ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Multidimensional Bucketization for Frequent Queries over Encrypted Data

Anuja Antony[1], Silpa Joseph[2]

M.Tech Student, Dept. of CSE, VJCET, MG University, Kerala, India [1]

Asst. Professor, Dept. of CSE, VJCET, MG University, Kerala, India [2]

ABSTRACT: Querying Encrypted data ensuring data and query security has been a difficult task for a very long time. A multidimensional Bucketization technique is introduced to query data based on multiple attributes. In this Bucketization method the encrypted data space is divided into buckets depending on the number of times a data occurs in the database. The method is quite useful for data which have higher demand and can be queried without the need of decrypting them. The Bucketization scheme proposed in this paper is Query Based Bucketization (QBB) which efficiently bucketize the encrypted data based on frequency of occurrence of data. A bucket number is assigned to each bucket in ascending order such that the most frequent data occurs in the initial buckets.

KEYWORDS: Bucketization, Data encryption, Database security, Symmetric encryption

I. INTRODUCTION

The main challenge encountered by an organization today in outsourcing data to a third party service provider is confidentiality. Along with the benefits of saving cost and time comes an unpredictable risk which might be detrimental to the organization. The service provider may have potential access to data above the allowed limit while providing database services. This can be fatal to the actual data owners when their sensitive data confidentiality is compromised. A possibly good solution to this problem is to store the data in its encrypted form. Even though it ensures security of data to an extent the data access with the expected performance by legitimate users posed a serious problem.

Many solutions were proposed for efficient queries over encrypted data. Order-preserving encryption, Bucketization, Hidden vector encryption and special data structure traversals are the main among these. Hidden vector encryption is not used these days as is it more complex and expensive. Based on the requirements the solution that best suits the situation is chosen. For queries (especially range queries) find difficulty in retrieving encrypted data from database. Bucketization is the best option to opt in such situations.

Various Bucketization techniques have been developed through the years with the aim of enhancing the performance [13]. The performance usually comes with compromising the security of data. Some Bucketization designs had been suggested in previous years in order to ensure the safety of sensitive data [7][1][5][9]. But in these cases the number of false-hits is large in comparison to their previous methods. Few techniques were developed addressing this problem [6]. When it comes to querying, the aggregate functions like SUM or AVG doesn’t work well with encrypted data. Queries having some useful operations like ORDER BY, GROUP BY and COUNT doesn’t perform well as they are difficult to implement as they require granular access to data, which somewhat defeats the purpose of using buckets..

Bucket security is sometimes described in terms of bucket width. For clarity, bucket width refers to the number of distinct values in a bucket. In general, wide buckets have relatively many distinct values (more secure) and narrow buckets have relatively few (less secure).

II. RELATED WORK

The problem of efficiently retrieving data from encrypted database without compromising security has attracted a lot of interest from different research communities. Previous methods were mainly focussing on protecting data from service providers [2]. But when security is ensured performance needs to be compromised. Different techniques had been proposed through these years. Order preserving encryption had been one of the most popular one among them. In this encryption the data are stored in buckets in increasing order. Only the minimum and maximum bucket numbers need to be send. All the rows in the buckets in the corresponding range is retrieved. False positives are removed in the

DOI: 10.15680/IJIRCCE.2015. 0311043

post-processing step. Some enhanced methods of OPE were developed (like in [4] and [10]). The drawbacks for the model in [4] were incapability to handle non-numerical data and performance degradation while using aggregate functions. The model in [10] used mutable ciphertext which strengthened the security. But the problem was that mutation of existing ciphertexts is needed to create a new ciphertext and also this model is insecure to a security model. Ordered bucketization was better than OPE in case of range queries.

Some indexing models have also gained interest in previous years [3][5]. The data availability was comparatively more and the disaster protection was high in these models. But along with merits came some drawbacks. Post processing cost increased tremendously and indexes built on bucket tags may lead to inference and linking attacks.

There are only few Bucketization techniques put forward until now[1][7][9][11]. The designs in [7], [9] and [11] mainly focus on the privacy of sensitive data. A combined use of fragmentation and encryption is demonstrated in [7]. The design proposed in [10] also is a combination a few methods. These include generalization, Bucketization and slicing. The main disadvantages are considerable amount of information loss (especially for high dimensional data), does not prevent membership disclosure and not applicable for data that do not have a definite separation between quasi-identifying attributes and sensitive attributes and finally generation of fake tuples due to the loss of utility of the dataset. Clustering and Multi-Sensitive Bucketization is used in [11] to publish micro data with multiple numerical sensitive attributes. The main achievement of the model [1] is the capability to achieve good efficiency on range queries.

False hits are always unavoidable in querying bucket based encrypted databases. A method was introduced to address this problem [5]. Its main aim was to improve the query efficiency by decreasing the expected number of false hits in queries and reconstructing buckets on the statistics of query behaviours. A split-and-merge method was used to serve the purpose.

III. SYSTEM OVERVIEW

On average, decreasing bucket width (by increasing the number of buckets) reduces false positives. This is by letting queries have more granular access to bucket domains, but is not without risk. As Bucketized data can be susceptible to estimation and linking attacks, as well as query access pattern attacks a tighter estimate of the underlying data distribution does not ensure that an adversary can determine precise plaintext values. The advantages of increasing the number of buckets, i.e., reducing false positives, turn invaluable due to a proportional loss of bucket security. So narrow bucketization strategies (like DB) are less secure, and losses the purpose of bucketization.

A. *Query Based Bucketization:*

QBB uses a partitioning strategy. By knowing something about the distribution of queries beforehand, the data store can be partitioned according to the probability with which a value from the data store is likely to be queried. The QBB algorithm operates in two primary steps:

1) Query Mapping:

Query mapping begins by calculating the mass function of an input query distribution. This results in two arrays the first of which contains the cumulative mass values for the query distribution, and the record contains indices mapping the query distribution to the data set.

Initially, QBB takes as input a query set Q where q ε Q with a frequency f_q. The query set is represented as a frequency table where values are sorted from highest to lowest frequency. QBB then takes the maximum frequency to generate the corresponding Zipf (predicted) frequency of each query value. This is given by eq. (1).

$$f_{qi} = F/r^{\rho}, \qquad \text{eq. (1)}$$

where f_{qi} is the predicted frequency of the ith query qi, F is the maximum frequency, r is the rank of qi, and r is the exponent characterizing the shape of the query distribution. The value of r is assumed to be known and set prior to running the algorithm. The mass of a query is then given by its predicted frequency over the sum of predicted frequencies. That is given by eq. (2).

$$pmf(q) = f_q /\Sigma^{N}_{i-1} f_{qi}, \qquad \text{eq. (2)}$$

where N is the size of (number of distinct values in Q). The cumulative mass for a query (given by eq. (3)) is the sum of preceding mass values, where the sum of all mass values is equal to 1.

cdf(qi) = pmf(qi - 1) . eq. (3)

The cumulative distribution is stored in the form of an array, used for the partitioning process in the step 2 (Bucketization).

The last part of this step uses a mapping function to associate each query value with the location of its representative in the data set. Assuming the data set is sorted in, say ascending, order the mapping function searches the data set until it finds the first instance matching the query value and stores its location index in an array. The query values are already indexed in relation to their corresponding values in the cumulative distribution, thus, the latter are now mapped to the data as well.

2) Bucketization:

In the Bucketization step, QBB uses the cumulative distribution array to partition the data set into buckets whose values cumulatively represent some proportion of the predicted query mass.
In this step, the data set is partitioned according to three factors:

- The maximum number of items in buckets M desired by the user,
- For M buckets, the proportion p of buckets allocated to the head bucket and the proportion allocated to the tail bucket 1-p, and
- The proportion of query mass c represented by the head bucket.

Initially, the distribution is split into a head and tail, where the head contains data values representing a proportion of the total query mass as defined by c. The tail contains mass of the proportion 1-c. The precise value of c is specified by the user prior to running the algorithm. For the remaining M-1 buckets, QBB will partition the head bucket into at most [M * p] buckets, then the tail into at least [M * (1- p)] buckets.

Initially, the head bucket is created by finding the first data value representing the proportion of mass closest to c percent of the rightmost bucket. If the head bucket does not have sufficient room for [M * p] buckets, it adds the difference to the number of tail buckets such that the total number of buckets is still M. The tail is partitioned into at least [M * (1- p)] buckets of approximately equal width, where width is determined by number of distinct values. The steps in bucketization are depicted in the block diagram given in the Figure.1.

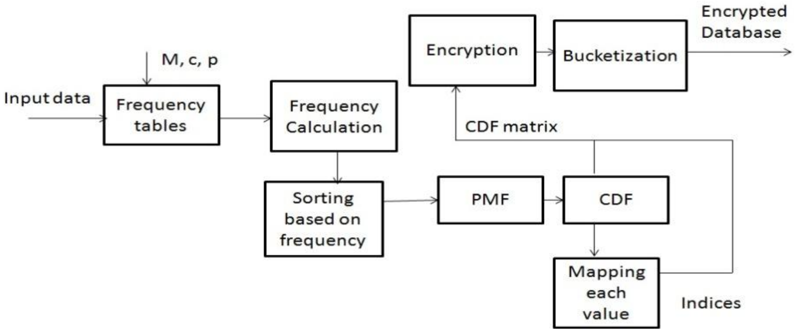


Fig 1. Bucketization

In QBB since the data is not uniformly distributed in the buckets the possibility of guessing attacks and linking attacks can be eliminated. The algorithm is given below:

Algorithm: QBB(D, Q, M, c, p)
Input : Data set D, Query distribution Q =(V, F) (where V = F = n, presorted descending by F), max no. of items in buckets M, criterion value c for primary partition, and percent of buckets p allocated to head bucket (where tail" bucket gets 1- p percent of buckets)
Output: Set of QSB bucket partitions S
Initialize

```
matrix CDF[n] to 0;
matrix S[M * 2] to 0;
matrix IndexMap[n] to 0;
nHeadBuckets = [M * p];
nTailBuckets = [M * (1- p)];
CDF CalculateMass(Q);
IndexMap MapQtoData(Q, D);
find closestCDF;
insert indexMap[k] into S;
insert indexMap[k+1] into S;
insert indexMap[n] into S;
set tailRange to n-k+1;
for j = 1 < nTailBuckets-1 do
    insert indexMap[k+(tailRange=nTailBuckets)*j+1] into S;
    insert indexMap[k+(tailRange=nTailBuckets)*j+1]+1 into S;
end
Return: set of QSB bucket partitions S;
```

B. *Query Execution:*

When a query is given by the client it is executed over the encrypted database rather than the actual database. Initially a connection is established to the encrypted database. With the help of metadata table which stores the mapping data the bucket IDs corresponding to the query is determined. The tuples satisfying the query are retrieved. A filtering process is done after the decryption inorder to remove the false positives that occur in the result. The result is then displayed to the client. Different step in query execution is shown in Figure 2.

The queries can be point or range queries. Only the required fields can be filtered out as required. Different types of operations supported are <, >, <=, >=, = and BETWEEN. Using BETWEEN opertation a range can be specified for each parameter in the condition. ORDER BY operation is also supported for ordering the results in the required order.

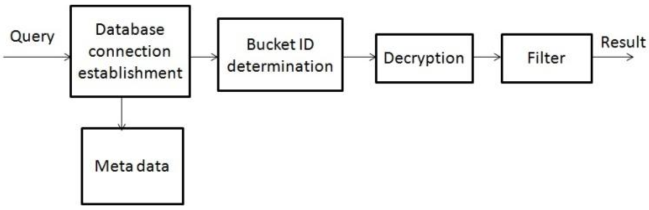
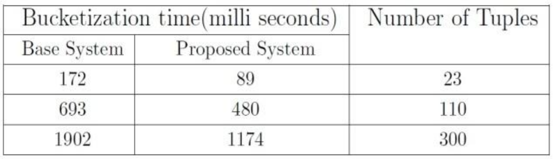


Fig 2. Query Execution

IV. PERFORMANCE EVALUATION

The performance evaluation is based on the execution time and communication time of the base system and the proposed system. Efficiency refers to the time required for an algorithm to perform an of its functions. The functions considered here are the time required to bucketize a data set and the time required for query execution (including decryption) over a data set.

Bucketization time is the time required for each algorithm to build its model over the data set and is given by the difference between the end time and start time of bucketization. Query execution time is the time required for execution of a query (sum of query execution time for false positives and query execution time for the actual result).



Bucketization time(milli seconds)		Number of Tuples
Base System	Proposed System	
172	89	23
693	480	110
1902	1174	300

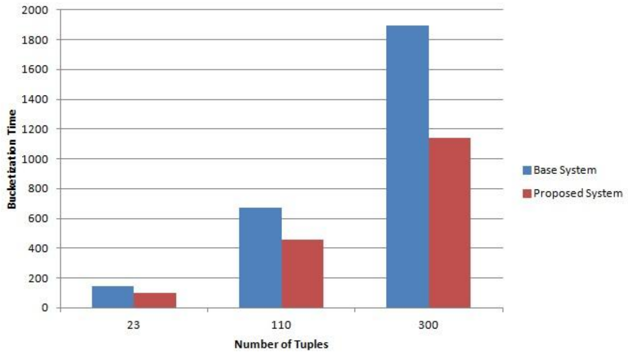


Fig. 3 Bucketization time vs number of tuples graph

Figure 3 plots the bucketization times (in milliseconds) for QBB and EOB. The bucketization time of QBB is less compared to EOB with different number of tuples. This is due to the minimal processing time of QBB. The Table above shows the values for Bucketization in case of base system as well as the QBB.

Execution time		Number of Tuples
Base System	Proposed System	
498	64	23
547	315	110
552	377	273

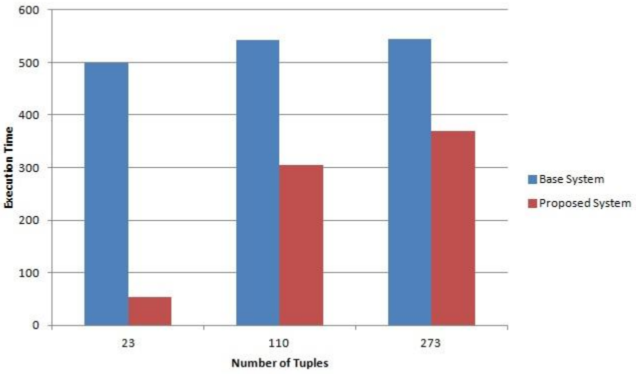


Fig. 4 Execution time vs number of tuples graph

Figure 4 plots the execution times (in milliseconds) for QBB and EOB. For each algorithm, the increase in execution time is proportional to the data sets (Refer Table above). The more distinct values a data set contains, the longer these algorithms will require for computation. For QBB, however, records cannot be inserted or deleted without affecting the bucketization strategy.

For performance evaluation, the method is to compare the communication time of base system and the proposed system. In all the cases the result is that the communication time for proposed system is less compared to the base system. The performance evaluation shows that the compressed results are received faster than uncompressed results. Fig 3 shows that the communication complexity is less for proposed system. It is achieved by using adaptive arithmetic compression.

V. Conclusion and Future Work

The task of efficient data retrieval over encrypted data for queries be resolved using the solution and task is extended to multi-dimensional queries for greater performance. Point queries and range queries are dealt with, also ORDER BY operation can be provided along with the queries.

As future work more number of query types should be handled. In part, this due to the present inability of query languages to work directly with encrypted data. Thus, the types of queries that can be executed on encrypted data are relatively small. For example, basic operations such as SUM or AVG on selected set of records are impossible without decryption.

REFERENCES

1. Younho Lee, "Secure Ordered Bucketization", IEEE Transactions on Dependable and Secure Computing, Vol. 11, Issue 3, 2014.
2. H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model", ACM SIGMOD international conference on Management of data, pp. 216 – 227, 2002.
3. Ernesto Damiani, S.De CapitanidiVimercati, Sushil Jajodia, Stefano Paraboschi and Pierangela Samarati, "Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs", 10th ACM conference on Computer and communications security, pp. 93 – 102, 2003.
4. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order Preserving Encryption for Numeric Data", ACM SIGMOD Int. Conf. Manage. Data, pp. 563 - 574, 2004.
5. B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries", 30th Int. Conf. Very Large Data Bases, pp. 720-731, 2004.
6. Yi Tang and Jun Yun, "A Method for Reducing False Hits in Querying Encrypted Databases", 8th IEEE Int. Conf. on E-Commerce Technology and the 3rd IEEE Int. Conf. on Enterprise Computing, E-Commerce, and E-Services, pp. 7695 - 2511, 2006.
7. Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati, "Fragmentation Design for Efficient Query Execution over Sensitive Distributed Databases", 29th IEEE International Conference on Distributed Computing Systems, pp. 1063 - 6927, 2009.
8. Joel Weis and Jim Alves, "Securing Database as a Service", IEEE Computer and Reliability Societies, pp.1540 - 7993, 2011.
9. S.Kiruthika and Dr.M.Mohamed Raseen, "Enhanced Slicing Models For Preserving Privacy In Data Publication", International Conference on Current Trends in Engineering and Technology, ICCTET13, pp 406 - 409, 2013.
10. R. A. Popa, F. Li and N. Zeldovich, "An Ideal-Security Protocol for Order-Preserving Encoding", IEEE Symp. Security Privacy, pp 463-477, 2013.
11. Qinghai Liu, Hong Shen and Yingpeng Sang, "A privacy-preserving data Publishing Method for Multiple Numerical Sensitive Attributes via Clustering and Multi-Sensitive Bucketization", 6th Int. IEEE Symp. on Parallel Architectures, Algorithms and Programming, pp. 2168-3034, 2014.
12. Bijit Hore, Sharad Mehrotra, Mustafa Canim, Murat Kantarcioglu, "Secure multidimensional range queries over outsourced data", International Journal on Very Large Data Bases, Vol. 21, Issue 3, 2012.
13. Tracey Raybourn, "Bucketization techniques for encrypted databases: quantifying the impact of query distributions (Unpublished masters thesis)", Graduate College of Bowling Green State University, 2013.

BIOGRAPHY

Anuja Antony is a M.TECH Student in CSE, Viswajyothi College of Engineering & Technology, Kerala, M. G University, India.

Mrs. Silpa Joseph is working as Assistant Professor at Department of CSE, Viswajyothi College of Engineering & Technology, Kerala, M. G University, India.

DOI: 10.15680/IJIRCCE.2015. 0311043