Efficient and Secure Encryption Adjustment for JSON Data

Maryam Almarwani, Boris Konev and Alexei Lisitsa
Department of Computer Science, University of Liverpool, Ashton Street, Liverpool, U.K.

Keywords: Onion Layers Encryption, Encryption Adjustment, Document-store, Communication Cost, Efficiency, Security Trade-off.

Abstract: Querying over data protected by multi-layered encryption requires encryption level adjustments. Different layers of encryption enable different sets of operations over encrypted data at the expense of possibly reducing protection levels and releasing some information about the data. Trade-offs between functionality and data security are non-trivial and have been addressed in previous work for various data models. In this paper, we consider encryption level adjustments for JSON-formatted data and propose new adjustments methodologies, based on *sorted criteria* and on *update-aware* principles. Sorted criteria are used to limit the number of encryption layers adjusted, while update-aware adjustments limit the number of updated values. We report on the empirical evaluation of the policies and show that they improve over previously proposed policies in terms of communication costs, exposing information, and performance.

1 INTRODUCTION

Database encryption, which protects sensitive data prior to storage, has been widely used to resolve multiple data security concerns. Query processing for encrypted data poses some challenges and trade-offs between functionality, security and performance, as it is necessary for information to be revealed for the required computations to be performed. Different approaches to generating viable query systems have been proposed to tackle these query processing issues. One option is to decrypt data on the server or client-side before executing a query, that requires a transfer of encryption keys. Another option is query processing over encrypted data that allows computations to be performed directly over encrypted data at the server-side, avoiding transferring encryption keys or downloading data. These encryption techniques, however, may expose some information about the data as they enable particular subsets of computations. Several encryption schemes (Boldyreva et al., 2009) can be used for customized computation. Such encryption schemes are used in the CryptDB system (Popa et al., 2011), which uses onion encryption that takes an advantage of various schemes, offering security guarantees for each item of data and requires adjustment techniques (Popa et al., 2011; Aburawi et al., 2018a; Almarwani et al., 2021), as different queries may require adjustments to particular layers supporting required operations. However, major challenges remain in the use of adjustment techniques, such as revealing information, communication costs, and scalability concerns due to increasing data sizes. This paper's main aim is to highlight the communication cost associated with revealed information during the adjustment and provide ways to reduce it. To address these challenges, this paper presents novel techniques for controlling the number of values adjusted while simultaneously limiting or avoiding data updates on the database during processing adjustments. These include the techniques based on: (i) **Sorted Criteria:** To limit the number of values adjusted; (ii) **Update-Aware Adjustments:** To decrease communication costs and exposed information as well as to improve the trade-off between these factors, performance and security.

The remainder of the paper is structured as follows. In the next section, preliminaries are presented. In Section 3, issues with existing adjustment techniques are outlined and solutions proposed. Section 4 provides a case study based on the proposed approach, while Section 5 presents the outcomes of the empirical evaluation. Finally, in Section 6, an overview of related work is presented to support a summary of the conclusions in Section 7.

Almarwani, M., Konev, B. and Lisitsa, A.
Efficient and Secure Encryption Adjustment for JSON Data.
DOI: 10.5220/0010791300003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 307-313
ISBN: 978-989-758-553-1; ISSN: 2184-4356

2 PRELIMINARIES

For this paper, we assume that the data is in a NoSQL Document-Oriented Database (OrientDB). JSON data are encrypted using onion layers encryption. To execute a query over such encrypted data, an encryption adjustment may be needed. Various policies for the adjustment have been proposed in the previous research such as a Simple Encryption Adjustment (SEA) (Popa et al., 2011) for relational data model, Traversal-Aware Encryption Adjustment **(TAEA)** (Aburawi et al., 2018a) for graph data model, and Release-Aware Encryption Adjustment **(RAEA)** (Almarwani et al., 2020; Almarwani et al., 2021) for document-based databases. The rest of this section thus offers a concise outline discussing onion layers encryption and RAEA in more detail.

2.1 Onion Layers Encryption

In onion layers encryption, each data object is encrypted in "onions," or at least one "onion". Each onion is composed of layers of increasingly secure types of encryption. Each onion layer supports/enables a different set of computation operations depending on the type of encryption used. RND (Random) layer (Popa et al., 2011) provides the highest level of protection as it does not reveal any information about encrypted values, not even equality. Deterministic encryption (DET)(Popa et al., 2011) or Order-Preserving Encryption (OPE) (Boldyreva et al., 2009) allow for more computation operations (equality and comparisons, respectively) at the cost of reduced security. Each encryption type releases some information about encrypted values, and the information inclusion induces generally a *partial* order on the encryption types (Almarwani et al., 2020). For the encryption types we use in this paper, this order is $RND \preceq DET \preceq OPE \preceq PLAIN$, where *PLAIN* denotes "no encryption". For this paper, we use Ciphertext Policy Attribute-Based Encryption(CP-ABE)(Bethencourt et al., 2007) which is public-key encryption at the RND layer[1] (Almarwani et al., 2019a). The **outer layers** of onion layers encryption provide higher level of protection, but limited or none operations support (i.e. *RND*), while **inner layers** allow for more operations at the cost of reduced security (i.e. DET, OPE). These arrangements lead to the necessity of the encryption level adjustments depending on a query to be executed.

[1]CP-ABE encryption provides more functionality and can be used for attribute-based access control, which is not our concern in this paper

2.2 Conjunctive Queries

While the techniques proposed in this paper are applicable to the wider classes of queries, we limit ourselves to conjunctive queries of the form "Find all documents satisfying a conjunction of simple criteria". To unify the presentation, we use the notation adopted in (Almarwani et al., 2021) which itself follows conventions of MongoDB API. So, for a query Q we denote its conjunction of criteria by $C(Q) = \texttt{\$And}\{e_1, \ldots, e_n\}$. Each e_i is an expression of the form $f\ op\ v$, where f, f_1, f_2 are fields names and v is a value. op is one of the comparison operators from $\{<, \leq, >, \geq, =\}$. For a comparison operator op we denote by SL_{op} the minimal wrt to $\preceq$ encryption level of the arguments of op sufficient for evaluation of op on these arguments. Thus we have $SL_{=} = DET$, while $SL_{<} = SL_{\leq} = SL_{\geq} = SL_{>} = OPE$. For an atomic expression e we denote by $op(e)$ the operator occurring in e and by $field(e)$ the field name ("property")) occurring in e. For a query Q we denote by $E(Q)$ the set of all atomic expressions occurring in $C(Q)$. For a query Q and a field name f occurring in Q we define $S_Q(f) = max_{\{e \in E(Q) | field(e) = f\}} SL_{op(e)}$. For a query Q and a database instance (set of documents) I we denote by $D(Q,I)$ a set of documents returned by the execution of Q on I.

2.3 Release-Aware In-Out Encryption Adjustment

Release-Aware In-Out Encryption Adjustment (RAEA) (Almarwani et al., 2021) is a dynamic adjustment policy that combines several simple but powerful ideas. The main idea behind RAEA is to query with the conjunction of criteria: 1) sort atomic criteria according to the popularity of fields in a database instance: from low to the high count of fields; 2) proceed gradually with the execution of sub-queries by adding one atomic condition at a time; 3) in between make inward adjustments sufficient to proceed with the next sub-query; 4) once the execution of the query is completed, make outward adjustments to restore the protection levels.

3 EFFICIENCY AND SECURITY TRADE-OFFS IN RAEA

It has been shown in (Almarwani et al., 2020) that RAEA, as compared with SEA, reduces the number of decryptions/re-encryptions, leading to better security and reduced computational cost of adjustments.

There are still some noticeable issues affecting performance and security in RAEA. One is *high communication cost* incurred, as back and forth communication is required between the client and the server to update the data. It is impacted by two factors: (i) the place of the adjustment: whether it is done on the client side or server side and with varying security implications for each (see **Subsection 3.1**); and (ii) the number of two-way communications required (see **Subsection 3.2**). Secondly, RAEA has made significant progress in reducing *the amount of information disclosed* to the database, but there is still scope for improvement.

3.1 Adjustment Side

The encryption level adjustments involve data decryption/re-encryption and it can be done either on a server or a client side. The most significant differences between the two adjustment-side concerns are :(i) **Back and Forth Communication:** Client-side adjustment demands additional back and forth communication than database server-side adjustments, increasing communication costs; (ii) **Data Download:** Client-side adjustment requires data downloads, a process which may expose data to external attacks; (iii) **Transferring Encryption Keys:** Server-side adjustment involves transferring encryption keys to support adjustment, which may expose internal databases to attacks or abuse of these keys. In general, server-side performance is better than client-side performance, as the client-side communication costs are higher. In terms of security, the client side often provides better protection due to its lack of key exchange. However, this is not always the case; it is dependent on the client's security policies.

3.2 Proposed modifications of RAEA

In this paper, we propose the modification of the original RAEA policy. Sorted Criteria and Update-Aware Adjustments are used to facilitate an efficient trade-off between disclosure of information, security, and performance. The application of the modified RAEA policy happens in stages which are described below.

- **Stage A (Sorted Criteria):** Three options for sorting expressions of criteria based on the data owner or user needs for security and performance are offered. Option 1 is preferable where performance is more important than security, whereas option 2 prioritizes security above performance, and option 3 is a trade-off between the two:
  - Option 1 (Frequency Order (FO)) ($C(Q_{\leq})$): Option 1 is defined by RAEA (Almarwani et al., 2021) to mean if the frequency properties is provided by DB features, the frequency in I of all properties f occurring in $C(Q)$ can be calculated. The original query condition $C(Q)$ is then re-written by sorting the atomic expressions by *frequency*, F such that $C(Q_{\leq}) = \texttt{\$And}\{e_1,\ldots,e_n\}$, where $F(property(e_i)_I) \leq F(property(e_{i+1})_I)$, $i = 1,\ldots,n-1$.
  - Option 2 (Sensitivity Order (SO)) ($C(Q_!)$): The data owner determines the sensitivity level of all properties. The original query condition $C(Q)$ is then re-written by sorting atomic expressions according to this *Sensitivity level*, S, such that $C(Q_!) = \texttt{\$And}\{e_1,\ldots,e_n\}$, where $S(property(e_i)_I) \leq S(property(e_{i+1})_I)$, $i = 1,\ldots,n-1$. The focus is thus on the amount of information exposed that represents sensitive user data, irrespective of the number of values adjusted. *Safety* therefore takes priority here over performance.
  - Option 3 (Encryption Level Order (EO)) ($C(Q\times)$): Encryption Level Order shows the number of layers peeled away to reach a support level layer, thus reducing the number of values updated to a low-security level that disclosed more information. The original query condition, $C(Q)$, is re-written by sorting the atomic expressions according to the *Support level* , $SL_{op(e)}$, such that $C(Q\times) = \texttt{\$And}\{e_1,\ldots,e_n\}$, where $SL_{op(e_i)} \preceq SL_{op(e_{i+1})}$, $i = 1,\ldots,n-1$. Each group at the same encryption level (i.e. support level) can then be sorted according to *frequency order* to help to minimise adjustment values based on accessing the advantages of the frequency order.
- **Stage B (Criteria Encryption):** Query Q is encrypted to Q^* by encrypting the constant value in e to the level $S_Q(property(e))$.
- **Stage C (Update-Aware Adjustment):** The number of values decrypted in the inward adjustment phase is determined based on the sorting stage options. However, the outward adjustments reveal that the trade-off between security, performance, and the disclosure of information to the Database Management System (DBMS) is dependent on the number of values updated on the DBMS. As a result, three cases for updating values have been described:
  - CASE 1 (Global Update (GU)): Case 1 is proposed in original RAEA (Almarwani et al., 2021) policy; while it reduces the number of updates required as compared with SEA, it includes a higher number of updated values as compared with newly proposed CASE 2 and CASE 3 below, particularly with regard to the first property

of the sorting criteria. This is happened by updating all existing $property(e_i)$ values for documents returned during progressive execution of the query $D(Q^*_{i-1}, I_{i-2})$ to **the encryption level** $SL_{op(e_i)}$, regardless of whether or not they match the current expression. Global Updated data is generated in both the inward and outward Encryption **(IEA,OEA)** adjustment directions. This has a significant impact on the amount of information disclosed to the DB and high communication costs, thus triggering greater outward adjustment processing costs.

- CASE 2 (Necessary Update (NU)): This case involves updating only property values for those documents returned by progressively executing the query, which refers to those matching the current expression. This has a significant impact in terms of reducing the amount of information disclosed to the DB as well as minimising communication costs, resulting in reduced outward adjustment processing costs. Necessary Update is also performed in both the inward and outward Encryption adjustment directions: (A) **IEA:** The following iterative steps are executed and Let $I_0 = I$: (i) All values of $property(e_1)$ in the whole I_0 are adjusted to **the encryption support level** $SL_{op(e_1)}$ and then matched with e_1; where a match is identified, this is updated in the DB Storage, resulting in a database state I_1; (ii) The values of $property(e_2)$ occurring in documents $D(Q_1, I_0)$ are adjusted to $SL_{op(e_2)}$ and then matched with e_2 ;matches are updated in the DB Storage, resulting in database state I_2;(iii) $\cdots$; (iv) The values of $property(e_i)$ occurring in documents $D(Q_{i-1}, I_{i-2})$ are adjusted to $SL_{op(e_i)}$ and then matched with e_i; matches are updated in the DB Storage, resulting in database state I_i. (B) **OEA:** The following iterative steps are executed:(i) All values of $property(e_1)$ across I_0 are restored and updated to **the Outer Layer** in the DB Storage; matches with e_1 result in database state I_1;(ii) All values of $property(e_2)$ across I_1 are restored and updated to **the Outer Layer** in the DB Storage; matches with e_2 result in database state I_2;(iii) $\cdots$; (iv) All values of $property(e_i)$ across I_{i-1} are restored and updated to **the Outer Layer** in the DB Storage; matches with e_i result in database state I_i;(v)$\cdots$.

- CASE 3 (Zero-Update (ZU)): It is not necessary to update values in this case. We assume that the documents contain *a unique value property uv*, whose values serve as *indexes* for the documents. It means for different documents the values of *uv* are different. Then Zero-Update policy is accomplished by comparing the current expression to the documents returned. Where matches occur, instead of the property values being updated in the data storage, a copy of the index values of the matching documents is obtained. The next expression is then checked against the documents with the indexes stored, and the list of indexes is updated to refer to the documents matching the criteria up the current in the progressive execution of the query. The final list, the ultimate result of a user query, is thus a set of documents referred by this list. This case does not disclose any information to the DB server and hence minimises the communication costs. Zero-Update is only performed in the inward adjustment direction, as no re-encryption of values at the server side is required. Retrieval of the documents during ZU processing is performed by the disjunctive queries with $C(U) = \texttt{\$0r}\{u_1, \ldots, u_n\}$. Each u_i is an expression of the form $f_u = v$ where U_f is an field that contains unique index value in the collection/ table. we denote by $D(U)$ a set of documents returned by the execution of $C(U)$. (A)**IEA:** The following iterative steps are executed : (i) All values of $property(e_1)$ in the whole I are adjusted to the encryption support level $SL_{op(e_1)}$ without any updates to the DB Storage; these are then matched with e_1 and where matches occur, U values are stored, resulting in U_1;(ii) The values of $property(e_2)$ occurring in documents $D(U_1)$ are adjusted to $SL_{op(e_2)}$ without any updates to the DB Storage; these are then matched with e_2 and where matches occur, U values are used to update , resulting in U_2; (iii)$\cdots$; (iv) The values of $property(e_i)$ occurring in documents $D(U_{i-1})$ are adjusted to $SL_{op(e_i)}$ without any updates to the DB Storage; these are then matched with e_i and where matches occur, U values are used to update , resulting in U_i.

4 CASE STUDY

A case study was conducted to verify the effective capabilities of the various proposed RAEA policies. This was based on a sample dataset with 15 documents, with four properties offering a total of 52 values (see Fig. 1(a)). Query 1 was then executed using SEA and the various proposed variants of RAEA (see Fig. 1(C to L)).

FirstName="Erica" AND LastName="Murt" AND Salary>5000 AND Salary<7000 AND Department="Engineering"; Q1

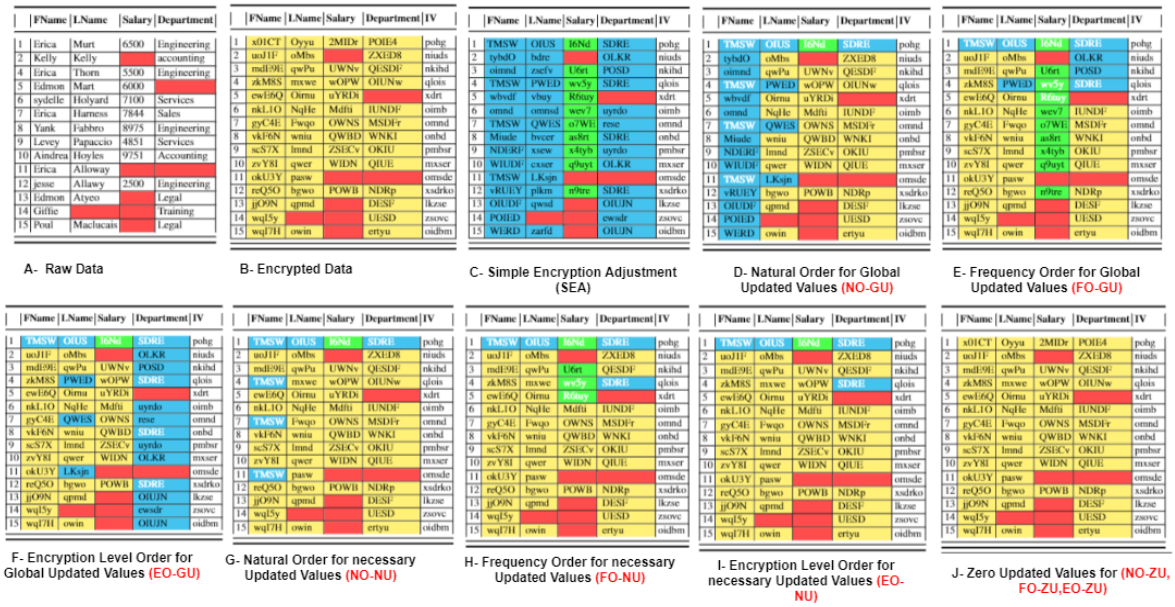


Figure 1: Case Study.

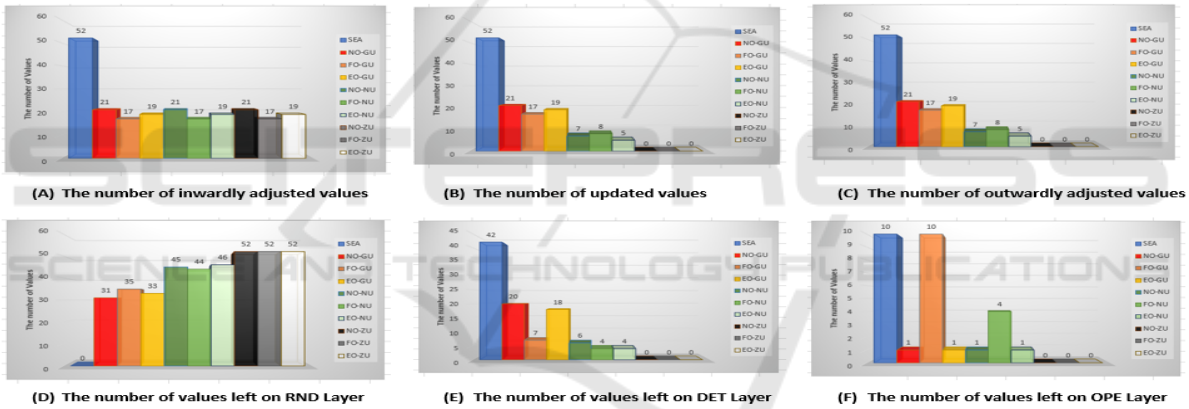


Figure 2: Analysis Results.

Table 1: Case study Results.

	SEA	GU			NU			ZU		
		NO-GU	FO-GU	EO-GU	NO-NU	FO-NU	EO-NU	NO-ZU	FO-ZU	EO-ZU
IEA	52	21	17	19	21	17	19	21	17	19
UA	52	21	17	19	7	8	5	0	0	0
OEA	52	21	17	19	7	8	5	0	0	0
RND	0	31	35	33	54	44	46	52	52	52
DET	42	20	7	18	6	4	4	0	0	0
OPE	10	1	10	1	1	4	1	0	0	0

aa

The amount of information revealed to DBMS relies on the number of RND, DET, and OPE values to be updated for all encrypted data queries, as shown in Figure 1. The case study findings are shown in Table 1. There are two main findings: *efficiency and security*. The efficiency evaluation included three components: *decryption, communication, and re-encryption costs*. *Decryption costs* depend on Inward-Adjustment. In order to lower such costs, it is necessary to decrease the number of decrypted values. The numbers of values decrypted by SEA and RAEA are shown in Figure 2(A). As with other RAEA variants, SEA requires more decrypting values. The total number of decrypted values is the same for all RAEA variants that are sorted in the same way, and the frequency order is the smallest of the three. *Communication costs* were reduced by decreasing the number of values updated. Figure 2(B) shows the greatest update values for SAE, highlighting that the update values on the GU were larger than those in NU but less than in SEA. The lowest update value, however, was 0, for ZU. *Re-encryption costs* are processed through outward adjustment. Charts B and C in Figure 2 show that the number of outward adjustment values was related to the number of values updated. *Security* was assessed using the amount of in-

formation revealed to DBMS. The cases NU and ZU were very simple and straightforward, offering only the information necessary for a query to be executed and results generated. Thus, the control of DB updates helps to maintain the RND layer values, which do not expose any information (see Fig 2(D-F)). ZU raised all RND values to 100 % , while NU is less than ZU, it increased RND more than GU on the DB. In all cases, the most obvious level of security happens in the ZU scenario.

5 EXPERIMENT & EVALUATION

The inward-adjustment time, outward-adjustment time, and communication time on both the client-side and server-side adjustments will be measured as data size increases. Data encryption was performed in three layers, in the following order: $CP-ABE \preceq DET \preceq OPE \preceq PLAIN$. The testing datasets are selected based on the number of documents: (i) 1,000 documents with 6,500 values; (ii) 2,000 documents with 13,000 values; (iii) 5,000 documents with 32,500 values; and (iv) 10,000 documents with 65,000 values. *SELECT,UPDATE,DELETE* were all evaluated using the same criteria as in Q1. It was tested 20 times. The prototype was written in Java and stored in a local database (OrientDB). The machine utilised for testing has 8 GB RAM and an i7-8565U CPU (1.99 GHz).

5.1 Performance Evaluation

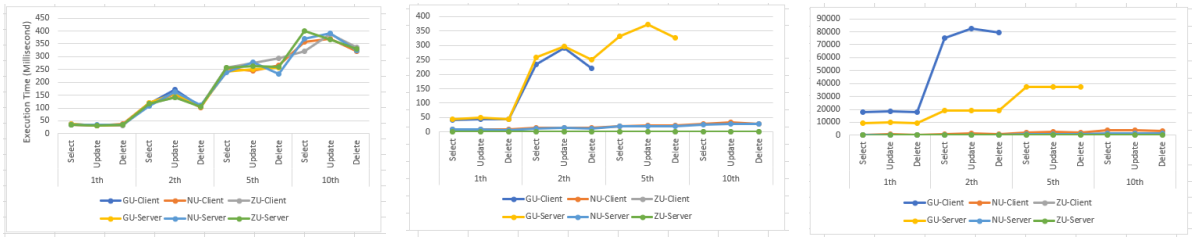
Figure 3a illustrates the time required for inward adjustment execution, which rises linearly as the number of documents increases. However, the execution time is equal for all cases. Figure 3b illustrates the Outward-Adjustment execution time, with ZU outperforming all other cases and GU taking the longest time of them. Figure 3c illustrates the time required to communicate with and retrieve data from a database. GU communicated slowly and was the least effective of the three cases, while the ZU outward adjustment outperformed the other two. Overall, in all three cases, server-side adjustment outperforms client-side adjustment.

6 RELATED WORK

The majority of research has focused on unmodified-based DBMS for querying encrypted data. (Popa et al., 2011; Waage and Wiese, 2017; Aburawi et al., 2018b; Almarwani et al., 2019b) encrypt data using multiple layers of encryption; thus, they need adjustment techniques that enable more computing classes. In a previous study, three types of adjustment techniques were identified: SEA (Popa et al., 2011), TAEA (Aburawi et al., 2018a), and RAEA(Almarwani et al., 2020; Almarwani et al., 2021). SEA performs adjustments prior to query execution by adjusting all values of columns occurring in query criteria to the encryption level based on the class of computation contained in the expression column. CryptDB was the first to offer an adjustment technique (SEA) for SQL, whereas CryptGraph (Aburawi et al., 2018b) and SDDB (Almarwani et al., 2019b; Almarwani et al., 2019a), respectively, apply the CryptDB transfer concept to graph databases (Neo4j database) and document databases (MongoDB). As SEA discloses more information than necessary, Aburawi et al. (Aburawi et al., 2018a) proposed TAEA for traversal queries for graph databases. TAEA processes adjustments during query execution to limit the quantity of information required as much as possible; this execution happens dynamically based on node-to-node relationships. However, the Document-Database lacks document-to-document relationships, thus, TAEA is useless in such scenarios. Almarwani et al (Almarwani et al., 2019b; Almarwani et al., 2019a) suggested RAEA for conjunctive conditions, as unlike TAEA, RAEA is concerned with limiting the amount of information disclosed after query conduction, a process based on restoring maximum security levels. TAEA and RAEA both offer better protection than SEA, but they create additional obstacles, such as communication costs or disruptions to adjustments in some cases; determining how to handle these while enabling an appropriate trade-off between security, effectiveness, and amount of revealed data is thus important.

7 CONCLUSION

This paper presented an overview of the RAEA adjustment techniques used to query encrypted data using Onion Layers Encryption. The main goal was to identify the approach that best met the following requirements: the approach had to (i) permit adjustment at a lower cost of decryption/encryption; (ii) permit adjustment with limited data updates; (iii) reduce communication costs; (iv) provide better security by revealing only the required information to the DBMS for each query; and (v) provide better performance concerning query execution speed. This paper thus presented both Sorted Criteria and Update-



(a) Execution Time of IEA (b) Execution Time of OEA (c) Execution Time of compunction

Figure 3: Execution Time of Queries.

Aware Adjustments to RAEA. The sorting approach provides three options, depending on the priorities given to performance and security, facilitating a trade-off between the two. Update-Aware Adjustments, on the other hand, give three cases based on Outward Adjustment costs and the predetermined trade-off between three requirements of communication cost, performance, and security. In the document-database experiment, the improved Update-Aware Adjustments yielded lower communication costs and increased security. More experiments on these Sorted Criteria and Update-Aware Adjustments will be conducted in future work to assess their applicability for various database types, such as SQL, and to assess their efficacy with regard to big data more formally.

REFERENCES

Aburawi, N., Coenen, F., and Lisitsa, A. (2018a). Traversal-aware encryption adjustment for graph databases. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications*. SCITEPRESS-Science and Technology Publications.

Aburawi, N., Lisitsa, A., and Coenen, F. (2018b). Querying encrypted graph databases. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SCITEPRESS-Science and Technology Publications.

Almarwani, M., Konev, B., and Lisitsa, A. (2019a). Fine-grained access control for querying over encrypted document-oriented database. In *International Conference on Information Systems Security and Privacy*, pages 403–425. Springer.

Almarwani, M., Konev, B., and Lisitsa, A. (2019b). Flexible access control and confidentiality over encrypted data for document-based database. In *ICISSP*, pages 606–614.

Almarwani, M., Konev, B., and Lisitsa, A. (2020). Release-aware encryption adjustment query processing for document database. In *Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*, pages 48–51.

Almarwani, M., Konev, B., and Lisitsa, A. (2021). Release-aware in-out encryption adjustment in mongodb query processing. In *ICISSP*, pages 714–722.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE.

Boldyreva, A., Chenette, N., Lee, Y., and O'neill, A. (2009). Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 224–241. Springer.

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

Waage, T. and Wiese, L. (2017). Property preserving encryption in nosql wide column stores. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 3–21. Springer.