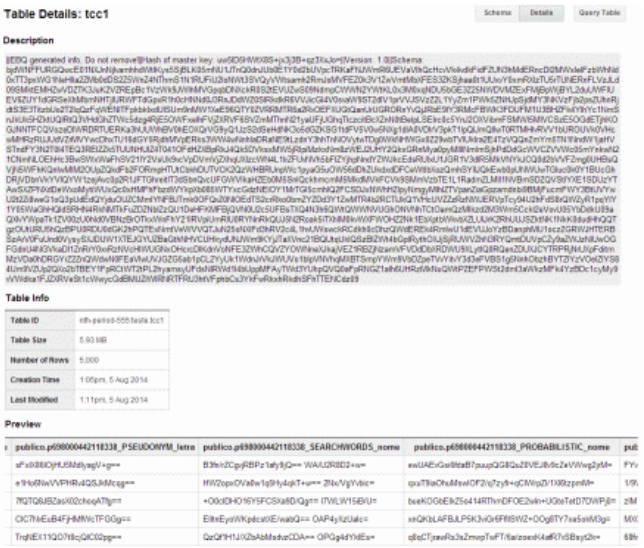

Figure 3. Display of BigQuery Web interface.

C. Queries over Encrypted Data

Once the data was loaded, we performed a set of queries under the SQL language with many attributes and results in order to assess the feasibility of implementing cryptosystems for large volumes of data. The first query consists of a simple SELECT operation that was ordering an attribute for each column family, namely: the number, date of registration, the registered name, and state of birth location, job title and salary.

The second round of query is made with the WHERE clause with attribute equality. This query aims to test the deterministic cryptosystem that admits this type of data. The query returns the name, and date of birth of the registered, which was born in the state of Rio de Janeiro.

The third test was performed with a query using the WHERE clause with the CONTAINS condition, to evaluate the efficiency and the result of the cipher probabilistic searchwords, which should return the number of children registered when it has a job title that contains the word "lawyer". As seen earlier, probabilistic searchwords does not provide search for fragments, only the whole words.

The fourth query is done with the WHERE clause using a parameter without encryption that should return the fields of name, state of birth location and date of birth registered with birth year over 1980. Finally, in the last query we wanted to test the functioning of homomorphic encryption, performing the query using the sum of two columns of salaries encrypted with this cryptosystem.

D. Results Evaluation

The response time of each EBQ query was evaluated using a table without encryption with the same records as a comparison. Each query process was repeated 50 times to obtain an average, since most of the processing is done on Google's servers in a non-controlled environment where there is no guarantee of the correct and expected homogeneous execution.

The first round of testing runs queries on a database of 5000 records. As shown in Figure 4, the EBQ queries add a delay to the response time due to the volume of data that the encryption schemes operate. A simple query has high values since ordered a column of each column family. It is also observed that the homomorphic sum query generates a large response time in relation to query unencrypted due to the processing time of summing numbers added to the decryption time of the result on the client machine. Still, it appears that the use of cryptography in database increases its size by almost 10 times more than the common database because the encrypted values are higher than values unencrypted for some encryption schemes, such as homomorphic encryption. However, performing the same operation in a proper environment for Big Data processing, this increase does not affect the efficiency or speed of dynamic queries.

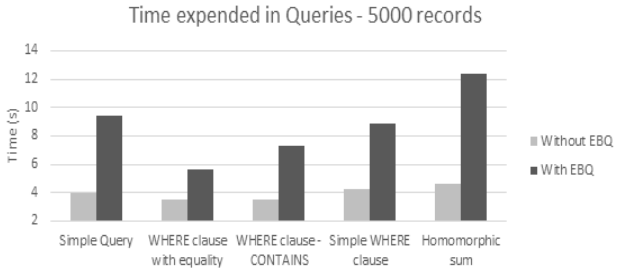

Figure 4. Average time (in seconds) expended in queries - 5000 records.

It has also generated a database with a volume of 50000 records in order to verify the degree of impact that a delay in response time to a query would result in a greater volume of data. In Figure 5 we note that even increasing the amount of data, slack remains apparently controlled, confirming this assertion by presenting the performance obtained with the number of records increased by 10 times its original load. The delays found are small compared with the size of the generated table. Again, it is observed that the most time-consuming queries are those involving reading different column families and those who carry out operations in homomorphic ciphers, requiring extra processing due to their size.

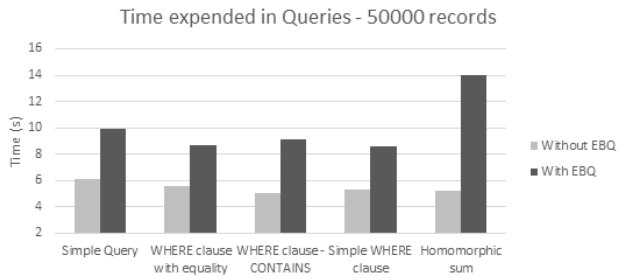

Figure 5. Average time (in seconds) expended in queries - 50000 records.

Conducting a new round of data loading, it can confirm the idea that the increase in the volume of data loaded does not generate much impact in the delays obtained with the use of encryption, such as seen in Figure 6.

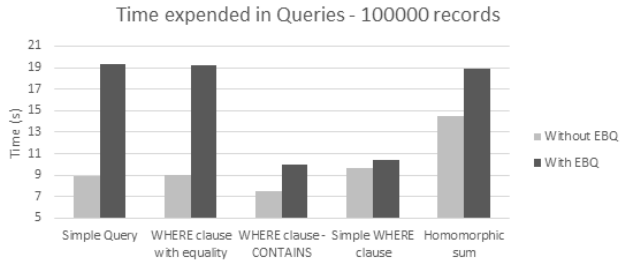

Figure 6. Average time (in seconds) expended in queries - 100000 records.

Therefore, it appears that the application of direct queries on large volumes of encrypted data stored in distributed environment, as the EBQ case, dalays the response time. However, as the volume of data grows, this variation is stabilized and can be considered viable when efficient encryption schemes that add a modest latency are all adopted.

Based on the values of response time obtained, it is possible to model the structure of a database, as well as encryption schemes, used in each column, efficiently in order to optimize data analysis of trial and error in large datasets.

Moreover, making a projection using the model of direct queries on data stored in a distributed manner, we note that the response time to queries is controlled due to their modest and almost linear growth behavior (Figure 7). Such behavior is due to encryption combination with optimized processing large volumes of data, thus the increase of data by the encryption does not affect as strongly as the tool already provides means to process large volume of data. We can infer that this efficiency has a saturation point, which reaches the maximum volume of the storage tool capacity and starts displaying delay values that impair the functioning of the application, the same is not noted in this work.

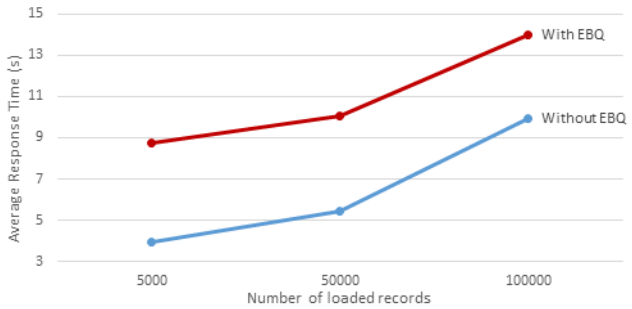

Figure 7. Behavior of the average response time while loading different record sizes.

V. CONCLUSION AND FUTURE WORK

With the increasing number of threats and attacks database suffers, the protection of confidential and sensitive data has become a priority for many applications. However, only access control is not enough to ensure the confidentiality and integrity of data, it is necessary to use encryption on the table fields. For this reason, this paper presented the implementation of a model for storing encrypted data in distributed environments that ensure proven safe means of access, both in the update query as the data with minimal additional loss in performance data recovery.

The implementation of the scenarios and presented results allow the identification of the use of encryption in large volume of data adds more data and a delay to the response time to research. However, with the use of frameworks designed for massive data processing, this extra expenditure of time and memory becomes not so relevant, considering the speed with which transactions are carried out.

As proposal for future work we suggest the inclusion of a broader range of query attributes, often used by traditional databases such as GROUP BY and DISTINCT. In addition, we emphasize the importance of a feature that enables operations with different parameters of column families. Furthermore, it would be interesting to implement a secure server key management allowing the configuration of multiuser access the encrypted data.

ACKNOWLEDGMENT

The authors wish to thank the Brazilian Ministries of Justice and of Planning, Budget and Management, as well as the Brazilian research and innovation Agencies CAPES, FINEP (Grant RENASIC/PROTO 01.12.0555.00) and PNPD/CAPES - Programa Nacional de Pós-Doutorado for their support to this work.

REFERENCES

[1] Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., & Venkatesan, R. (2013). Orthogonal Security with Cipherbase. In CIDR.

[2] Boneh, D., & Waters, B. (2007). Conjunctive, subset, and range queries on encrypted data. In Theory of cryptography (pp. 535-554). Springer Berlin Heidelberg.

[3] Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.

[4] Gentry, C. (2009). A fully homomorphic encryption scheme. Ph.D Thesis. Stanford University.

[5] Goldman, A., Kon, F., Junior, F. P., Polato, I., & de Fátima Pereira, R. (2012). Apache Hadoop: Conceitos teóricos e práticos, evoluçao e novas possibilidades. XXXI Jornadas de atualizaçoes em informatica.

[6] Goldreich, O. (2009). Foundations of Cryptography: Volume 2, Basic Applications (Vol. 2), Cambridge University Press.

[7] Kahate, A. (2013). Cryptography and Network Security, Tata McGraw-Hill, 3ª edition. ISBN: 1259029883.

[8] Kaufman, C., Perlman, R; Speciner, M. (2002). Network Security: Private Communication in a Public World. Editora: Prentice Hall, 2ª edition. ISBN: 9780130460196.

[9] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Advances in cryptology—EUROCRYPT'99 (pp. 223-238). Springer Berlin Heidelberg.

[10] Popa, R. A., Redfield, C., Zeldovich, N., & Balakrishnan, H. (2011). Cryptdb: protecting confidentiality with encrypted query processing. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (pp. 85-100). ACM.

[11] Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on (pp. 44-55). IEEE.

[12] Stallings, W. (2007). Network security essentials: applications and standards. Pearson Education India.

[13] Tigani, J., & Naidu, S. (2014). Google BigQuery Analytics, John Wiley & Sons. 1st edition. ISBN: 978-1-118-82482-5.

[14] Unisys. (2014). Critical Infrastructure: Security Preparedness and Maturity. Independently conducted by Ponemon Institute LLC.