Extension to CryptDB with Support for Arithmetic Expressions

Karthik Jagilinki and Ray Kresman

Abstract Growth of cloud infrastructure has made it attractive for enterprises to rethink how and where to position their resources. CryptDB (Raluca Ada Popa, et al.: CryptDB: Protecting Confidentiality with Encrypted Query Processing. MIT CSAIL, SOSP '11, October 23–26, 2011) allows storage of cloud data in encrypted form and helps mitigate privacy concerns. It performs encryption of data in layers without revealing plain-text data to the cloud vendor. While CryptDB supports simple queries, it does not appear to handle queries with arithmetic expressions. We discuss the pros and cons of a couple of schemes to address this issue and finally propose a component-based approach that provides support for queries with arithmetic expressions.

Keywords Encryption · Cloud · Service provider · Onion encryption

1 Introduction

With the advent of cloud computing, variety of services are offered by several cloud vendors in terms of infrastructure, platform, and software [1, 2]. This frees enterprises from the job of maintaining servers and associated resources. Applications routinely use databases and several new applications are emerging with databases as backend. Database as service is getting popular with service providers; they provide services to store, manage, maintain, and administer large amounts of business data in cloud databases [3, 4]. This comes with additional advantages such as availability, scalability, and usage-based cost models [5]. High availability can be ensured through replication in geographically distributed locations [6]. SQL Azure cloud database service, for example, allows partitioning of databases either horizontally or vertically using their in-house elastic tools. These and other features from service providers

K. Jagilinki · R. Kresman (✉)
Department of Computer Science, Bowling Green State University, Bowling Green, OH, USA
e-mail: Kresman@bgsu.edu

K. Jagilinki
e-mail: Jagilinki@bgsu.edu

H. Kim and K. J. Kim (eds.), *IT Convergence and Security*, Lecture Notes in Electrical Engineering 782, https://doi.org/10.1007/978-981-16-4118-3_10

make cloud databases more attractive and help entice customers to move even critical data to the cloud.

However, for customers to move their critical data to cloud, service providers must address the privacy concerns of their clients. Security of their sensitive data is a critical factor for customers in deciding whether to outsource data to these untrusted third-party cloud vendors. One approach is to store the data in encrypted form and not provide the encryption key to the cloud provider. The idea is to encrypt the data on client side before migration to the service provider and store the data in encrypted form on the untrusted server.

CryptDB employs such an approach [7, 8]. It allows storage of data in encrypted form on the cloud/untrusted server. Queries are performed on the encrypted data without fully decrypting the data on the server. It works by performing SQL-aware encryption schemes in layers of onion to encrypt the data. And the data is decrypted only on the client side with the server having no access to the key. This makes CryptDB a practical solution for querying encrypted data. While it supports simple queries, the version that we had access to could not handle queries with arithmetic expressions. This paper describes a generic extension to the CryptDB framework to address this issue.

2 Encrypted Cloud Storage

CryptDB [7] follows a novel approach in processing encrypted data. Their approach suggests use of (SQL) operator-specific encryption schemes. These SQL-aware encryption schemes are then used to encrypt a single data value in layers. It employs multiple encryption schemes stacked into layers (or onions) that are chosen based on the operators in SQL queries. This approach allows query processing on the server side without fully decrypting the cipher text. It eliminates the false positives that are common with other approaches [9]. In terms of efficiently executing queries over encrypted data, CryptDB has low overhead on the client side compared to other approaches. Use of onions of encryption appears to be novel and practical and provides adequate confidentiality.

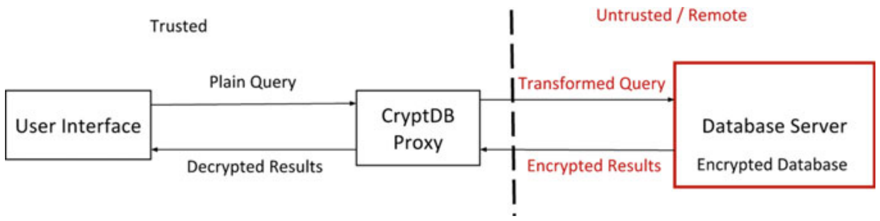


Fig. 1 CryptDB architecture

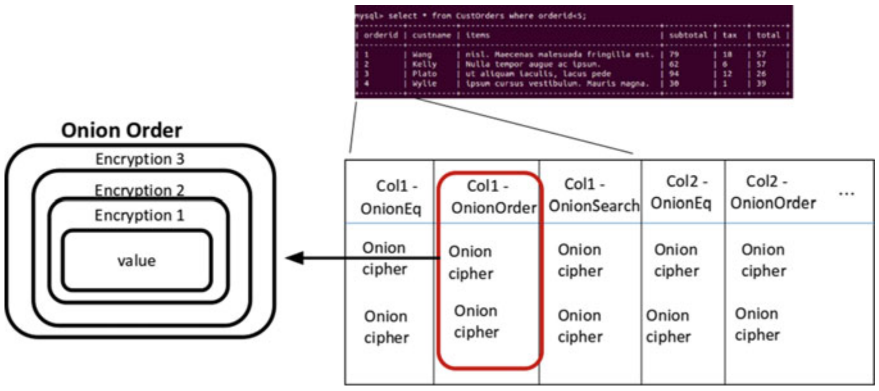
The model is depicted in Fig. 1, with a proxy between user interface and database server. The proxy is located on the client side and is trusted, while the database server is on the remote server in an untrusted location. Therefore, the data is always stored in encrypted form on the database server. The proxy located on the client side is lightweight and does not store any data. The only information that the proxy stores is schema and the master key.

Whenever the user issues a plaintext query from the interface to the database server, CryptDB proxy intercepts the query, analyzes it and then anonymizes the sensitive information in the query by performing certain encryption based on the operators in the query. The proxy performs two steps. First, it transforms the query by rewriting it with the anonymized names of columns and tables, and then sends it to the database server. Second, it strips off a layer of onion on the database server based on the intercepted query by invoking a user defined function. This way, based on the query, the proxy transforms the query and keeps the data on the server at the same level. The server processes the transformed query on the encrypted database and then returns the encrypted results back to the proxy. Note that the server works in the encrypted domain and cannot decrypt the underlying data. The proxy then decrypts them and sends the query result, in plaintext, to the user. The proxy does not perform any query execution, instead the query is fully executed on the database server. CryptDB utilizes server-side processing more and has less client-side overhead.

2.1 Encryption and Storage at the Backend

CryptDB has various encryption schemes based on the operators used in SQL queries. Figure 2 outlines the data encryption process on the database server.

As shown in Fig. 2, each column data item is encrypted using a different set of encryption schemes. Also, for the sample query, we see that a column orderID is



```
ysql> select * from CustOrders where orderid<5;
+---------+----------+------------------------------------------+----------+-----+-------+
| orderid | custname | items                                    | subtotal | tax | total |
+---------+----------+------------------------------------------+----------+-----+-------+
| 1       | Wang     | nisl. Maecenas malesuada fringilla est.  | 79       | 18  | 57    |
| 2       | Kelly    | Nulla tempor augue ac ipsum.             | 62       | 6   | 57    |
| 3       | Plato    | ut aliquam iaculis, lacus pede           | 94       | 12  | 26    |
| 4       | Wylie    | ipsum cursus vestibulum. Mauris magna.   | 30       | 1   | 39    |
+---------+----------+------------------------------------------+----------+-----+-------+
```

Fig. 2 Query processing

encrypted and stored in three different sets i.e. Col1-OnionEq, Col1-OnionOrder, and Col1-OnionSearch. Based on the operators present in the query, any of the encrypted dataset can be used to process the query. OnionEq, OnionOrder, and OnionSearch as the names suggest are used to perform query processing based on the different (SQL) operators equal, order, and search respectively. For example, in Fig. 2, the user issued the query: select * from CustOrders where orderID < 5. Here, we have an order operator < . Once the proxy sees this operator in the query, it will invoke the user defined function to map the user query to the OnionOrder encrypted dataset. As noted in Fig. 2, each of these data sets consists of an onion cipher that had encrypted the column value using multiple encryption schemes in layers i.e. Encryption 1, Encryption 2, and Encryption 3. Each onion cipher consists of data value being encrypted with these multiple encryption schemes in layers. Here, the functionality of lower layer is strictly higher than the functionality of higher layers to help ensure overall security of data.

2.2 Query Processing in the Encrypted Domain

Whenever the client (or the interface) issues a query, the proxy intercepts it and transforms the query based on the type of operators it has. For instance, if the query has equality operator then Deterministic encryption scheme is used. And if the query has order operator to check less than or greater than, the Order-Preserving scheme will be used to perform encryption to the query.

Invoke UDF to adjust the level of onion. Different queries coming from client may perform different computation on server data. Based on the user query, and proxy invokes a user defined function (UDF) to adjust the database encryption level on the backend. Whenever certain UDF is invoked based on the user query, proxy would give the key to the SQL function on database server for the specific layer of the onion. This would simply strip off certain layer of encryption. This way proxy processes queries over encrypted data without giving master key to the server. Thus, as noted earlier, the data on the server is always in encrypted form.

3 Proposed Framework

CryptDB does not appear to support queries with arithmetic expressions. This is because expressions are encrypted with different encryption schemes which may not be compatible with one another. To illustrate this problem, let us invoke a query with an arithmetic expression. The query and the error message/CryptDB response is this: *mysql> select subtotal + tax from CustOrders where total < 20;* **ERROR 1105 (0700): Current crypto schemes do not support this query**

The query, *Select subtotal + tax from CustOrders where total < 20* has an arithmetic expression *subtotal + tax*. Here subtotal and tax are two different columns. The data of these columns are stored in the backend encrypted at their highest secured level. When the user gives this query, CryptDB proxy intercepts it and sees that the query has an arithmetic expression, which is trying to perform an addition operation. Initially these columns are encrypted at layer 3. As addition operation is not possible at Encryption layer 3, proxy gives the partial key of layer 3 to the backend to decrypt a layer of onion for column subtotal from Encryption layer 3 to Encryption layer 2. At this layer 2, CryptDB uses homomorphic encryption scheme that is suitable to perform addition operation over encrypted data. However, the problem here is CryptDB proxy only decrypts the first column subtotal to Encryption layer 2, but it still keeps column tax at Encryption layer 3. Clearly, addition operation is incompatible between the two columns that belong to different encryption schemes. Our extension to correctly handle such queries is discussed next.

3.1 Tweak CryptDB UDFs

Given the complexity of systems such as CryptDB, we attempt to address the question of resolving queries with arithmetic expressions [10, 11] using a component-based approach. Our goal is to add a component that would interface with the underlying software architecture of CryptDB and yet respect its (CryptDBs) black-box nature.

One approach is to update CryptDB UDF to support queries with arithmetic expressions. When the UDF is triggered, if we could modify the UDF in a way to decrypt multiple columns that are present in the query at the same time, then that would help keep these columns on the server at the same layer allowing one to perform a valid operation, such as addition, over encrypted data. A second approach is to include additional encryption schemes to each of the onions, to help provide support for queries with arithmetic expressions. Unfortunately, neither approach respects the black-box nature of CryptDB components and so we set out to explore alternate mechanisms.

3.2 Piped Architecture

A third possibility is the piped architecture as shown in Fig. 3. The vertical bar means pipes between adjacent subsystems. The Pre-Intercept stage provides input to CryptDB and the Post-Intercept receives the results returned from CryptDB. The Pre-Intercept receives the user's query and does some pre-processing on it before

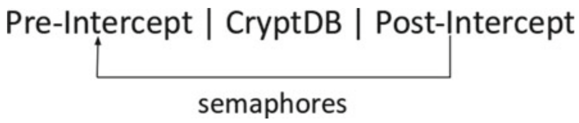


Fig. 3 Piped architecture

shipping it to CryptDB. If the query has no arithmetic expression, it is passed as it is.

However, for a query with arithmetic expressions, a simple approach is to split the user's query into multiple queries based on the arithmetic expression. Then, the queries are issued, one-by-one, to CryptDB. Post-Intercept can then combine the results of each of these split query results returned by CryptDB and generate the composite result (or response) to the original user's query. As shown in Fig. 3, we need some coordination between the Pre-Intercept and Post-Intercept subsystems. Semaphores can be used to synchronize such communication between the two stages and help ensure that the user's query is processed in a sequential manner. The advantage of the piped architecture is that it respects the black-box nature of CryptDB components.

3.3 Integrated Architecture

The piped architecture can be modified slightly to integrate the functionality of Pre-Intercept and Post-Intercept stages in a seamless manner (into one component) to process arithmetic expressions in the query. We call this component-based approach, 'Integrated Architecture.'

The proposed system will coordinate with CryptDB in an integrated manner to process queries with arithmetic expressions. As noted earlier, it also respects the black-box nature of CryptDB components.

Figure 4 is a high-level schematic of our Integrated Architecture. Whenever the user enters queries from the interface, they flow through EnhancedCryptDB to the CryptDB proxy. The middle step does some pre-processing on the query before giving to CryptDB proxy. As noted in Sect. 2, CryptDB follows a sequence of steps to interact with backend database server maintained by the cloud provider and finally returns the query response to EnhancedCryptDB. Then, the latter component does post-processing on these results before sending the results back to the user.

EnhancedCryptDB consists of two major pieces: Query Interceptor and Query Processor. Query Interceptor acts as a query manager that handles queries from the client (or interface). Query Processor provides the primary functionality in our approach, i.e. it receives the queries from the Query Interceptor and splits the ones with arithmetic expressions. Figure 5 shows the components of our architecture to illustrate the information flow in our framework.

Query Interceptor and Query Processor are two different processes that coordinate with each other in processing the user's query. As these are two different processes, we require interprocess communication between them. Query Interceptor reads user queries from the interface and handles them one at a time. For each query, it internally calls Query Processor to handle arithmetic expressions in the query. Query Processor then rewrites the query, as needed, before sending them to the CryptDB proxy. CryptDB proxy performs its functionality (see Sect. 2.1) and returns the results that are intercepted by Query Processor. Note that these results are in plaintext. Query

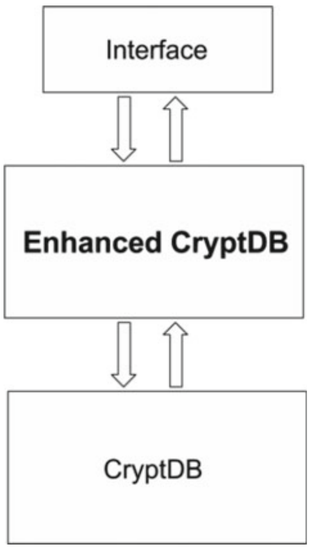


Fig. 4 Integrated architecture

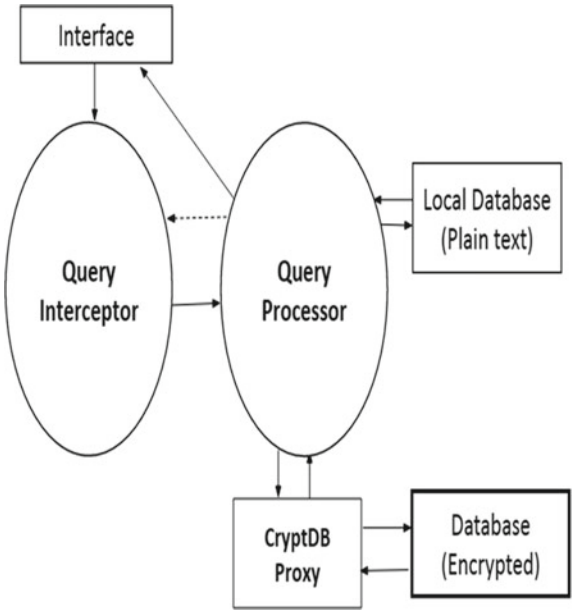


Fig. 5 EnhancedCryptDB srchitecture

Processor dumps these result tuples to a local database; it then queries this (local) database against the original user expression. The output is just the response to the original user's query and this output is returned to the user.

Our approach can handle any type of arithmetic expression in the query. Consider for example: select a + b / c from T1. Such a query will result in 3 calls to CryptDB: select a from T1; select b from T1; select c from T1. CryptDB response to these three queries is stored in a local database, say table localTable. At the end, the Query Processor does a query of the form, select a + b / c from localTable that yields the result to the original query, which is then returned to the user.

Clearly, one downside with our approach is that it is not as efficient compared to a scheme that handles the entire query in one indivisible unit. However, our approach is component-based in the sense that it integrates with CryptDB and at the same time respects the black box nature of the (CryptDB) software components. We feel that this trade-off between performance and simplicity is something for the client to consider. A second issue is security; since the local database handles data in plaintext, does it open any security holes? This work was done as a Master's project and we have not done any analysis on data leakage in our system, but we feel—assuming the proposed system sits next to CryptDB (perhaps in the same hardware as CryptDB)—that its security is perhaps comparable to CryptDB [12, 13]. In any event, these two issues are worthy of additional investigations.

4 Concluding Remarks

CryptDB lets users store cloud data in encrypted form. It supports several layers of encryption. Clients can issue SQL commands against the data; Queries are performed directly on the encrypted data and the cloud provider does not have access to the decryption key.

This paper proposed an extension framework to CryptDB to provide support for arithmetic expressions in SQL queries. The advantage of our approach is that it respects the black-box nature of CryptDB components - the proposed scheme does not affect, or is not even aware of, the internal details of various CryptDB onion layers. For brevity, software implementation details and performance evaluation results are omitted in our discussions. These details will be addressed in a future paper.

Acknowledgements We thank the reviewers for their insight into an earlier draft.

References

1. RDS (2016) Amazon relational database service (RDS) – AWS. Amazon Web Services, 2016. https://aws.amazon.com/rds/. Accessed 4 Aug 2016

2. Salesforce (2016) What is cloud computing technology? Cloud definition. Salesforce.com, 2000. http://www.salesforce.com/cloudcomputing/. Accessed 4 Aug 2016
3. Google Developers (2016) Cloud SQL—MySQL relational database. Google Developers. https://cloud.google.com/sql/. Accessed 4 Aug 2016
4. Oracle (2016) Database. https://cloud.oracle.com/database. Accessed 4 Aug 2016
5. Wikipedia (2016) SQL Azure. Wikimedia Foundation, 2016. https://en.wikipedia.org/wiki/SQL_Azure. Accessed 4 Aug 2016
6. MS Azure (2016) Scaling out with azure SQL database. 2016. https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-introduction/. Accessed 4 Aug 2018
7. Popa RA et al (2011) CryptDB: protecting confidentiality with encrypted query processing. MIT CSAIL, SOSP '11
8. CryptDB (2016) In: CryptDB. https://css.csail.mit.edu/cryptdb/#Software. Accessed 4 Aug 2016
9. Alwarsh M, Kresman R (2011) On querying encrypted databases. In: Proceedings of the 10th international conference on security and management, pp 256–262
10. Jagilinki K (2016) Enhanced query processing with CryptDB. Master's project. Department of Computer Science, Bowling Green State University, Bowling Green, p 73
11. Jagilinki K, Kresman R (2020) An extension to CryptDB. IAET international conference on artificial intelligence, information systems, engineering, Budapest, Hungary. 2:1
12. Naveed M, Kamara S, Wright C (2015) Inference attacks on property-preserving encrypted databases. In: CCS '15: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pp 644–655. https://doi.org/10.1145/2810103.2813651
13. Almarwani M, Konev B, Lisitsa A (2019) Flexible access control and confidentiality over encrypted data for document-based database. In: Proceedings of the 5th international conference on information systems security and privacy - volume 1: ICISSP, pp 606–614. ISBN 978-989-758-359-9. https://doi.org/10.5220/0007582506060614