A Survey of Security Aspects of CryptDB in Cloud

M.Sathya Devi, Assistant Professor, CVR College of Engineering (A), Hyderabad,
K.Srinivasa Chakravarthy, Assistant Professor, Vasavi College of Engineering (A), Hyderabad,

Abstract:*Data being stored in the cloud is susceptible to intruders, where one of the intruders can be the database administrator. In order to protect the critical data stored in the cloud from the database administrator, CryptDB can be used. CryptDB is used for Relational Cloud. Relational Cloud, which is referred to as 'Database as a Service', is provided by the cloud providers. Relational Cloud makes use of encryption mechanisms to encrypt the data, to store the data and decrypt the data to retrieve it.*

Introduction

Relational Cloud makes use of DBMS servers at the back end for query processing and storage nodes.

Relational Cloud provides 'relational database as a service'. In this regard the main aspects needed for relational cloud are security, multi-tenancy which are basis for cloud. This paper mainly deals with security in relational cloud. The security has to be provided mainly from the cloud administrator who can view the data stored in the cloud. The end users for the relational cloud would be the database users who want to store the data in the relational database stored in the cloud.

It is the privacy of the user which has to be handled ultimately.The security can be provided when all the data stored in the cloud is encrypted. If the data is encrypted, then how can the queries be executed on the encrypted data? CryptDB provides a set of techniques using which the SQL queries be executed on encrypted data. CryptDB supports various encryption techniques which can be applied on various types of data. Depending on the queries run by the user, these algorithms would vary.

Providing Security in cloud through Cryptdb

[4]Cloud applications are vulnerable to theft of sensitive information because adversaries can exploit software bugs to gain access to private data, and because curious or malicious administrators may capture and leak data. CryptDB is a system that provides practical and provable confidentiality in the face of these attacks for applications backed by SQL databases. It works by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. CryptDB can also chain encryption keys to user passwords, so that a data item can be decrypted only by using the password of one of the users with access to that data. As a result,a database administrator never gets access to decrypted data, and even if all servers are compromised, an adversary cannot decrypt the data of any user who is not logged in.

Theft of private information is a significant problem, particularly for online applications. An adversary can exploit software vulnerabilities to gain unauthorized access to servers; curious or malicious administrators at a hosting or application provider can snoop on private data ; and attackers with physical access to servers can access all data on disk and in memory.

One approach to reduce the damage caused by server compromises is to encrypt sensitive data, as in SUNDR, SPORC andDepot, and run all computations (application logic) on clients. Unfortunately, several important applications do not lend themselves to this approach, including database-backed web sites that process queries to generate data for the user, and applications that compute over large amounts of data. Even when this approach is tenable, converting an existing server-side application to this form can be difficult. Another approach would be to consider theoretical solutions such as fully homomorphic encryption , which allows servers to compute arbitrary functions over encrypted data, while only clients see decrypted data. However, fully homomorphic encryption schemes are still prohibitively expensive by orders of magnitude.

CryptDB works by intercepting all SQL queries in a database proxy, which rewrites queries to execute on encrypted data (CryptDB assumes that all queries go through the proxy). The proxy encrypts and decrypts all data, and changes some query operators, while preserving the semantics of the query. The DBMS server never receives decryption keys to the plaintext so it never sees sensitive data, ensuring that a curious DBA cannot gain access to private information (threat 1). To guard against application, proxy, and DBMS server compromises(threat 2), developers annotate their SQL schema to define different principals, whose keys will allow decrypting different parts of the database. They also make a small change to their applicationsto provide encryption keys to the proxy, as described in §4. The proxy determines what parts of the database should be encrypted under what key. The result is that CryptDB guarantees the confidentiality of data belonging to users that are not logged in during acompromise (e.g., user 2 in Figure 1), and who do not log in until the compromise is detected and fixed by the administrator.Although CryptDB protects data confidentiality, it does not ensurethe integrity, freshness, or completeness of results returned to the application. An adversary that compromises the application, proxy,or DBMS server, or a malicious DBA, can delete any or all of thedata stored in the database. Similarly, attacks on user machines,such as cross-site scripting, are outside of the scope of CryptDB. Two threat models addressed by CryptDB are described, and the security guarantees provided under those threat models.

Threat 1: DBMS Server Compromise

In this threat, CryptDB guards against a curious DBA or other external attacker with full access to the data stored in the DBMS server. Our goal is confidentiality (data secrecy), not integrity or availability. The attacker is assumed to be passive: she wants to learn confidential data, but does not change queries issued by the application, query results, or the data in the DBMS. This threat includes DBMS software compromises, root access to DBMS machines, and even access to the RAM of physical machines. With the rise in database consolidation inside enterprise data centres, outsourcing of databases to public cloud computing infrastructures, and the use of third-partyDBAs, this threat is increasingly important.

Approach

CryptDB aims to protect data confidentiality against this threat by executing SQL queries over encrypted data on the DBMS server. The proxy uses secret keys to encrypt all data inserted or included in queries issued to the DBMS. Our approach is to allow the DBMS server to perform query processing on encrypted data as it would on an unencrypted database, by enabling it to compute certain functions over the data items based on encrypted data. For example, if the DBMS needs to perform a GROUP BY on column c,the DBMS server should be able to determine which items in that column are equal to each other, but not the actual content of each item. Therefore, the proxy needs to enable the DBMS server to determine relationships among data necessary to process a query. By using SQL-aware encryption that adjusts dynamically to the queries presented, CryptDB is careful about what relations it reveals between tuples to the server. For instance, if the DBMS needs to perform only a GROUP BY on a column c, the DBMS server should not know the order of the items in column c, nor should it know any other information about other columns. If the DBMS is required to perform an ORDER BY, or to find the MAX or MIN, CryptDB revealsthe order of items in that column, but not otherwise.

Guarantees

CryptDB provides confidentiality for data content and for names of columns and tables; CryptDB does not hide the overall table structure, the number of rows, the types of columns,or the approximate size of data in bytes. The security of CryptDBis not perfect: CryptDB reveals to the DBMS server relationships among data items that correspond to the classes of computation that queries perform on the database, such as comparing items for equality, sorting, or performing word search. The granularity at which CryptDB allows the DBMS to perform a class of computations is an entire column (or a group of joined columns, for joins), whichmeans that even if a query requires equality checks for a few rows, executing that query on the server would require revealing that class of computation for an entire column. The classes of computation map to CryptDB's encryption schemes, andthe information they reveal.More intuitively, CryptDB provides the following properties:

- Sensitive data is never available in plaintext at the DBMS server.
- The information revealed to the DBMS server depends on the classes of computation required

by the application's queries,subject to constraints specified by the application developer in the schema

1. If the application requests no relational predicate filtering on a column, nothing about the data content leaks (otherthan its size in bytes).
2. If the application requests equality checks on a column,CryptDB's proxy reveals which items repeat in that column(the histogram), but not the actual values.
3. If the application requests order checks on a column, the proxy reveals the order of the elements in the column.

- The DBMS server cannot compute the (encrypted) results for queries that involve computation classes not requested by the application.

How close is CryptDB to "optimal" security? Fundamentally, optimal security is achieved by recent work in theoretical cryptography enabling any computation over encrypted data however, such proposals are prohibitively impractical. In contrast, CryptDB is practical, it also provides significant security in practice. Specifically, we show that all or almost all of the most sensitive fields in the tested applications remain encrypted with highly secure encryption schemes. For such fields, CryptDB provides optimal security, assuming their value is independent of the pattern in which they are accessed (which is the case for medical information, social security numbers, etc). CryptDB is not optimal for fields requiring more revealing encryption schemes, but we find that most such fields are semi-sensitive (such as timestamps).

Finally, a passive attack model is realistic because malicious DBAs are more likely to read the data, which may be hard to detect, than to change the data or query results, which is more likely to be discovered.

Threat 2: Arbitrary Threats

The second threat is that where the application server,proxy, and DBMS server infrastructures may be compromised arbitrarily.The approach in threat 1 is insufficient because an adversarycan now get access to the keys used to encrypt the entire database.The solution is to encrypt different data items (e.g., data belongingto different users) with different keys. To determine the key that should be used for each data item, developers annotate the application's database schema to express finer-grained confidentiality policies. A curious DBA still cannot obtain private data by snooping on the DBMS server (threat 1), and in addition, an adversary who compromises the application server or the proxy can now decrypt only data of currently logged-in users (which are stored in the proxy).

Data of currently inactive users would be encrypted with keys not available to the adversary, and would remain confidential.In this configuration, CryptDB provides strong guarantees inthe face of arbitrary server-side compromises, including those that gain root access to the application or the proxy. CryptDB leaks at most the data of currently active users for the duration of the compromise, even if the proxy behaves in a Byzantine fashion. By"duration of a compromise", mean the interval from the start of the compromise until any trace of the compromise has been erased from the system. For a read SQL injection attack, the duration of the compromise spans the attacker's SQL queries.

QUERIES OVER ENCRYPTEDDATA

This section describes how CryptDB executes SQL queries over encrypted data. The threat model in this section is threat 1 from The DBMS machines and administrators are not trusted, but the application and the proxy are trusted.

CryptDB enables the DBMS server to execute SQL queries on encrypted data almost as if it were executing the same queries on plaintext data. Existing applications do not need to be changed. The DBMS's query plan for an encrypted query is typically the same as for the original query, except that the operators comprising the query, such asselections, projections, joins, aggregates, and orderings, are performed on ciphertexts, and use modified operators in some cases.

CryptDB's proxy stores a secret master key MK, the database schema, and the current encryption layers of all columns.

Processing a query in CryptDB involves four steps:

1. The application issues a query, which the proxy intercepts andrewrites: it anonymizes each table and column name, and, using the master key MK, encrypts each constant in the query with an encryption scheme best suited for the desired operation.

2. The proxy checks if the DBMS server should be given keys to adjust encryption layers before executing the query, and if so,issues an UPDATE query at the DBMS server that invokes a UDF to adjust the encryption layer of the appropriate columns.
3.The proxy forwards the encrypted query to the DBMS server,which executes it using standard SQL (occasionally invoking UDFs for aggregation or keyword search).
4. The DBMS server returns the (encrypted) query result, which the proxy decrypts and returns to the application.

SQL-aware Encryption

The encryption types that CryptDB uses, including a number of existing cryptosystems, an optimization of a recent scheme, and a new cryptographic primitive for joins. For each encryption type, the security property that CryptDB requires from it, its functionality, and how it is implemented has been explained.

Random (RND)

RND provides the maximum security in CryptD indistinguishability under an adaptive chosen-plaintext attack (IND-CPA); the scheme is probabilistic, meaning that two equal values are mapped to different ciphertexts with overwhelming probability. On the other hand, RND does not allow any computation to be performed efficiently on the ciphertext. An efficient construction of RND is to use a block cipher like AES or Blowfish in CBC mode together with a random initialization vector (IV). (mostly use AES, except for integer values, where Blowfish is used for its 64-bit block size because the 128-bit block size of AES would cause the ciphertext to be significantly longer).
Since, in this threat model, CryptDB assumes the server does not change results, CryptDB does not require a stronger IND-CCA2 construction (which would be secure under a chosen-ciphertext attack). However, it would be straightforward to use an IND-CCA2- secure implementation of RND instead, such as a block cipher in UFE mode, if needed.

Deterministic (DET)

DET has a slightly weaker guarantee, yet it still provides strong security: it leaks only which encrypted values correspond to the same data value, by deterministically generating the same ciphertext for the same plaintext. This encryption layer allows the server to perform equality checks, which means it can perform selects with equality predicates, equality joins, GROUP BY,COUNT, DISTINCT, etc.
In cryptographic terms, DET should be a pseudo-random permutation(PRP) .For 64-bit and 128-bit values, a block is used cipher with a matching block size (Blowfish and AES respectively); make the usual assumption that the AES and Blowfish blockciphers are PRPs. pad smaller values out to 64 bits, but for data that is longer than a single 128-bit AES block, the standard CBC mode of operation leaks prefix equality (e.g., if two data itemshave an identical prefix that is at least 128 bits long). To avoid this problem, AES with a variant of the CMC mode is used, which can be approximately thought of as one round of CBC, followed by another round of CBC with the blocks in the reverse order. Since the goal of DET is to reveal equality, a zero IV is used for our AES-CMC implementation of DET.

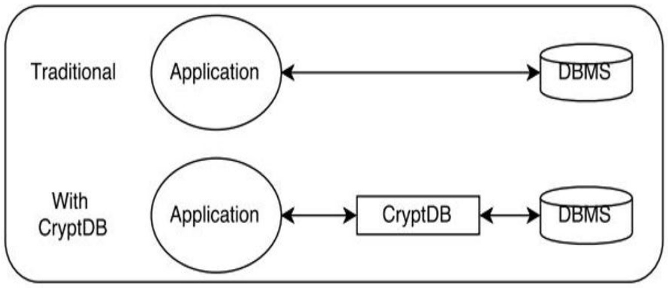
System Architecture [1]

CRYPT DB

Crypt DB is a weaker attacker model. Crypt DB is an implementation that allows query processing over encrypted databases.

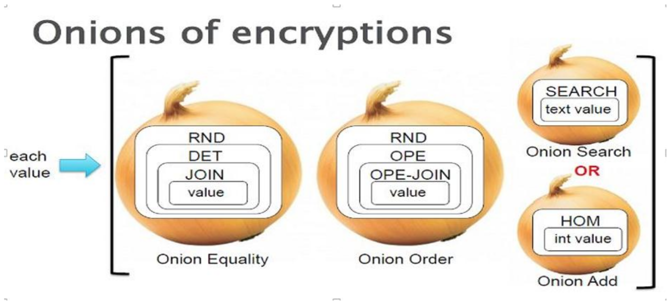
The database is managed by the cloud provider, but database items are encrypted with keys that are only known by the data owner.

Assumes trusted cloud based server and proxy. Uses the encryption schemes for comparisons made on ciphers.



CRYPTDBARCHITECTURE

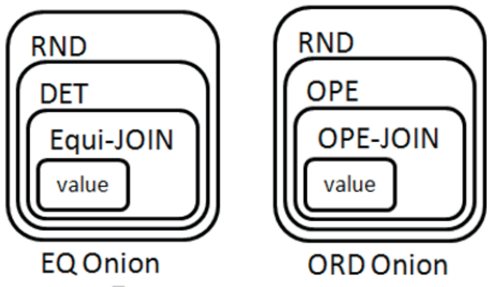
- Crypt DB is a layered architecture; each has 4 layers (onions).
- <u>Layered encryption:</u> The encryption of a data in the database is computed in a layered way.



EQANDORDONION

- There are four different main goals to achieve, and for each goal there exists a different layered particle, which is called as onion: EQ, ORD, SEARCH and ADD onion.

- EQ onion aims to adjust layers for equality queries, while ORD onion aims to adjust the order leakage for the queries including comparison.



SEARCH AND ADD ONION

SEARCH onion is used to search a text in the database without leaking any information. This onion is not allowed to execute integer values. Finally, ADD onion aims to add encrypted values which only support integer values. These onions have different layers each encrypted by using different algorithms.

Encryption Schemes

Scheme	Construction	Function	
RND	AES in CBS	NONE	
HOM	Paillier	+	
SEARCH	Song's algorithm	Word search	

CONCLUSION

Once the cloud environment is established master container handles all the requests from worker containers and deploys applications on worker machine. Master Container is associated with database in which data is stored in encrypted format .If the user wants to retrieve data he is provided with a decryption password every time for a particular session. Every session details are saved in Master Container and loaded when user logs in.

REFERENCES

1. Raluca Ada Popa, Catherine M.S. Redfield, Nickolai Zeldovich and Hari Balakrishnan CryptDB – Protecting Data Confidentiality with Encrypted Query Processing
2. Carlo Curino, Evan P. C. Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, Nickolai Zeldovich,Relational Cloud: A Database-as-a-Service for the Cloud.
3. Nuno Santos, Krishna.P.Gummadi, Rodrigo Rodrigues, Towards Trusted Cloud Computing"Cryptdb",http://web.mit.edu/ralucap/www/CryptDB-sosp11.pdf
4. http://css.csail.mit.edu/cryptdb/
5. http://security.hsr.ch/msevote/seminar-papers/HS09_Homomorphic_Tallying_with_Paillier.pdf].
6. .http://www.embedded.com/design/configurable-systems/4024599/Encrypting-data- with-the-Blowfish-algorithm