© 2017 IJSRSET | Volume 3 | Issue 8 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

A Scalable Approach for Encrypted Query Routing on Encrypted Databases

Arun Kumar Silivery[1], Suvarna S[2]

Department of CSE, Raja mahendra College of Engineering,Ibrahimpatnam,Hyderabad, Andhra Pradesh, India[1]
Assistent Professor, Department of CSE,Raja mahendra College of Engineering,Ibrahimpatnam,Hyderabad, Andhra Pradesh, India[2]

ABSTRACT

Online applications are powerless against robbery of sensitive data since adversaries can abuse programming bugs to access private information, and in light of the fact that curious or malware administrators may catch and break information. CryptDB is a framework that gives practical or provable confidentiality even with these attacks for applications backed by SQL databases. It works by executing SQL queries over encoded information utilizing an accumulation of effective SQL-aware encryption models. CryptDB can likewise link encryption keys to client passwords, with the goal that an information item can be decoded just by utilizing the secret key of one of the clients with access to that information. Accordingly, a database administrator never accesses decrypted information, and regardless of the possibility that all servers are compromised, an enemy can't decrypt the information of any client who isn't signed in. An investigation of a trace of 126 million SQL queries from a generation MySQL server appears that CryptDB can support operations over encoded information for 99.5% of the 128,840 columns found in the trace. Our assessment appears that CryptDB has low overhead, decreasing throughput by 14.5% for phpBB, a web forum application, and by 26% for queries from TPCC, contrasted with unmodified MySQL. Fastening encryption keys to client passwords requires 11– 13 special pattern annotations to secure more than 20 sensitive fields and 2– 7 lines of source code changes for three multi-client web applications.

Keywords : DBMS server, CryptDB proxy server, SQL-aware Encryption and Adjustable Query-based Encryption

I. INTRODUCTION

We present CryptDB, a practical framework that investigates a middle of configuration point to give confidentiality for applications that utilize database administration frameworks (DBMSes). CryptDB is the primary framework that can execute an extensive variety of SQL queries over encrypted information. The key understanding that makes our approach functional is that most SQL queries utilize a little arrangement of all around characterized administrators, each of which we can support proficiently over encoded information. CryptDB tends to two dangers, as showed in Figure 1. The primary risk is an adversary who accesses the DBMS server and tries to learn private information by snooping on the server. This danger may emerge when an attacker abuses some vulnerability to specifically get to the DB server, when the database is outsourced to an outside organization (e.g., an open "cloud"), or when the DBMS is directed by an curious framework or database administrator (DBA) who won't not be trusted. CryptDB intends to prevent the enemy from learning private information for this situation. The second danger is a adversary who gets complete control of the application and the DBMS servers. For this situation, CryptDB ensures the confidentiality of the information having a place just to clients logged-out of the application during an attack, however can't give any certifications to signed in clients. This paper concentrates principally on the answer for begin risk; our SOSP paper points of interest the extra mechanisms that address the second risk. CryptDB requires no progressions to the internals of the DBMS server, and should work with most standard SQL DBMSes. Our execution utilizes a MySQL back-end. Our practicals demonstrate that the overhead of CryptDB is modest: throughput decreases by 26% for queries from the standard TPC-C benchmark, and by 14.5% for a multiuser bulletin board

application (phpBB), 18 looked at to running them over MySQL without encryption. We find that CryptDB accepts most queries watched by in practice: an examination of 126 million SQL queries from a MIT MySQL service demonstrated that CryptDB accepts operations over encrypted information for 99.5% of the 128,840 sections found in the query trace.

II. CryptDB's THREAT MODEL

We discuss CryptDB's threat model and provide an overview of our approach.

Threat at DBMS Server

In this danger, CryptDB makes preparations for a curious DBA or other outside attacker with full access to the information put away in the DBMS server. Our objective is confidentiality (information security), not trustworthiness or accessibility. The attacker is thought to be passive: she needs to learn confidential information, yet does not change queries issued by the application, query comes about, or the information in the DBMS.

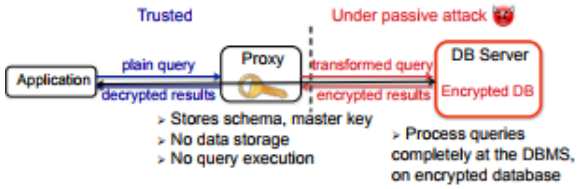

Figure: Passive Attacks on DBMS Server

This risk incorporates DBMS programming compromises, root access to DBMS machines, and even access to the RAM of physical machines. With the ascent in database solidification inside big business data centers, outsourcing of databases to open distributed cloud computing frameworks, and the utilization of outsider DBAs, this risk is progressively essential.

Threat at Arbitrary level

We now depict the second risk where the application server, intermediary proxy, and DBMS server frameworks might be compromised arbitrarily. The approach in danger 1 is deficient in light of the fact that a adversary would now be able to access the keys used to encode the whole database. The arrangement is to encrypt distinctive information items (e.g., information having a place to various clients) with various keys. To analyze the key that should be utilized for every data item, developers annotate the application's database construction to express better grained privacy policies. A curious DBA still can't acquire private information by snooping on the DBMS server (danger 1), and likewise, a adversary who compromises the application server or the intermediary proxy would now be able to decode only information of right now signed in clients (which are put away in the intermediary proxy). Information of right now inactive clients would be encoded with keys not available to the adversary, and would stay confidential. In this setup, CryptDB gives solid assurances in the substance of arbitrary server-side compromises, including those that pick up root access to the application or the intermediary proxy. CryptDB reveals at most the information of as of now dynamic clients for the length of the compromise, regardless of the possibility that the intermediary proxy acts in a Byzantine form. By "length of a compromise ", we mean the interval from the begin of the compromise until the point when any trace of the compromised has been removed from the framework. For a read SQL infusion attack, the length of the compromised traverses the attacker's SQL queries. In the above illustration of a adversary changing the email address of a client in the database, we consider the framework compromised for whatever length of time that the attacker's email address endures in the database.

III. CryptDB OVERVIEW

CryptDB gives privacy in the face of an attacker with full read access to the information put away in the DBMS server. CryptDB tends to these difficulties utilizing three key ideas: a) the first is to execute SQL queries over encrypted information. CryptDB implements this thought utilizing a SQL-aware encryption procedure, which use the way that all SQL queries are comprised of an all around characterized set of primitive administrators, for example, balance checks, arrange correlations, aggregates (entireties), and joins. By adopting known encryption plans (for balance, increases, and request checks) and utilizing another security cryptographic technique for joins, CryptDB encodes every data item in a way that enables the DBMS to execute on the carried information. CryptDB is proficient in light of the fact that it for the most part utilizes symmetric-key encryption, maintains a strategic distance from completely homomorphism encryption, and keeps running on unmodified DBMS programming (by utilizing client characterized functions). b) The second strategy is adjustable query-based encryption. Some encryption plans release more data than others

about the information to the DBMS server, however are required to process certain queries.

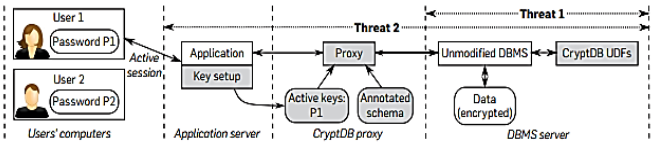

Figure 1: CryptDB's Architecture

To avoid from uncovering every possible encryption of information to the DBMS from the earlier, CryptDB carefully changes the SQL aware encryption technique for any given information item, depending on the queries saw at run-time. To actualize these adjustments proficiently, CryptDB utilizes onions of encryption. Onions are a novel approach to minimally store different cipher texts inside each other in the database and maintain a strategic distance from costly re-encryptions. c) The third thought is to chain encryption keys to user passwords, so that every data thing in the database can be decrypted just through a chain of keys established in the keyword of one of the clients with access to that information. Accordingly, if the client isn't signed into the application, and if the adversary does not know the client's secret key, the enemy can't decrypt the client's information, regardless of the possibility that the DBMS and the application server are completely compromised. To develop a chain of keys that catches the application's information security and sharing approach, CryptDB enables the developer to give policy annotations over the application's SQL schema, determining which clients (or different principals, for example, groups) have access to every data item.

IV. SQL QUERIES OVER ENCRYPTED DATA

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 CryptDB proxy stores a private master key MK, the database technique, and the present encryption layer of each column. The DBMS server sees an anonymized model (in which table and column names are exchanged by misty identifiers), encoded client information, and some assistant tables utilized by CryptDB. CryptDB likewise outfits the server with certain user defined functions (UDFs) that empower the server to figure on cipher texts for specific operations. Preparing a query in CryptDB includes four steps: 1. the application issues an query, which the intermediary proxy captures and rewrites: it anonymizes each table furthermore, column name, and, utilizing the master MK, encrypts every consistent in the query with an encryption technique most appropriate for the desired operation. 2. The intermediate proxy additionally replaces certain operations with UDFs. 3. The intermediary proxy checks if the DBMS server should be given keys to change encryption layers before executing the query, and provided that this is true, issues an UPDATE query at the DBMS server, which conjures a UDF to modify the encryption layer of the suitable columns (Section 3.2). 4. The intermediary proxy sends the encoded query to the server, which executes it. 5. The server restores the encoded query result, which the intermediary proxy decrypts and comes back to the application.

A. SQL aware Encryption: We now depict the encryption techniques utilized as a part of CryptDB, counting various existing cryptosystems and another cryptographic primitive for joins. For every encryption technique, we clarify the security property that CryptDB requires from it, its functionality, and how it is executed. ***Irregular (RND).*** RND gives the most extreme security in CryptDB: indistinctness under a adaptive chosen plaintext attack (IND-CPA); the plan is probabilistic, implying that two rise to values are mapped to various cipher texts with overpowering probability. Then again, RND does not permit any calculation to be performed efficiently on the cipher text. An effective development of RND is to utilize a square figure like AES or Blowfish in CBC mode together with an arbitrary instatement vector.

Homomorphic encryption (HOM): HOM is as secure a probabilistic encryption technique as RND, however permits the server to perform calculations on encrypted information with the last result decrypted at the intermediary proxy. Although completely homomorphism encryption is restrictively moderate, homomorphic encryption for particular operations is efficient. To help increments, we executed the Paillier cryptosystem. With Paillier, duplicating the encryptions of two values brings about an encryption of the aggregate of the values, that is, HOMK (x) · HOMK (y) = HOMK (x + y), where the multiplication is performed modulo some public key value. To register SUM totals, the intermediary proxy replaces SUM with calls to a UDF that performs Paillier increase on a column

encrypted with HOM. HOM can likewise be used to compute midpoints by having the DBMS server return the total and the tally independently, and to increase values (e.g., SET id = id + 1). HOM cipher texts are 2048 bits in length.

JOIN and OPE-JOIN (join): A different encryption technique is expected to permit balance join between two columns, on the grounds that we utilize distinctive column particular keys for DET to avoid connections between's columns. JOIN not just supports all the operations permitted by DET, yet in addition empower the server to decide repeating values between two unique columns. OPE-JOIN empowers joins by arrange relations. We give another cryptographic plan for JOIN.

Word Search (SEARCH): SEARCH is utilized to perform searches on encrypted content to help operations, for example, MySQL's LIKE administrator. Search is about as secure as RND. We executed the technique for Song et al. SEARCH as of now supports just full word search. At the point when the client plays out a search, for example, SELECT * FROM messages WHERE msg LIKE "% alice %", the intermediary proxy gives the DBMS server a token, which is an encryption of alice. The server can't decrypt the token to make sense of the hidden word. Utilizing a client characterized work, the DBMS server checks if any of the word encryptions in any message coordinate the token. All that the server gains from a SEARCH query is regardless of whether the token coordinated a message or not, and just for the tokens asked for by the client. The server would take in the same data while restoring the result set to the clients, so the plan reveals the negligible measure of extra data expected to restore the outcome.

B. Adjustable Query-based Encryption

CryptDB's adjustable query-based encryption technique takes care of this issue by progressively adjusting the layer of encryption on the DBMS server. The thought is to encrypt each information item in at least one onion: that is, each value is dressed in layers of progressively stronger encryption, as appeared in Figures 2 and 3. Each layer of every onion empowers a certain class of computation, as clarified prior. Different onions are required on the grounds that the calculations supported by various encryption plans are not generally strictly requested. Contingent upon the sort of the information, CryptDB may not keep up all onions for every column. For example, the Search onion does not make well for numbers, and they Include onion does not make sense well for strings.

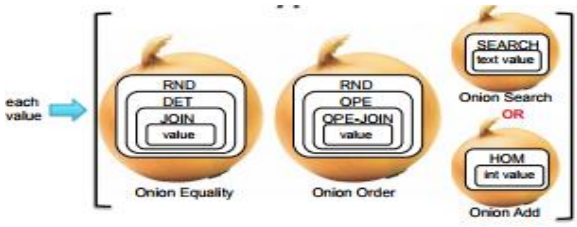

Figure 2: Onion encryption layers and the classes of computation they allow. Onion names stand for the operations they allow at some of their layers

For each layer of every onion, the intermediary proxy utilizes the same key for encoding values in a similar column, and unique keys crosswise over tables, columns, onions, and onion layers. Utilizing a similar key for all values in a column permits the intermediary proxy to perform operations on a column without having to process separate keys for each column that will be controlled. Utilizing individual keys crosswise over columns keeps the server from adapting any extra relations. These keys are derived from the private master key MK. For instance, for table t, column c, onion o, and encryption layer l, the intermediary proxy utilizes the key

$$K_{t,c,o,l} = PRP_{MK}(\text{table t, column c, onion o, layer l})$$

Where PRP is a pseudorandom permutation (e.g., AES). Every onion begins with the most secure encryption technique as the best level (RND for onions Eq and Ord, HOM for onion Add, and SEARCH for onion Search). As the intermediary proxy gets SQL queries from the application, it decides regardless of whether layers of encryption should be evacuated. On the off chance that query requires predicate P on column c, the intermediary proxy initially sets up what onion layers are expected to figure P on c. On the off chance that the encryption of c isn't now at an onion layer that permits P, the intermediary proxy strips off the onion layers to permit P on c, by sending the related onion key to the server. The intermediary proxy never decodes the information past the minimum secure non-plaintext encryption onion layer, which might be superseded by the blueprint designer to be a more secure layer (e.g., one may determine that charge card data may even under the least favorable conditions be at DET, also, never at OPE). CryptDB analyzes onion layer decryption utilizing UDFs that keep running on the

DBMS server. For instance, in Figure 3, to decrypt onion Ord of column 2 in Table 1 to layer OPE, the intermediary issues the accompanying query to the server, invoking the

DECRYPT_RND UDF: UPDATE Table1 SET C2-Ord = DECRYPT_RND (K, C2-Ord, C2-IV,)

Where K is the proper key computed from Equation (1). In the meantime, the intermediary proxy updates its own inward state to keep in mind that column C2-Ord in Table1 is currently at layer OPE in the DBMS.

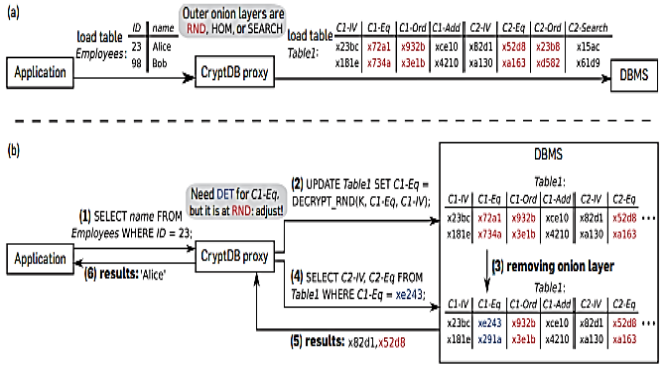

Figure 3. Examples of (a) how CryptDB transforms a table's schema and encrypts a database

V. EXPERIMENTAL EVALUATION

We examine the functionality and security of CryptDB on five applications and one huge trace: phpBB (Web forum application), HotCRP (a meeting framework), grad apply (the MIT EECS graduate confirmation application), Open-EMR (an electronic healthcare records application putting away patient medical information), TPC-C (an industry-standard database benchmark), and a follow of SQL queries from a well known MySQL server at MIT, sql. Mit.edu. This server is utilized fundamentally by Web applications running on scripts.mit.edu, a common Web application facilitating operated by MIT's Student Information Handling Board (SIPB). What's more, this SQL server is utilized by various applications that keep running on different machines what's more, utilize sql.mit.edu just to store their information. Our query trace traverses around ten days, and incorporates roughly 126 million queries more than 1193 databases and 18,162 queries. Each database is probably going to be a different occasion of an application. Every one of these applications and the huge SQL trace contain delicate data that should be secured.

Functional evaluation: We find that CryptDB accepts most queries; the number of columns in the "requirements plaintext" column, which checks columns that can't be prepared in encrypted form by CryptDB, is little in respect to the quantity of columns encrypted. For OpenEMR, CryptDB does not accept queries on certain sensitive fields that perform string control (e.g., substring and lowercase transformations) or date control (e.g., acquiring the day, month, or year of an encoded date). Notwithstanding, if these capacities were precomputed with the outcomes included as independent columns (e.g., by encoding the three columns of a date independently), CryptDB would support these queries.

Security evaluation: To determine the measure of data that would be uncovered to the adversary practically speaking, we look at the steady state onion levels of various columns. To evaluate the level of security, we characterize the MinEnc of a column to be the weakest onion encryption technique uncovered on any of the onions of a column when onions achieve a steady state (i.e., after the application creates all query types, many running the entire trace). We consider RND and HOM as the strongest plans, trailed via SEARCH, trailed by DET and JOIN, and completing with OPE, which is the weakest technique. For instance, if a column has onion Eq at RND, onion Ord at OPE, and onion Add at HOM, the MinEnc of this column is OPE. The correct side of Figure 4 demonstrates the MinEnc onion level for our applications and query traces.

Application	Total columns	Encrypted columns	Min level is RND	Min level is DET	Min level is OPE
phpBB	563	23	21	1	1
HotCRP	204	22	18	1	2
grad-apply	706	103	95	6	2
TPC-C	92	92	65	19	8
sql.mit.edu	128,840	128,840	80,053	34,212	13,131

Most columns at RND

Most columns at OPE analyzed were less sensitive

Figure 4: Steady-state onion levels for database columns required by a range of applications and traces.

VI. PERFORMANCE EVALUATION

To assess the performance of CryptDB, we utilized a machine with two 2.4GHz Intel Xeon E5620 4-center processors and 12GB of RAM to run the MySQL 5.1.54

server, and a machine with eight 2.4GHz AMD Opteron 8431 6-center processors furthermore, 64GB of RAM to run the CryptDB intermediary proxy and the customers. The two machines were associated over a mutual Gigabit Ethernet network. The higher-provisioned customer machine guarantees that the customers are not the bottleneck in any experimental trails. All workloads fit in the server's RAM. We look at the performance of a TPC-C query blend when running on an unmodified MySQL server versus on a CryptDB intermediary proxy in front of the MySQL server. We prepared CryptDB on the query set so there are no onion alterations during the TPC-C tests.

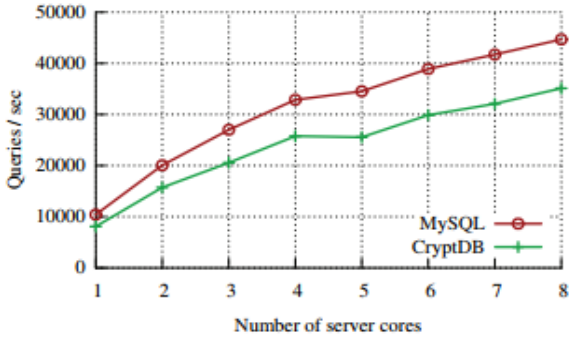

Figure 5: Throughput for TPC-C queries, for a varying number of cores on the underlying MySQL DBMS server.

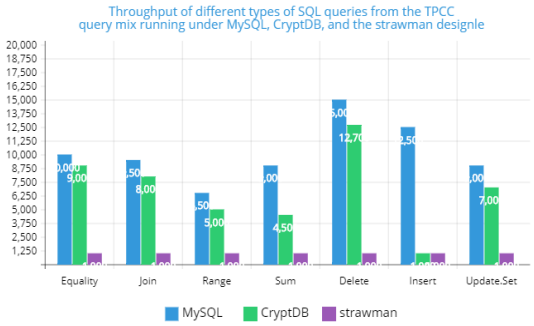

Figure 6: Throughput of different types of SQL queries from the TPCC query mix running under MySQL, CryptDB, and the strawman design.

Figure 5 demonstrates the throughput of TPC-C queries as the quantity of centers on the server shifts from one to eight. In all cases, the server invests 100% of its CPU energy preparing queries. Both MySQL and CryptDB scale well at first, however begin to level off because of inside secure conflict in the MySQL server, as announced by SHOW STATUS LIKE 'Table%'. The general throughput with CryptDB is 21– 26% lower than MySQL, contingent upon the correct number of centers. To comprehend the wellsprings of CryptDB's overhead, we measure the server throughput for various types of SQL queries seen in TPC-C, on a similar server, however running with just a single center empowered. Figure 6 demonstrates the outcomes for MySQL, CryptDB, and a strawman outline; the strawman plays out each query over information encrypted with RND by decrypting the significant information utilizing a UDF, playing out the query over the plaintext, and re-encoding the outcome (if refreshing rows).

VII. CONCLUSION

We exhibited CryptDB, a framework that gives a practical and solid level of confidentiality even with two significant dangers going up against database-supported applications: curious DBAs and arbitrary compromises of the application server and the DBMS. CryptDB meets its objectives utilizing three thoughts: running queries effectively finished encrypted information utilizing a novel SQL-mindful encryption system, powerfully modifying the encryption level utilizing onions of encryption to limit the data revealed to the untrusted DBMS server, also, chaining encryption keys to client passwords in a way that permits just approved clients to access scrambled information.

VIII. REFERENCES

[1]. F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In Proceedings of the 4th International Conference on Information Security Practice and Experience, Sydney, Australia, April 2008.

[2]. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Orderpreserving symmetric encryption. In Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Cologne, Germany, April 2009.

[3]. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In Proceedings of the 4th Conference on Theory of Cryptography, 2007.

[4]. A. Chen. GCreep: Google engineer stalked teens, spied on chats. Gawker, September 2010. http://gawker.com/5637234/.

[5]. A. Chlipala. Static checking of dynamically-varying security policies in database-backed applications. In Proceedings of the 9th Symposium on Operating Systems Design and Implementation, Vancouver, Canada, October 2010.

[6]. S. S. M. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In Proceedings of the 16th Network and Distributed System Security Symposium, February 2009.
[7]. V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In Proceedings of the 14th European Symposium on Research in Computer Security, September 2009.
[8]. M. Cooney. IBM touts encryption innovation; new technology performs calculations on encrypted data without decrypting it. Computer World, June 2009.