Protection of Sensitive Data in a Multi-Cloud Database Based on Fragmentation, Encryption, and Hashing

Iva Jurkovic*, Dejan Skvorc, and Rudolf Lovrencic

Abstract—**With proper data fragmentation and distribution of data chunks to different computer clouds, data owners may stay protected from unauthorized access or data breach even in cases when some of the involved computer clouds get compromised. To reduce the number of cloud providers needed to store fragmented data, we propose an outsourcing model for relational databases that uses a combination of data fragmentation, encryption, and hashing. Since client-side encrypted or cryptographically hashed data significantly limits the ability to process them in the cloud, we propose an algorithm to select the most appropriate data protection method that still enables their processing inside the DBMS without converting them into a plaintext. TPC-H benchmark shows that the proposed algorithm successfully processes all 22 types of reference queries, at the same time being able to reduce the necessary number of cloud storage sites up to 80 percent.**

Index Terms—**Data outsourcing, multi-cloud database, fragmentation, encryption, hashing.**

I. Introduction

With the rapid growth of data collected, stored, and processed, there is an increasing need for organizations to outsource their workloads to an external cloud. One of the biggest challenges in exporting private data to an external computing infrastructure is the lack of trust between data owners and cloud service providers [1]. There are several ways in which data exported to the cloud can be compromised, from targeted external attacks on the cloud infrastructure to disloyal spying by curious cloud administrators to legislations requiring cloud providers to allow access to stored data for inspection and investigation purposes [2], [3].

Recently, the idea of multi-cloud computing has emerged where the data is intelligently partitioned into chunks and then exported to multiple independent computer clouds in order to reduce the necessary degree of trust between data owner and cloud service providers. Portions of data tuples containing sensitive information are separated from each other and deployed to different computer clouds [4], presumably ones that are geographically dispersed and operated by different service providers. Storing the data with different cloud service providers reduces the risk of data leakage or disclosure of confidential information because no participant in the system has access to all parts necessary to disclose the confidential information.

Manuscript received July 9, 2021; revised February 22, 2022; accepted May 3, 2022.

The authors are with the University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia.

*Correspondence: iva.jurkovic@fer.hr.

The problem with using fragmentation as a method of protecting outsourced data is that the required number of data fragments is likely to exceed the number of available cloud service providers, especially for large databases with numerous data links that must be broken prior to outsourcing. In [5], researchers describe metrics for evaluating the security risk for different combinations of multi-cloud data deployments with a limited number of cloud providers, and then select the deployment with the highest security rating using multi-objective optimization. Even with this approach, it is not always possible to split the data into fragments in a way that satisfies all confidentiality constraints.

We expand on this research by combining data fragmentation with various methods of transforming the data from plaintext to ciphertext. To protect the data from unauthorized access from outside, the data transformation methods are applied on the application side prior to outsourcing. The rationale for this approach is that sensitive data pairs can still be stored together if at least part of the pair is stored in ciphertext. This means that some of the sensitive data links are broken by data encryption, which in turn reduces the number of cloud service providers required to store the entire database. In this paper, we combine data fragmentation with deterministic and nondeterministic encryption, limited forms of order-preserving and homomorphic encryption, and cryptographic hashing. Different methods are necessary if we want to enable a database management system (DBMS) to process queries over encrypted data anyway. By applying a particular method to the selected data, we hide the information that meaningfully connects that data to the rest of the sensitive data tuple, but still enable the DBMS to answer queries. For example, the data used in range search queries must be protected with order-preserving encryption, while the data used solely for retrieval can be protected with stronger nondeterministic encryption. In this paper, we propose a method to detect parts of the database that can be cryptographically protected on the application side in such a way that DBMS still retains its full functionality. The detection process and the assignment of specific transformation methods to the specific parts of the database is based on the analysis of the representative queries that the data consuming application sends to the database.

The rest of the paper is organized as follows. Section II presents related work in the area of encrypted databases and multi-cloud related data fragmentation. Section III demonstrates, through an example, the approach to reduce the required number of cloud-based storage sites using a combination of data fragmentation and encryption. Section IV describes the data transformation methods used in our

doi: 10.18178/ijml.2023.13.1.1126

research and analyzes the conditions under which each method is applicable. Section V describes the algorithm for assignment of a particular transformation method to different columns of database tables based on the analysis of the representative queries. The benefits of using the algorithm to reduce the number of cloud providers required to run the fragmented database are discussed in Section VI. Section VII concludes the paper.

II. Related Work

The idea of using cryptography and similar data transformation methods to export encrypted data to untrusted databases is by no means new and is exploited so far by several research groups. Their approaches differ mostly in what portion of the query processing is enabled in the DBMS over the ciphertext, and what is left out to be transferred to the application side that holds the encryption keys.

CryptDB [6] is an early try to combine several different encryption techniques to maintain a confidentiality of the outsourced data. Data are encrypted in layers, where each layer adds a new level of protection to the data encrypted in previous layer. Going from outer to inner layers, encryption is weaker, but allows the DBMS to perform more complex operations over the encrypted data. A minimum set of decryption keys necessary to strip out the outer encryption layers is transferred from client to the DBMS along with the query up to the layer that enables query processing, except for the most inner layer. This way, the DBMS never gets access to the plaintext data, but can still answer certain queries. However, the critics emphasize that such encryption scheme penalizes the DBMS too heavily since many practical queries cannot be answered [7].

Monomi [7] builds on the CryptDB design, but instead of using layered encryption, adds partial query executions. Processing of the query is split between the DBMS and the client. The encryption technique differs from data to data and is selected according to the expected types of queries that target specific data. In case of conflicting requirements, at the DBMS side the query is answered just coarsely and then transferred to the client for final processing. Monomi's approach is similar to the one proposed in this paper, with the difference that in case there is no encryption method suitable for particular part of database, they transfer part of the processing to the client, while we introduce new storage site that enables physical separation of sensitive data from each other. Neither approach is absolutely dominant over another one. They are rather complementary and the decision which one to use depends on the tradeoff between the number of clouds someone is ready to use and the amount of processing that is acceptable to transfer to the client.

Arx [8] is another database system built on the CryptDB design. Unlike the CryptDB which combines several types of encryption, Arx applies only the strongest encryption schemes. To answer queries over the encrypted data, the system uses two proxies between the application and the database server: one to rewrite queries and decrypt the results, and another to maintain indices over the encrypted data and execute queries. This architecture enables that the decryption key is never sent to the untrusted database server. Since CryptDB is implemented on MySQL, while Arx is on MongoDB, they cannot be directly compared in terms of query execution performance.

Seabed [9] is designed for efficient analytics over large encrypted datasets. Since CryptDB and Monomi both rely on expensive cryptographic operations that are too costly for big data analytics, Seabed introduced a new additive symmetric homomorphic encryption scheme. Experimental evaluation shows a significant improvement in response time compared to existing solutions. In addition, they propose dividing sensitive database columns into multiple columns to hide the frequency in the data. This approach provides better security, but is more costly than regular deterministic encryption.

Zeph [10] offers users the ability to set the privacy level of their data, i.e. to decide how the data are to be shared and processed. Users can set their preferences in four levels: share with no one, share with specific users only, share without restrictions, and share only a general view. Depending on the settings, the data in the database is encrypted accordingly. The similarity between Zeph and our proposal is that data owners classify the sensitivity of particular data groups by themselves. However, Zeph do not offer fine grained sensitivity relations among the groups.

The approach most similar to the one presented in this paper is described in [4] where a combination of encryption and data fragmentation is used. The premise is that data remain secure if they get physically separated from the rest of the sensitive tuple they are semantically connected with. Any data that falls under the confidentiality constraint and cannot be divided due to an insufficient number of clouds is encrypted, leaving only one element of the sensitive tuple in plaintext. This approach provides decent data security, but is quite inefficient since the encrypted data columns have to be retrieved from the database in its entirety and then transferred to the client side for decryption. We build on this approach by using several different encryption methods instead of a single one. Since some of the schemes allow certain computations over the encrypted data, we map them to particular database columns only if this does not prevent the DBMS to process the expected set of queries. Otherwise, we fragment the data and deploy them to separate computer clouds.

III. Reduction of Data Fragments

Suppose a database containing sensitive information needs to be outsourced to a public cloud. In our example, the database consists of a single table with the following columns: `first_name`, `last_name`, `city`, and `age`. Let suppose that person's living place and the age are private information that any unauthorized party should not be able to link with a person's identity. Therefore, the data are considered secure if person's first and last name are not accessible together with the information about its living place and the age. However, if we suppose that in a small group people can still be identified based solely on their living place and age, even if no one has access to the part of the database storing their first and last name, we have to further separate these two pieces of information.

We describe the confidentiality relations among the database columns using the *confidentiality constraints*. A

confidentiality constraint contains the columns that stored together may reveal a sensitive information. To secure the data, it is mandatory to separate at least one of the columns from the other columns that belong to the same confidentiality constraint. In our example database, we assume the following confidentiality constraints:

- `first_name, city`
- `last_name, city`
- `first_name, age`
- `last_name, age`
- `city, age`

We can satisfy the given set of constraints if we separate the data into three fragments, as shown in Fig. 1. The first fragment contains the columns `first_name` and `last_name`, while the second and third contain `city` and `age`, respectively. Since the `first_name` and `last_name` never appear in the same confidentiality constraint, they may belong to the same data fragment i.e. they are allowed to be stored together on the same cloud. To run a fragmented database, we need three independent computer clouds.

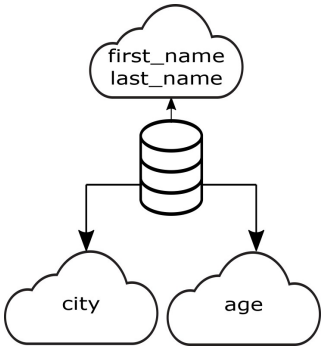


Fig. 1. Protection of data confidentiality in a multi-cloud database based on fragmentation. Three computer clouds are necessary to run a database.

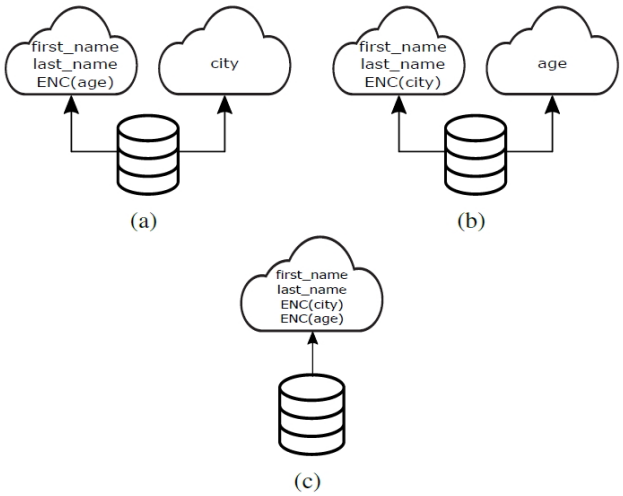


Fig. 2. Protection of data confidentiality in a multi-cloud database based on combination of fragmentation and encryption. Only two (a and b), or even a single computer cloud (c) is enough to run a database.

With a large number of columns and confidentiality constraints, it is likely that the number of cloud providers necessary to deploy a fragmented database would not be available. In the given example, if only two cloud providers are available, there is no way of splitting the data into two fragments without breaking at least some of the confidentiality constraints. However, if we encrypt the data prior to deployment to the cloud, then we can rejoin some data pieces that should not be stored together otherwise. Storing the data in ciphertext effectively breaks the confidentiality constraints among the data pieces. Fig. 2 shows three viable database deployments where encryption is selectively applied to a subset of the database columns. Each of them satisfies the given set of confidentiality constraints, while at the same time reduces the resources needed to run the database from three to two or even a single cloud.

IV. Data Encryption Methods

Data fragmentation involves dividing the database into fragments that are stored separately. Confidentiality is achieved through an increased number of storage sites. Since the fragmentation just breaks the meaningful connections among the semantically linked data parts, but leaves the actual data in plaintext, computations required to answer the database queries can still be performed in the cloud. On the other hand, application-side encryption is useful to reduce the necessary number of storage sites, but makes the cloud-based computations impractical. Thus, instead of encrypting the entire database, we prefer to apply the encryption selectively to particular columns to enable the DBMS to still process the queries without having access to the encryption keys. In addition, we simultaneously combine several different encryption methods and apply the one that suits the most to a particular database column. If none is applicable, then this database column if split from the rest of the sensitive data using fragmentation. List of encryption methods used in our approach and the database operations they support over the encrypted data is shown in Table I.

TABLE I: Database Operations Supported Over the Encrypted Data for Various Encryption Methods

Method / Operation	Project	Eq	Order	Sum	Mul
RND	✓				
DET	✓	✓			
OPE	✓	✓	✓		
HOM-P	✓			✓	
HOM-G	✓				✓
HASH		✓			

A. Random Encryption (RND)

Random encryption transforms two equal plaintexts into two different ciphertexts. This is considered the most secure confidentiality protection method, but does not allow for effective computation with ciphertexts. The only thing the cloud-based DBMS can do is to deliver the encrypted data for the application-side decryption.

B. Deterministic Encryption (DET)

In deterministic encryption, equal plaintexts are converted into equal ciphertexts, which allows for equality checks over the encrypted data. The application holding the encryption keys can still retrieve the encrypted data from the storage, decrypt it locally, and disclose the hidden information. Compared to random encryption, this method is considered weaker in terms of security since it is possible to tell which parts of the data have the same value. There is a high risk of

confidential information disclosure if the range of values is small, or frequency of the same data brings valuable information to an attacker.

C. Order-preserving Encryption (OPE)

With order-preserving encryption [11]–[15], it is possible to compare plaintext values by comparing their ciphertexts. This is useful to perform database functions, such as `ORDER BY`, `MIN`, and `MAX`, without having access to the plaintext data. The method is reversible, which means that the application can still select encrypted data from the database and decrypt the original values. In terms of security, this method is considered the weakest of those described, since revealing the order of the elements carries a lot of useful information to the attackers.

D. Homomorphic Encryption (HOM-P, HOM-G)

Homomorphic encryption is a type of encryption that allows for mathematical calculations over the encrypted data. The result of the operation is protected as well since its value has to be decrypted on the application side. Although practically applicable fully homomorphic encryption scheme is not developed yet, efficient methods exist for certain operations. Paillier homorphic encryption (HOM-P) [16] allows summations of ciphertexts, while multiplication over the encrypted data is possible with the ElGamal cryptosystem (HOM-G) [17]. Both methods are reversible and allow for ciphertext retrieval and application-side decryption.

E. Hash Function

Cryptographic hash function maps arbitrary size data into fixed size ciphertext. It allows for equality checking as equal data are always hashed into the same value. The hashing is keyless, which does not require a complex cryptographic key distribution if many actors access the stored data. In certain situations, this makes it more practical than random encryption, although both provides very strong confidentiality protection. However, keyless hashing is also its main drawback since the transformation is irreversible and does not allow to revert the plaintext once the value is stored as ciphertext. Hashing is applicable if the data exported to the cloud is used only for equality checking and are never retrieved back from the database (e.g. passwords).

V. Encryption Method Selection

The solution proposed in this paper combines data fragmentation, encryption, and hashing to hide sensitive relations among the data in database columns. For each column, an appropriate protection method is selected. The selection of the method depends on how the data from a given column are used in queries. The goal is to select a method that supports all the required database operations for a given column in a set of expected queries. As shown in Fig. 3, during the first step we analyze the queries sent from the database consuming application and keep track of what database operations are required over particular columns. During the second step, we select the strongest encryption method for each column that still enables all the required database operations over that column, according to the Table I. In case of conflicting requirements where none of the methods are applicable, we apply a data fragmentation to physically separate a given column from the rest of the database it is constrained with.

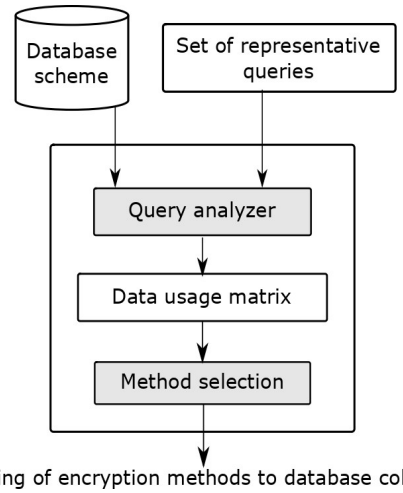


Fig. 3. Selection of the encryption methods for database columns based on query analysis.

A. Query Analysis

The process of method selection starts with query analysis. The input to the analyzer is a database schema with table and column names and a set of queries sent to the database from the consuming application. Results are represented as an $n \times m$ matrix as shown in Table II, where rows are assigned to different database columns, while columns represent different types of database operations. The analyzer iterates through queries, following the procedure described in the Algorithm 1, and marks in the matrix what database operations are required over particular database columns.

For the projection clause, each item in the query projection list, which can either be a named single column or a more complex expression, is analyzed according to the Algorithm 2. If a single column is specified, then a query is used to retrieve the data from that column. The column is marked in matrix for retrieval (column *Project* in Table II). If complex functional expression is specified instead, then we recursively analyze what type of function is expected to be performed. At the end of recursion, columns used in summation and multiplication expressions are marked for summation and multiplication, respectively (columns *Sum* and *Mul* in Table II), while those used for other types of operations are marked as unsupported (column *Other* in Table II) since there is no encryption method that supports calculation for that type of operation.

Join conditions, as well as conditions in `WHERE` and `HAVING` clauses, are analyzed following the steps in the Algorithm 3. Depending on the type of condition, it is decomposed into subexpressions, each of which is further analyzed using the Algorithm 2 with the appropriate query type as a parameter.

For the `GROUP BY` and `ORDER BY` clauses, that in turn check the equality and calculate the order of the data, we start the subexpression analysis with an indication that the final result is used for equality check or order calculation. If

subexpression is a single database column, that column is immediately marked for equality check or comparison (columns *Equal* and *Order* in Table II). If subexpression has a more complex structure, then it is further decomposed and analyzed recursively.

Fig. 4 shows the analysis of the Query (1) according to the aforementioned procedure. The results are shown in Table II.

Algorithm 1 Query analysis

Input: Database schema, Set of representative queries
Output: Data usage matrix

```
1  for each query ∈ representative_queries do
2    for each clause ∈ query do
3      if projection_clause then
4        if subquery then
5          representative_queries += subquery
6        else
7          for each expr ∈ projection_clause do
8            a_ex(expr, "project")
9      if from_clause then
10       if subquery then
11         representative_queries += subquery
12       else
13         for each join_on_condition ∈ from_clause do
14           a_con(join_on_condition)
15     if where_clause then
16       for each condition ∈ where_clause do
17         a_con(condition)
18     if group_by_clause then
19       for each expr ∈ group_by_clause do
20         a_ex(expr, "equal")
21     if order_by_clause then
22       for each expr ∈ order_by_clause do
23         a_ex(expr, "order")
24     if having_clause then
25       for each condition ∈ having_clause do
26         a_con(condition)
```

Algorithm 2 Expression analysis (a_ex)

Input: expr, query_type

```
1  if expr is column_name then
2    mark_in_matrix(column_name, query_type)
3  if expr is conditional_expr then
4    a_con(when_condition)
5    a_ex(then_expr, query_type)
6    a_ex(else_expr, query_type
7  if expr is function_expr then
8    if expr is sum_function then
9      for each function_expr do
10       a_ex(function_expr, "sum")
11   else if expr is mul_function then
12     for each function_expr do
13       a_ex(function_expr, "mul")
14   else
15     for each function_expr do
16       a_ex(function_expr, "other")
```

Algorithm 3 Condition analysis (a_con)

Input: condition

```
1  if condition is comparison_condition then
2    if condition is relational_comparison then
3      if checks equality then
4        a_ex(left_expr, "equal")
5        a_ex(right_expr, "equal")
6      if checks order then
7        a_ex(left_expr, "order")
8        a_ex(right_expr, "order")
9    if condition is between_condition then
10     a_ex(main_expr, "equal")
11     a_ex(lower_expr, "equal")
12     a_ex(upper_expr, "equal")
13     a_ex(main_expr, "order")
14     a_ex(lower_expr, "order")
15     a_ex(upper_expr, "order")
16   if condition is in_condition then
17     a_ex(in_expr, "equal")
18   if condition is is_null then
19     a_ex(is_expr, "equal")
20   if condition is like_condition then
21     a_ex(left_expr, "other")
22     a_ex(right_expr, "other")
23 if condition_with_subquery then
24   if condition is in_condition then
25     a_ex(left_expr, "equal")
26     analyze_query(subquery)
27   if condition is exists_condition then
28     analyze_query(subquery)
```

```
SELECT
  name,
  SUM(price*(1-discount)) AS revenue
  SUM(CASE
    WHEN nation = "CROATIA"
    THEN volume
    ELSE 0
  END) / SUM(volume) as mkt_share                    (1)
FROM
  lineitem
WHERE
  shipdate >= 1.6.2019
  AND shipdate < 1.6.2020
  AND discount BETWEEN 0.4 AND 0.6
GROUP BY
  name
ORDER BY
  revenue DESC;
```

TABLE II: RESULTS OF QUERY ANALYSIS WITH INDICATION OF WHAT TYPE OF DATABASE OPERATIONS ARE REQUIRED FOR PARTICULAR DATABASE COLUMNS (DATA USAGE MATRIX FOR QUERY (1))

DB Column / Operation	Project	Eq	Order	Sum	Mul	Other
name	✓	✓				
price					✓	
discount		✓	✓			✓
nation		✓				
volume				✓		
shipdate		✓	✓			

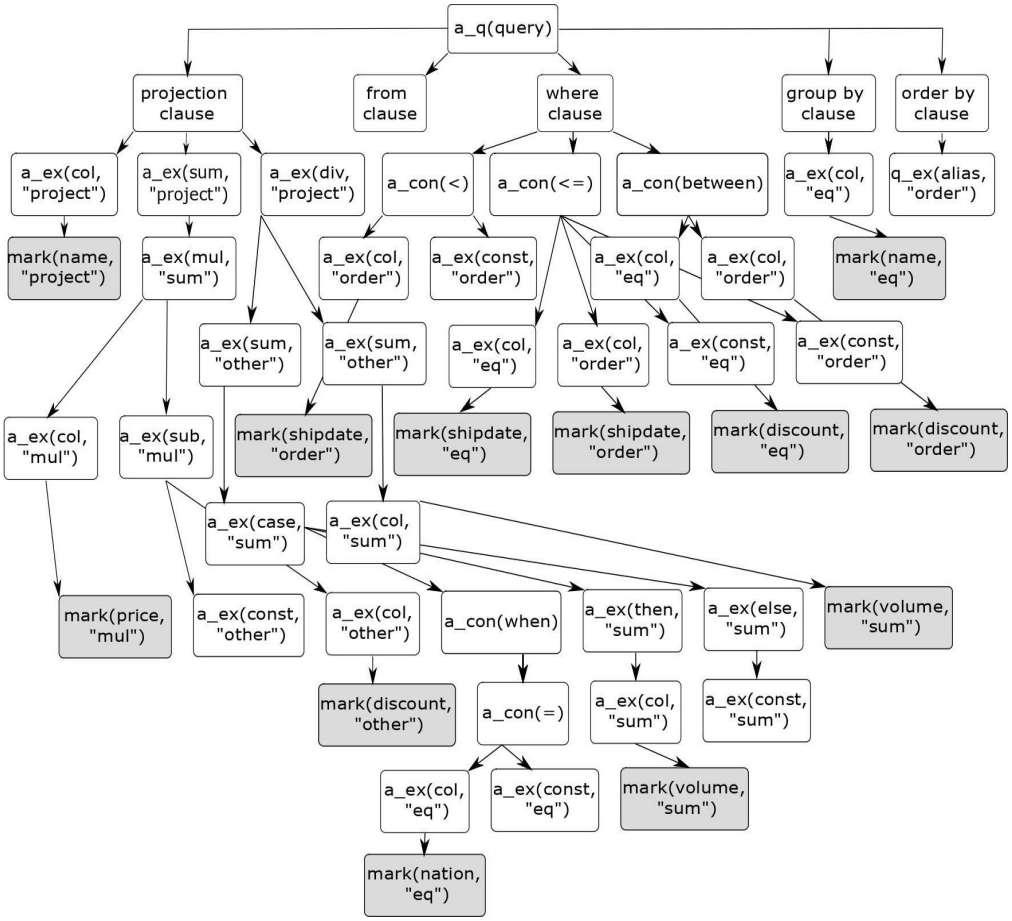


Fig. 4. Example of the query analysis (algorithm invocation tree for Query (1) is shown).

B. Method Selection

The output of the query analyzer is used as input for the method selection. Based on the marcation in the data usage matrix, a database column-wise list of applicable methods is generated. From the given list, the method with the highest confidence is selected to be applied to the data.

A simple example is used to explain this process. The input to the analyzer is a database from Section III, which contains a table with four columns (`first_name`, `last_name`, `city`, and `age`). Initially, a 4×5 matrix is created with all the cells empty (4 rows for 4 database columns, 5 columns for 5 database operations). Suppose that a set of representative queries contains only the Query (2).

```
SELECT first_name, last_name
FROM table
WHERE age > 26
AND city = "Rijeka"
```
(2)

Results of the query analysis are shown in Table III. The columns `first_name` and `last_name` are used for retrieval and are delivered to the consuming application as a result of the query execution. The column `city` is used for equality checking, while the column `age` is used for order calculation. Both operations need to be performed internally by the DBMS in order to produce valid query results.

Since this is the only query that is expected to ever hit the database, the columns `first_name` and `last_name` can be encrypted using any method, except the irreversible hash function. When selecting the method, random encryption would be chosen as it provides the highest security among all. The column `age` can only be encrypted using the order-preserving method, since the DBMS has to compare the values for order. The column `city` can either be encrypted with deterministic encryption or hashed since both methods enable the DBMS to perform equality check over the encrypted data.

TABLE III: QUERY ANALYSIS RESULTS WITH ONLY QUERY (2) IN A SET

DB Column / Operation	Project	Eq	Order	Sum	Mul	Other
first_name	✓					
last_name	✓					
city		✓				
age			✓			

TABLE IV: QUERY ANALYSIS RESULTS WITH QUERY (2) AND QUERY (3) IN A SET

DB Column / Operation	Project	Eq	Order	Sum	Mul	Other
first_name	✓					
last_name	✓					
city		✓				
age			✓	✓		

Since there exists at least one applicable encryption method for each column, we can export the entire database to a single cloud without any risk of disclosure of sensitive relations among the data. Moreover, confidentiality constraints would be satisfied even if one out of three groups of data (either `first_name` and `last_name`, or `age`, or `city`) remains in plaintext. Therefore, we have four different

possibilities for deployment of the given database to the cloud, as shown in Fig. 5a.

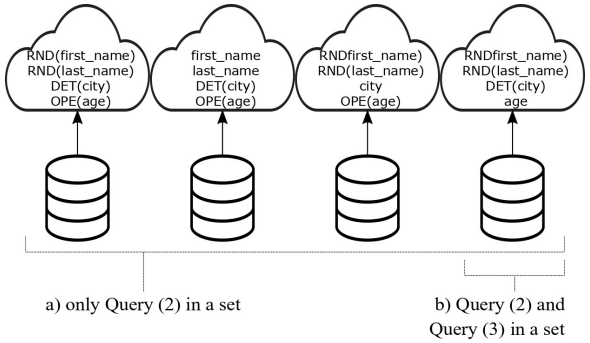


Fig. 5. Database deployment solutions for the example query sets.

If the query set gets extended with the Query (3), then the result of the analysis is slightly different, as shown in Table IV.

```
SELECT SUM(age)                    (3)
 FROM table
```

The column age can no longer be encrypted since it is now used both to compare the values and to calculate the sum. There is no encryption method available so far that supports both operations over the encrypted data. In this case, we have to export the column age in plaintext. Fortunately, we can still apply the encryption to the rest of the columns and export the entire database to a single cloud, as shown in Fig. 5b. Otherwise, database fragmentation would step in, which would in turn require one additional computer cloud to run the database.

VI. Validation of Data Fragment Reduction

We validated the process of encryption method selection using the TPC Benchmark™ H (TPC-H) [18]. TPC-H is a decision-support benchmark with broad industry-wide relevance. It contains a database of large data sets and provides a list of queries with a high degree of complexity. The database consists of 8 individual tables that have a total of 61 columns. The benchmark contains 22 queries that are used for the method selection.

Since confidentiality constraints are not specified in the benchmark, the experiment assumed the worst-case scenario – each database column is constrained with every other column. The results of the validation are presented in Table V.

TABLE V: Reduction of the Number of Computer Clouds Necessary to Deploy the TPC-H Database Using a Combination of Fragmentation, Encryption, and Hashing

Data protection method	Required number of clouds
Fragmentation only	61
Fragmentation + encryption + hashing	13

If protection of the database would rely solely on the data fragmentation, the minimum number of cloud providers required to split and deploy all the data into separate fragments is equal to the number of columns. For the TPC-H database, this means that 61 cloud providers are required to export the data in a secure manner.

Analyzing 22 representative queries from the TPC-H documentation using the procedure described in Section V, 48 out of 61 columns can be transformed using one of the methods described in Section IV. This means that the number of cloud providers required to run the database is reduced from 61 to 13 or by 78.69 %. This not only reduces the operational costs of running a multi-cloud database, but also significantly improves the performance because of much less inter-cloud join operations that are necessary to rebuild the fragmented data.

Compared to [4], where only a single encryption scheme is enabled along the fragmentation, our approach is far more efficient in terms of necessary number of storage sites. In [4], only the data that can be encrypted randomly may reside alongside the plaintext data. For the TPC-H query set, this reduces the number of required cloud providers by 18 %, i.e. they still need 50 cloud providers to store and process data securely, instead of 13.

As expected, CryptDB [6] and Monomi [7] outperform our solution in terms of storage sites since both are inherently single-cloud database systems. However, using a combination of fragmentation and multiple types of encryption, we can successfully answer the entire set of 22 reference queries specified in the TPC-H benchmark, unlike the CryptDB which is able to process only 4 of them [7]. Monomi, on the other hand, can answer the entire set of reference queries, but requires the entire database columns to be transferred to the client for final filtering if they are encrypted with the scheme that does not fit the given query. Our solution never transfers excessive data to the client, but requires simultaneous access to multiple storage sites and completion of query processing at the client side to rejoin the fragmented data tuples. As we already stated in Section II, neither approach is absolutely dominant over another one. They are rather complementary and the decision which one to use depends on the tradeoff between the number of clouds someone is ready to use and the amount of processing that is acceptable to transfer to the client.

VII. Conclusion

A decent balance between running a data storage on-premise and taking a risk of sensitive data disclosure when exported to the cloud is using a multi-cloud database. Besides the increased operational costs and performance penalties imposed by distribution of data among several independent clouds, practical problem is how to find an adequate number of dependable and trustworthy cloud providers. In this paper, we propose a methodology for reducing the number of computer clouds necessary to run a multi-cloud database based on combination of fragmentation, encryption, and hashing. By using the combination of methods, the data remain secure in fewer fragments than if a single method was used. Furthermore, by intelligent selection of different cryptographic methods to protect different parts of the database, the DBMS still retains the capabilities for query processing, regardless of the encrypted data. Validation of the process over the TPC-H benchmark resulted in a reduction of the number of cloud providers up to 80%, compared to using only the fragmentation.

Further research would be directed towards the multi-modal data encryption and the dynamic properties of

the query processing. When no encryption method is applicable for a particular database column, multi-modal encryption allows that a single data is encrypted and stored in several different forms, each of which supports different types of queries. Another direction is to detect frequent computations used in queries, precompute the results, and store them as new columns in the database. By storing elementary data items separate from the precomputed expressions we can still hide the sensitive information, while enabling the DBMS to answer even more complex queries over the encrypted data. However, a tradeoff between introducing new columns with derived data items and database normalization are yet to be explored.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Dejan Skvorc was a principal investigator and research supervisor. He designed the methodology and developed the idea. Iva Jurkovic conducted the research, implemented the algorithms, and validated the results. Rudolf Lovrencic revised the methodology and prepared the validation framework. All authors discussed the results and contributed to the preparation of the paper.

Funding

This research is co-sponsored by the European Union Regional Development Fund through a research grant KK.01.2.1.01.0109 "Cloud Computing Security during the Use of Mobile Applications".

Acknowledgment

We acknowledge the support of the Ministry of Economy of the Republic of Croatia as well as our research partners OROUNDO Mobile GmbH Austria and OROUNDO Mobile GmbH Subsidiary Croatia.

References

[1] K. Hashizume, D. G. Rosado, E. Fernandez-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing," *Journal of Internet Services and Applications*, vol. 4, no. 1, pp. 1-13, 2013.

[2] H. Tianfield, "Security issues in cloud computing," in *Proc. 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2012, pp. 1082-1089.

[3] J. Sarabdeen, M. Mazahir, and M. Ishak, "Impediment of privacy in the use of clouds by educational institutions," *Journal of Advances in Information Technology*, pp. 167-172, 2015.

[4] V. Ciriani, S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Combining fragmentation and encryption to protect privacy in data storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 3, pp. 1-33, 2010.

[5] R. Lovrencic, D. Jakobovic, D. Skvorc, and S. Gros, "Security risk optimization for multi-cloud applications," in *Proc. International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, Springer, 2020, pp. 659-669.

[6] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 85-100.

[7] S. L. Tu, M. F. Kaashoek, S. R. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. the VLDB Endowment*, 2013, vol. 6, no. 5, pp. 289-300.

[8] R. Poddar, T. Boelter, and R. A. Popa, "Arx: an encrypted database using semantically secure encryption," in *Proc. the VLDB Endowment*, 2019, vol. 12, no. 11, pp. 1664-1678.

[9] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with Seabed," in *Proc. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 587-602.

[10] L. Burkhalter, N. Kuchler, A. Viand, H. Shafagh, and A. Hithnawi, "Zeph: Cryptographic enforcement of end-to-end data privacy," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pp. 387-404, 2021.

[11] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 04)*, 2004, pp. 563-574.

[12] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. 2013 IEEE Symposium on Security and Privacy*, 2013, pp. 463-477.

[13] F. Kerschbaum and A. Schropfer, "Optimal average-complexity ideal-security order-preserving encryption," in *Proc. the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 275-286.

[14] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 656-667.

[15] A. Tueno and F. Kerschbaum, "Efficient secure computation of order-preserving encryption," in *Proc. the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS 20*, 2020*)*, pp. 193-207.

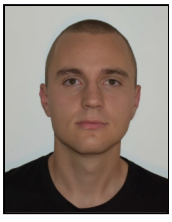
[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 223-238.

[17] G. Xiang, B. Yu, and P. Zhu, "A algorithm of fully homomorphic encryption," in *Proc. 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, 2012, pp. 2030-2033.

[18] TPC BENCHMARK™ H. (2021). *Transaction Processing Performance Council, rev. 3.0.0*. [Online]. Available: http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.0.pdf

Iva Jurkovic is a Ph.D. student at the University of Zagreb, Faculty of Electrical Engineering and Computing. Her research interests include cloud computing, data security, and relational databases. She received her master's degree in computer science from the University of Zagreb.

Dejan Skvorc is an associate professor at the University of Zagreb, Faculty of Electrical Engineering and Computing, and principal investigator for the project under which this research is conducted. His research interests include distributed systems, cloud computing, and mobile and web information systems. He received his Ph.D. in computer science from the University of Zagreb. He is a member of the IEEE and former chair of the IEEE Croatia Section Computer Chapter.



Rudolf Lovrencic is a Ph.D. student at the University of Zagreb, Faculty of Electrical Engineering and Computing. His research interests include cloud computing, parallel computing, and application of optimization algorithms. He received his master's degree in computer science from the University of Zagreb.