2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Crowdsourced Data Integrity Verification for Key-Value Stores in the Cloud

Grisha Weintraub
Dept. of Mathematics and Computer Science
The Open University
Raanana, Israel

Ehud Gudes
The Department of Computer Science
Ben-Gurion University of the Negev
Be'er-Sheva, Israel

***Abstract*—Thanks to their high availability, scalability, and usability, cloud databases have become one of the dominant cloud services. However, since cloud users do not physically possess their data, data integrity may be at risk. In this paper, we present a novel protocol that utilizes crowdsourcing paradigm to provide practical data integrity assurance in key-value cloud databases. The main advantage of our protocol over previous work is its high applicability - as opposed to existing approaches, our scheme does not require any system changes on the cloud side and thus can be applied directly to any existing system. We demonstrate the feasibility of our scheme by a prototype implementation and its evaluation.**

***Keywords*-cloud; NoSQL; key-value stores, data integrity; secure storage;**

I. Introduction

Relational database management systems (RDBMS) are not the only solution for data storage any more. A large number of non-relational databases have been developed in recent years in order to manage continuously growing amounts of data. The main characteristics of these systems, often referred to as NoSQL [1,2], are flexible schema, horizontal scaling and relaxed consistency. They store and replicate data in distributed systems, commonly across datacenters, thereby achieving scalability and high availability. NoSQL databases are usually classified into three groups, according to their data model:

- *Key-value stores:* Data is stored as key-value pairs, such that the key is a unique identifier and a value is an arbitrary entry.
- *Document-based stores:* Data is stored in a document-like structure such as JSON or XML.
- *Column-oriented stores:* Data is organized in tables, which consists of row keys and column keys. Column keys are grouped into sets called column families.

In this paper, we will focus on the first group - key-value stores.

Key-value stores can be used either as internal database systems or as cloud databases - cloud services that provide users with access to data without the need for managing hardware or software. However, storing data in a cloud introduces several security concerns. In particular, since cloud users do not physically possess their data, data integrity may be at risk. Cloud providers (or some malicious entity) can change users' data, omit some of the data from query results or return a version of the data which is not the latest. In other words, data *correctness*, *completeness* and *freshness* might be compromised.

Data integrity in outsourced relational databases has been studied for several years [13-16, 18-21]. Nevertheless, existing solutions are inappropriate for key-value stores for the following reasons:

- Data volumes in key-value stores are expected to be much higher than in RDBMS and therefore data is (usually) distributed across many different nodes.
- The query model of key-value stores is much simpler than in RDBMS.

These differences between RDBMS and key-value stores introduce both challenges and opportunities. On the one hand, data integrity assurance in key-value systems requires more sophisticated solutions due to its distributed architecture and high data volumes. On the other hand, their extremely simple query model may allow us to design much simpler and efficient protocols for data integrity verification. We are especially interested in developing a method that does not require server side modifications, so it may be seamlessly applied to existing real world systems.

The goal of this paper is to demonstrate that data integrity of key-value stores in the cloud can be verified without server side modifications and with reasonable performance overhead. Our main contributions are as follows:

- Development of a novel probabilistic method that allows users to verify data integrity of the data that resides in cloud key-value stores and its analysis.
- A demonstration of the feasibility of our method through a prototype implementation and its experimental evaluation.

The rest of the paper is structured as follows: Section II outlines system and threat models. Section III presents our method for data integrity verification. Security analysis of our approach is presented in Section IV. Section V introduces our proof-of-concept implementation and provides an experimental evaluation thereof. Section VI reviews related work and Section VII concludes the paper.

978-1-5090-6611-7/17 $31.00 © 2017 IEEE
DOI 10.1109/CCGRID.2017.17

IEEE computer society

II. System and Threat Models

A. System Model and Assumptions

Database-as-a-service paradigm (DBaaS), firstly introduced more than a decade ago [3], has become a prevalent service of today's cloud providers. Micorosoft's Azure SQL Database, Amazon's DynamoDB and Google's Cloud BigTable are just a few examples.

We assume that there are three entities in the DBaaS model:

- *Data owner (DO):* uploads the data to the cloud.
- *Cloud provider (CP):* stores and provides access to the data.
- *Clients:* retrieve the data from the cloud.

There is only one instance of DO and CP in our model, whereas the number of clients is not limited. We will assume that data uploaded to the cloud is stored in a key-value database. We model key-value database as a set of tuples $D = \{(k, v)\}$, where k is the key and v is the value associated with k. The DO uploads data to the cloud by sending to the cloud bulks of the predefined size $B = \{(k, v)\}$. By sending a key k to the cloud, client retrieves a tuple $(k, v) \in D$.

Our system model is both write and read intensive. We assume that the DO only appends new values to the database; existing tuples are not updated. While privacy in the cloud is a topic that received much attention (see for example [22-24]), our model focuses on data integrity. As already mentioned above, we are interested in a highly applicable solution and therefore we assume that no server changes can be performed on the cloud side.

B. Integrity and Attack Model

We assume that the CP is not trusted neither by the DO nor by the clients and that it can behave maliciously in any possible way to break data integrity. For example, the CP may modify some of the tuples, add or delete tuples, or return partial (or empty) results to the clients' queries.

We focus on data integrity protection in the following two dimensions:

- *Correctness:* Data received by the clients was originally uploaded to the cloud by the DO and has not been modified maliciously or mistakenly in the cloud side.
- *Completeness:* The CP returns to the clients all the data that matches the query. In other words, no data is omitted from the result.

Freshness is another important dimension of data integrity, meaning that the clients get the most current version of the data that was uploaded to the cloud. However, since in our system model there are no updates, freshness is not an issue.

III. Our Approach

In our model, clients query the cloud by "get value by key" queries. The result returned by the CP may be either empty or not. In case it is not empty, the client only needs to check its correctness, and that can be easily achieved by data authentication. The major focus of our work is to ensure that empty results are supposed to be empty. In other words, we want to ensure that the results of clients' queries are *complete*.

Below we describe our protection techniques for correctness and completeness verification.

A. Preliminaries

1) Hash Function: We use collision-resistant hash function that has a property that it is computationally hard to find two inputs that hash to the same output. SHA-256 and SHA-512 [4] are examples of such functions. Hash operation on value x is denoted by $H(x)$.

2) Secret Keys: We assume that the DO and the clients share two secret keys $\{K_e, K_m\}$; K_e for data encryption and K_m for data authenticity.

3) Data Authentication: To verify data authenticity we use message authentication codes (MAC's). The DO signs its data according to the MAC scheme (e.g. HMAC [5]) and stores the MAC value in the cloud along with the signed data. Then, based on the MAC value and the received data, clients can verify data authenticity. Signing and verification operations are denoted by $Sign(data, key)$ and $Verify(data, MAC, key)$.

4) Data Encryption: Sensitive data that is stored in the cloud is encrypted by the DO and then decrypted by the clients by using symmetric encryption (e.g. AES [6]). We denote encryption and decryption operations as $Enc(plaintext, key)$ and $Dec(ciphertext, key)$.

B. Completeness Verification Scheme

Our goal is to verify that no data was omitted from the results of the client queries. In our approach, we use two techniques - *tuples linking* and *crowdsourced verification* described below.

1) Tuples Linking: The Intuition behind the *tuples linking* is that every tuple is aware about the existence of some other tuples. For example, consider the following key-value database (Table I), where key is a user id and value is a user object.

Table I
Sample Key-value Database

Key	Value
1457	Name = "Bob", Phone = "781455"
1885	Name = "John", Email = "john@g.com"
2501	Name = "Alice", Email = a@aaa.com"
3456	Name = "Carol", City = "Paris"

If we would apply tuples linking to the sample data from Table I, the result might look as in Table II.

Table II
TUPLES LINKING EXAMPLE FOR DATA FROM TABLE I

Key	Value
1457	Name = ”Bob”, ... , Link_Data = ”1885, 3456”
1885	Name = ”John”, ... , Link_Data = ”2501, 3456
2501	Name = ”Alice”, ... , Link_Data = ”1457, 1885
3456	Name = ”Carol”, ... , Link_Data = ”1457, 2501”

The formal definition of tuples linking is as follows:

$$\forall (k_1, v_1), (k_2, v_2) \in D, (k_1, v_1) \text{ is linked to } (k_2, v_2) \iff k_2 \subset v_1 \tag{1}$$

The DO is responsible for the linking between the tuples when uploading the data to the cloud. Afterwards, the clients can rely on this *linking data* to verify the result completeness. For example, consider a query ”get users with ids 1457, 1885” on data from Table II and the result that contains only tuple with id 1457. By checking linking data of the tuple 1457, the client knows that tuple 1885 should be a part of the result and thus detects the attack.

The implementation of the tuples linking is based on the assumption that the DO uploads its data to the cloud by bulk loading, and therefore tuples inside a bulk may be linked one to another forming a *bulks linking*. However, it is not enough to link between the tuples of the same bulk, because if the CP removes all the bulk there is no way to detect that. We suggest that the DO keeps the copy of the previous bulk locally and when the new bulk is arrived, links its tuples to both the current bulk and the previous one. The number of linked tuples is defined by a parameter p. The bulks linking can be formally defined as follows:

$$\begin{aligned} &\forall B_i, B_{i+1} \in D, x \in B_i, p > 0, \\ &\quad \text{there are } \left\lceil \frac{p}{2} \right\rceil \text{ tuples in } B_i \text{ and } \left\lceil \frac{p}{2} \right\rceil \text{ tuples in } B_{i+1} \\ &\quad \text{that are linked to } x \end{aligned} \tag{2}$$

The illustration of the bulks linking of 3 bulks with 5 keys each is shown in Fig. 1:

- Graph nodes represent database tuples.
- An edge from the node A to the node B depicts that A is linked to B (A knows about the existing of B).

2) Crowdsourced verification: In crowdsourcing (CS) systems [7] users collaborate to achieve a goal that is beneficial to the whole community. The collaboration may be explicit (e.g. Wikipedia and Linux projects) or implicit as in ESP game [8] where users label images as a side effect of playing the game. In our approach we build CS system where users implicitly collaborate to achieve a mutual goal database integrity assurance. Unlike the previous work which focused on the query integrity (clients verify only their own queries), our focus is on the database integrity (clients execute random verification queries to verify that the CP behaves honorably). The rationale behind this strategy is based on the observation that the database integrity is more important than the query integrity; clients would not like to work with the CP that returns provably wrong results to some of the queries (even if these queries are fake and were issued by other users). We rely on this observation in our CS system, where each client contributes a little bit of his computational power in order to verify that the CP can be trusted. It works as follows:

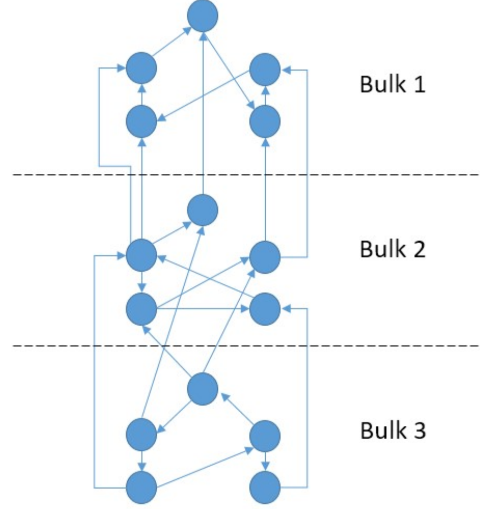

Figure 1. Bulks linking example of 3 bulks with $p = 2$

1) A client sends a query to the CP.
2) The CP sends the query result along with the linking data back to the client.
3) The client builds verification queries based on the received linking data and sends them to the CP.
4) The CP sends the result of the verification queries back to the client.
5) The client verifies that the result of the verification queries matches the linking data.

Verification queries (step 3) are built such that the CP cannot distinguish between them and the regular client queries (step 1). Thanks to that, CP's malicious behavior with the client queries will inevitably cause malicious behavior with the verification queries as well and thus will be detected. For example, consider a query ”get user with id 1885” on data from Table II. The verification queries for this client query will be ”get user with id 2501” and ”get user with id 3456”. If the result of at least one of these queries is empty, the attack is detected.

Note that there is no dependency between client query (step 1) and verification queries (step 3) and hence steps 3-5 can be executed asynchronously (i.e. without hurting reads latency).

C. Pseudo-Code

The pseudo-code of the algorithms that provide both correctness and completeness verification in our model is presented below.

Algorithm 1 puts a bulk of tuples into the cloud database. It iterates over all the tuples in the bulk (lines 3-9) and calculates linking data for each tuple. Linking data then is encrypted and stored as a part of the tuple's value. In order to be able to link the next bulk to the current, we store the current bulk keys in a global variable (line 10).

Algorithm 1 Put a new bulk of tuples

```
1:  procedure PUTBULK(bulk)
2:      Shuffle(bulk)
3:      for each Tuple t ∈ bulk do
4:          link_data ← GenLinkData(GetKeys(bulk))
5:          t.value ← t.value || Enc(link_data, K_e)
6:          t_hash ← H(t.key||t.value)
7:          t.value ← t.value || Sign(t_hash, K_m)
8:          Cloud.put(t.key, t.value)
9:      end for
10:     Global.prev_bulk_keys ← GetKeys(bulk)
11: end procedure
```

Algorithm 2 returns a tuple by key. First the correctness of the tuple is verified in lines 4-6. Completeness verification is based on verification queries which are built based on the decrypted linking data (line 8).

Algorithm 2 Get tuple by key

```
1:  procedure GET(key)
2:      t ← Cloud.get(key)
3:      if t is not empty then
4:          t_hash ←H(key||t.value||GetLinkData(t))
5:          if Verify(t_hash, GetMac(t), K_m) = false then
6:              ALARM ATTACK (Tuple was modified)
7:          end if
8:          ExecVerQueries(Dec(GetLinkData(t), K_e))
9:      end if
10:     return new Tuple(key, t.value))
11: end procedure
```

IV. Security Analysis

A. Correctness

Our correctness verification scheme is directly based on the well-known primitives (MAC scheme and collision resistant hash function) and is secure as long as they are.

B. Completeness

Our approach for completeness verification is based on two techniques: tuples linking and crowdsourced verification, described above.

Assuming a uniform distribution of both deleted tuples and range of queries, the probability that the clients will detect that d tuples were deleted from the DB (or omitted from the result) with $|D|$ tuples after Q queries is:

$$1 - (\frac{|D| - p \times d}{|D|})^Q \tag{3}$$

Fig. 2 shows the probability to detect an attack as a function of number of queries performed by the clients with $|D| = 1.000.000$, $p = 4$ and $d \in \{1, 5, 10, 20\}$. It can be seen that even with p as small as 4, after a relatively small number of queries (production systems receive tens of thousands of queries per second [25]) and deleted tuples, the chance of the CP to escape from being caught is very low.

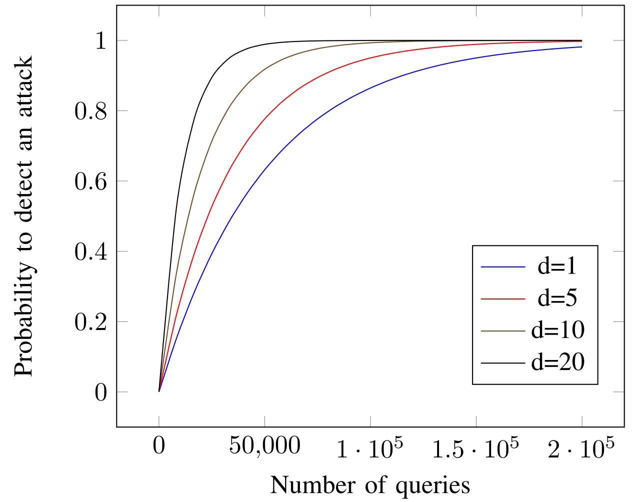

Figure 2. Completeness verification analysis

V. Implementation and Experimental Results

For experimental evaluation, we have implemented a prototype of our solution. As a cloud key-value store we use Redis [9]. In order to evaluate our solution, we use Yahoo! Cloud Serving Benchmark (YCSB) framework [10]. The only thing needs to be done in order to benchmark a particular database with YCSB is to implement a database interface layer. This will allow framework client to perform operations like "read tuple" or "insert tuple" without having to understand the specific API of the database.

We use YCSB in the following way:

- YCSB framework already has Redis client implementation called RedisClient.
- We implemented our version of Redis client (IRedisClient) based on algorithms from Section III. Our implementation is available online [11].
- We configured a workload in which predefined number of interchangeable read and insert operations are executed against the database.

- We executed this workload on both clients and compared their execution time. The results are presented below.

A. Setup

We used tuples of 1KB size with random values. We defined a bulk size to be 100 and the parameter p (number of linked tuples) to be 4. The workload was performed on database that initially contained 30.000 tuples. We executed the workload three times for each client with 5.000, 10.000 and 30,000 operations. The results below represent the average value of these three executions.

B. Performance Analysis

The cost of IRedisClient insert operation is dominated by two encryption and one hash operations (MAC calculation and encryption of linking data). The cost of IRedisClient read operations is similar to the cost of inserts (MAC calculation and linking data decryption) with an additional cost of p verification queries. Experimental results (Fig. 3) show that this overhead increases execution time by 29 % in average.

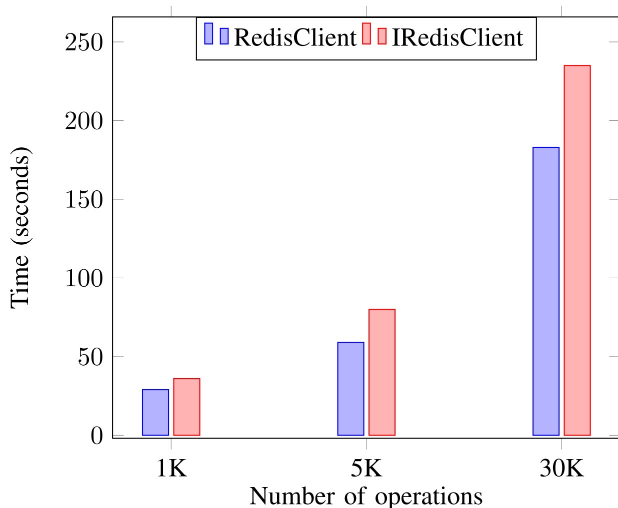

Figure 3. Workload results

VI. Related Work

Existing solutions can be mainly categorized into three types. The first type is based on the Merkle Hash Tree (MHT), the second is based on digital signatures (DS), and the third uses a probabilistic approach.

A. MHT-based approach

MHT [12] is a binary tree, where each leaf is a hash of a data block, and each internal node is a hash of the concatenation of its two children. Devanbu et al. in [13] introduce a method that uses MHT as Authenticated Data Structure (ADS) to provide data integrity assurance in DBaaS model. The general idea is to build MHT for every database table such that MHT's leaves are hashes of table's records ordered by a search key. To reduce I/O operations cost in both client and server sides, instead of using binary trees, trees of higher fanout (MB-Trees) can be used [14]. Different MHT-based techniques to provide efficient integrity assurance for join and aggregate queries are presented in [15] and [16] respectively. MHT-based approach does not suite our system model since it was designed for range queries and does not support efficient queries on arbitrary keys. It also requires significant server side changes and that is not allowed in our system model.

B. DS-based approach

A natural and intuitive approach to provide data integrity in RDBMS is to use digital signatures scheme (e.g. RSA [17]) in the following way:

- An additional column containing a hash of concatenated record values signed by the DO's private key is added to every table.
- Clients verify records integrity by using the DO's public key.

To reduce the communication cost between client and server and the computation cost on the client side, signature aggregation technique [18] can be used to combine multiple record signatures into a single one. In our approach, we use a similar technique for data correctness (the only difference is that we use MAC's instead of DS and do not use aggregation). To guarantee completeness, rather than sign individual records, the DO signs consecutive pairs of records [19]. In order to do that, the DO must either have the copy of the database locally or be able to get the consecutive records from the cloud; both options are not possible in our model.

C. Probabilistic approach

Probabilistic approaches provide only probabilistic integrity assurance, but do not require DBMS modifications and have better performance than MHT-based and DS-based approaches. In this approach, a number of additional records is uploaded to the cloud along with the original records. The more additional records are being uploaded, the higher is the probability to detect data integrity attack. All data is encrypted on a client side so the CP cannot distinguish between the original and the additional records. These additional records may be completely fake as was proposed in [20] or original records encrypted with a different (secondary) secret key as was proposed in dual encryption scheme [21]. Fake-records approach is appropriate only for range queries and therefore is not relevant for our model, whereas dual-encryption scheme could be applied to our model, but requires that the whole database must be encrypted. In our scheme only a small number of keys (i.e. linking data) in each tuple are encrypted. Compared to our approach,

dual-encryption scheme also requires much more additional storage on the cloud side due to records duplication.

VII. Conclusion

In this paper, we present our novel method for data integrity assurance in cloud key-value stores. Our method relies on crowdsourcing paradigm - users collaborate to achieve a mutual goal - database integrity assurance. The main advantage of our method over existing approaches is its high applicability - it can be applied to existing systems without modifying server side. We implemented a proof-of-concept prototype of our protocol and conducted experimental evaluation thereof. The results show that our scheme imposes a reasonable overhead. For future work, we plan to extend our scheme on additional types of cloud databases (e.g. column stores).

References

[1] Leavitt, Neal. ”Will NoSQL databases live up to their promise?.” Computer 43.2 (2010): 12-14.

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

[3] Hacigms, H., Iyer, B., Mehrotra, S. (2002). Providing database as a service. In Data Engineering, 2002. Proceedings. 18th International Conference on (pp. 29-38). IEEE.

[4] Standard, S. H. National Institute of Standards and Technology (NIST), FIPS Publication 180-2, Aug 2002.

[5] Krawczyk, H., Canetti, R., Bellare, M. (1997). HMAC: Keyed-hashing for message authentication.

[6] Pub, N. F. (2001). 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication, 197, 441-0311.

[7] Doan, A., Ramakrishnan, R., Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. Communications of the ACM, 54(4), 86-96.

[8] Von Ahn, L., Dabbish, L. (2004, April). Labeling images with a computer game. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 319-326). ACM.

[9] Redis. http://redis.io/

[10] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R. (2010, June). Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143-154). ACM.

[11] IRedisClient. https://github.com/grishaw/ycsb-iredis-binding

[12] Merkle, R. C. (1989, August). A certified digital signature. In Advances in CryptologyCRYPTO89 Proceedings (pp. 218-238). Springer New York.

[13] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S. G. (2003). Authentic data publication over the internet. Journal of Computer Security, 11(3), 291-314.

[14] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2006, June). Dynamic authenticated index structures for outsourced databases. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data (pp. 121-132). ACM.

[15] Yang, Yin, et al. ”Authenticated join processing in outsourced databases.” Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009.

[16] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2010). Authenticated index structures for aggregation queries. ACM Transactions on Information and System Security (TISSEC), 13(4), 32.

[17] Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

[18] Mykletun, E., Narasimha, M., Tsudik, G. (2006). Authentication and integrity in outsourced databases. ACM Transactions on Storage (TOS), 2(2), 107-138.

[19] Narasimha, M., Tsudik, G. (2005, October). DSAC: integrity for outsourced databases with signature aggregation and chaining. In Proceedings of the 14th ACM international conference on Information and knowledge management (pp. 235-236). ACM.

[20] Xie, M., Wang, H., Yin, J., Meng, X. (2007, September). Integrity auditing of outsourced data. In Proceedings of the 33rd international conference on Very large data bases (pp. 782-793). VLDB Endowment.

[21] Wang, H., Yin, J., Perng, C. S., Yu, P. S. (2008, October). Dual encryption for query integrity assurance. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 863-872). ACM.

[22] Hacigm, H., Iyer, B., Li, C., Mehrotra, S. (2002, June). Executing SQL over encrypted data in the database-service-provider model. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (pp. 216-227). ACM.

[23] Damiani, E., Vimercati, S. D. C. D., Jajodia, S., Paraboschi, S., Samarati, P. (2003, October). Balancing confidentiality and efficiency in untrusted relational DBMSs. In Proceedings of the 10th ACM conference on Computer and communications security (pp. 93-102). ACM.

[24] Ciriani, V., Vimercati, S. D. C. D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P. (2010). Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC), 13(3), 22.

[25] Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M. (2012, June). Workload analysis of a large-scale key-value store. In ACM SIGMETRICS Performance Evaluation Review (Vol. 40, No. 1, pp. 53-64). ACM.