International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 9
ISSN: 2454-4248
91 – 97

Retrieving Data from Encrypted file by using Cosine Similarity

G. Mounica,
PG Scholar,
Department of CSE,
MVSR Engineering College.

J. Prasanna Kumar,
Professor,
Department of CSE,
MVSR Engineering College.

Abstract-Nowadays every user who uses Internet wants to search for anything and everything using Search Engines. This is the need for everyone. To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of files involving a comparable number of distinct terms. They answer tens of millions of queries every day. Due to rapid advance in technology and need for information security, creating a web search engine today is very different from three years ago. The overall goal of this project is to develop a scalable, high performance search engine which searches the encrypted data without decryption. The main focus is on the algorithmic challenges and encryption while supporting fast searches on it. To develop this project, an applied ranking algorithm to give better results to the user and also used other algorithms to encrypt the data stored. To ease for searching of various information over the data by giving search keywords requires software. The search engine software ensures the end user to get the information by accessing the data specified in the database.

Keywords: *Database Administration, Security, integrity, and protection.*

*****

I. Introduction

Online 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. An analysis of a trace of 126 million SQL queries from a production MySQL server shows that CryptDB can support operations over encrypted data for 99.5% of the 128,840 columns seen in the trace. the evaluation shows that CryptDB has low overhead, reducing throughput by 14.5% for phpBB, a web forum application, and by 26% for queries from TPC-C, compared to unmodified MySQL. Chaining encryption keys to user passwords requires 11-13 unique schema annotations to secure more than 20 sensitive fields and 2-7 lines of source code changes for three multi-user web applications.

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, 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 Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.

 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 homomorphism encryption schemes are still prohibitively expensive by orders of magnitude.

The cleverest part of CryptDB, however, is that it's able to switch between crypto systems on the fly depending on the operation. The data in the database is encrypted in multiple layers of different encryption, what the researchers call an "onion" of encryption. Every layer allows different kinds of computation and has a different key.The most secure schemes are used on the outside of that onion and the least secure are used on the inside. CryptDB manages to perform all its functions of a database without ever removing that

 ISSN: 2454-4248
91 – 97

last layer of the onion, so that the data always remains secure.

CryptDB has its limits, the MIT researchers warn--no square roots, for one example. And while the data is never completely decrypted, it does "leak" information about the underlying data when enough outer layers of encryption are removed, revealing attributes like which data points are equal to each other. the sampled operations from several real databases like one used by a Web forum and another by a grade-calculating application, and found that their encrypted system would allow the same calculations as an unencrypted database in 99.5% of those operations, and that data the researchers deemed "sensitive" is never leaked in those test cases.

II. Related work

Now a days every user who uses Internet wants to search for anything and everything using Search Engines. Due to rapid advance in technology and need for information security, creating a web search engine today is very different from three years ago. The overall goal is to develop a scalable, high performance search engine, which searches the encrypted data without decryption. In this calculated the tf-idf and cosine similarity for every term to search the keyword.

Query-biased preview over outsourced and encrypted data.

For both convenience and security, more and more users encrypt their sensitive data before outsourcing it to a third party such as cloud storage service. However, searching for the desired documents becomes problematic since it is costly to download and decrypt each possibly needed document to check if it contains the desired content. An informative query-biased preview feature, as applied in modern search engine, could help the users to learn about the content without downloading the entire document. However, when the data are encrypted, securely extracting a keyword-in-context snippet from the data as a preview becomes a challenge. Based on private information retrieval protocol and the core concept of searchable encryption, a propose of a single-server and two-round solution to securely obtain a query-biased snippet over the encrypted data from the server.

A layered searchable encryption scheme with functional components independent of encryption methods.

Searchable encryption technique enables the users to securely store and search their documents over the remote semitrusted server, which is especially suitable for protecting sensitive data in the cloud. However, various settings (based on symmetric or asymmetric encryption) and functionalities (ranked keyword query, range query, phrase query, etc.) are often realized by different methods with different searchable structures that are generally not compatible with each other, which limits the scope of application and hinders the functional extensions.

To prove that asymmetric searchable structure could be converted to symmetric structure, and functions could be modeled separately apart from the core searchable structure. Based on this observation, To propose a layered searchable encryption (LSE) scheme, which provides compatibility, flexibility, and security for various settings and functionalities. In this scheme, the outputs of the core searchable component based on either symmetric or asymmetric setting are converted to some uniform mappings, which are then transmitted to loosely coupled functional components to further filter the results.

Secure Inverted Index Scheme

An inverted index is a data structure loading words or numbers in a file along with its location. The determination of an inverted index is to progress the time of full text searches. An inverted index holds an index of keywords which stores a different list of terms finding the collection and, for individually term, a posting the updating list of documents that hold the keyword. An inverted index improves search effectiveness which is required for very large text files. An inverted index consists of a distinct term and a posting list which stores the IDs of the documents that hold that term. In count to an ID, each posting holds list element gives the number of rates occurrences of that term in the document.

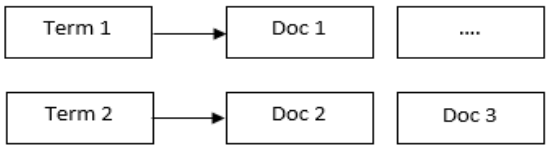


Figure: Structure of an Inverted Index

It provides good retrieval performance as well as better security for indexes. The major drawback of this process is that, It Track unnecessary network traffic for retrieval of data. Public Key Encryption with keyword Search.

III. Problem Statement

This problem is concerned with how to provide a list of documents in the database corresponding to the words in a query. Each time the documents containing a one word of query are only given as the output of the proposed system. The query may contain many words that need to be located in the documents.

IV. System Implementation Architecture

CryptDB comprises of three major components, namely the Application Server, Proxy Server and DBMS Sever. Application Server is the main server that runs CryptDB's

 ISSN: 2454-4248
91 – 97

database proxy and also the DBMS Server. The Proxy Server stores a secret Master Key, the database Schema and the onion layers of all the columns in the database. The DBMS server has access to the anonymized database schema, encrypted database data and also some cryptographic user-defined functions (UDF's). The DBMS server uses these cryptographic UDF's to carry out certain operations on the encrypted data (cipher text).

The CryptDB proxy consists of a C++ library and a Lua module. The C++ library consists of a query parser; a query encryptor/rewriter, which encrypts fields or includes UDFs in the query; and a result decryption module. To allow applications to transparently use CryptDB, used MySQL proxy [47] and implemented a Lua module that passes queries and results to and from our C++ module. CryptDB implementation consists of _18,000 lines of C++ code and _150 lines of Lua code, with another _10,000 lines of test code. CryptDB is portable and had implemented versions for both Postgres 9.0 and MySQL 5.1. The initial Postgres-based implementation is described in an earlier technical report [39]. Porting CryptDB to MySQL required changing only 86 lines of code, mostly in the code for connecting to the MySQL server and declaring UDFs. As mentioned earlier, CryptDB does not change the DBMS; all server-side functionality are implemented with UDFs and server-side tables. CryptDB's design, and to a large extent implementation, should work on top of any SQL DBMS that supports UDFs.

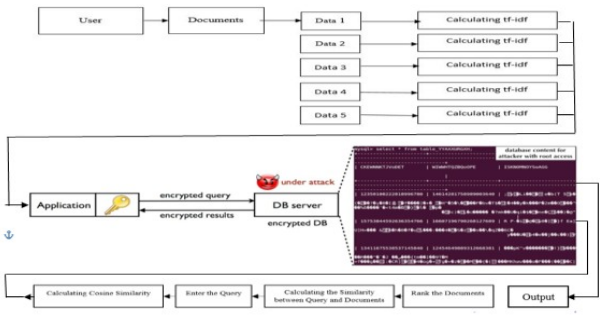


Figure: System Architecture

V. CryptDB

CryptDB is a DBMS that provides provable and practical privacy in the face of a compromised database server or curious database administrators. CryptDB works by executing SQL queries over encrypted data. At its core are three novel ideas: an SQL-aware encryption strategy that maps SQL operations to encryption schemes, adjustable query based encryption which allows CryptDB to adjust the encryption level of each data item based on user queries, and onion encryption to efficiently change data encryption levels. CryptDB only empowers the server to execute queries that the users requested, and achieves maximum privacy given the mix of queries issued by the users.

The database server fully evaluates queries on encrypted data and sends the result back to the client for final decryption; client machines do not perform any query processing and client-side applications run unchanged.

CryptDB is a system that provides practical, provable confidentiality and privacy that guarantees without having to trust the DBMS server or the DBAs who maintain and tune the DBMS. CryptDB is the first private system to support all of standard SQL over encrypted data without requiring any client-side query processing, modifications to existing DBMS codebases, changes to legacy applications and offloads virtually all query processing to the server. CryptDB works by rewriting SQL queries, storing encrypted data in regular tables, and using an SQL user-defined function (UDF) to perform server-side cryptographic operations.

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.

Below describe the steps involved in processing a query inside CryptDB

In the first step a query is issued by the application server. The proxy server receives this query and it anonymizes the table name and each of the column names. The proxy server also encrypts all the constants in the query using the stored secret master key. The encryption layers or the onion layers are also adjusted based on the type of operation required by the issued query. For example, if the query has to perform some equality checks then the deterministic encryption Scheme (DET)is applied to encrypt all the values in that particular column (on which equality check is to be performed).

The encrypted user query is then passed on to the DBMS server. The DBMS server executes these queries using standard SQL and also invokes UDF's to perform certain operations like token search and aggregation. The queries are executed on the encrypted database data. The DBMS server performs computations on the encrypted data and forwards the encrypted results back to the proxy server. The proxy server decrypts the encrypted query result obtained and returns it to the application server.

 ISSN: 2454-4248
91 – 97

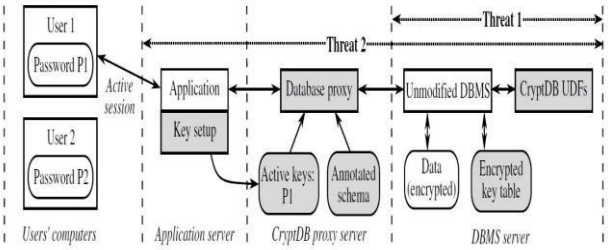


Figure: Threats

Threat 1: DBMS server compromise CryptDB provides confidentiality for the content of the data and for names of columns and tables, but does not hide the overall table structure, the number of rows, the types of columns, or the approximate size of data in bytes. The only information that CryptDB reveals to the DBMS server is relationships among data items corresponding to 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), which means 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.

Threat 2: arbitrary confidentiality attacks on any servers the second threat is where the application server, proxy, and DBMS server infrastructures may be compromised arbitrarily. The approach in threat 1 is insufficient because an adversary can now get access to the keys used to encrypt the entire database. The solution is to encrypt different data items (e.g., data belonging to 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.

VI. Encryption Types

Each type uses a different algorithm that meets the specified requirements for a certain type and can be exchanged for another algorithm should the need arise, e.g. when a used cipher is broken. In such an event existing encrypted data would have to be decrypted with the old algorithm and re-encrypted using the new one. And had listed the different layers from most to least secure. Whereas least secure means that this particular layer does reveal the most information about its encrypted content, please notice that this is sometimes unavoidable in order to perform certain operations and is not automatically insecure.

Random (RND)

The RND onion layer provides the strongest security assurances: It is probabilistic, meaning that the same plaintext will be encrypted to a different cipher text. On the other hand, it does not allow any feasible computation in a reasonable amount of time. If someone wants to know something about the content of these fields the encrypted data has to be retrieved as a whole to be decrypted by CryptDB. This type seems to be reasonable choice for highly confidential data like medical diagnosis, private messages or credit card numbers that do not need to be compared to other entries for equality.

Homomorphic encryption (HOM)

The HOM onion layer provides an equally strong security assurance, as it is considered to be IND-CPA secure too [1]. It is specifically designed for columns of the data type integer and allows the database to perform operations of an additive nature. This includes of course the addition of several entries, but also operations like SUM or AVG. The reason that only addition is supported lies in the fact that fully homomorphic calculations, while mathematically proven by M. Cooney, is unfeasible slow on current hardware. An exception is the homomorphic addition $HOM(x) \cdot HOM(y) = HOM(x + y) \bmod n$, that can be performed in a reasonable amount of time. In CryptDB the developers choose to implement the homomorphic addition using the Paillier cryptosystem [13]. Currently the cipher text of a single integer is stored a VARBINARY (256), this means it uses 256 bytes of space which is 64 times the size of a normal integer thatwouldonlyuse4bytes. Considering that integers are among the most used data types in a database. This is a huge overhead. Popa et al. indicate that there might be a more efficient way to store the integers with the use of a scheme developed by Ge and Zdonik. As of today this has not been implemented.

Order-preserving encryption (OPE)

The OPE onion layer is significantly weaker than the DET layer as it reveals the order of the different entries. This means that the DBMS knows relations like bigger and smaller, but also equality (without having to look at the Eq onion). This means that if $x < y$, then $OPE(x) < OPE(y)$, also if $x = y$, then $OPE(x) = OPE(y)$. This allows us to use ordered operations like MIN, MAX or ORDER BY. To achieve this functionality, the developers of CryptDB implemented an algorithm that was published by Boldyreva et al. and was inspired by the ideas suggested by Agrawal et al. In regards to security it is noteworthy that this onion

 ISSN: 2454-4248
91 – 97

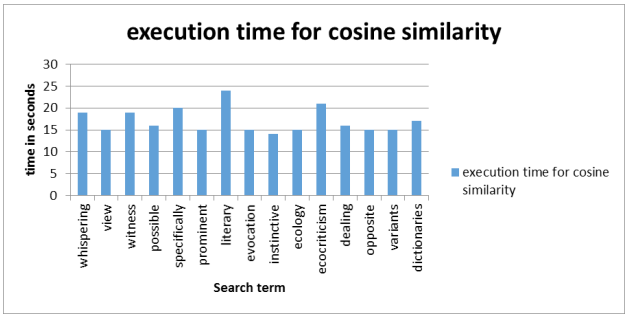
layer is the most revealing one: It cannot fulfill the security definition of IND-CPA, as is shown by Boldyreva et al. Even more important it reveal snot just the order but also the proximity of the transformed values to an attacker. This behavior might be acceptable for some values (e.g. text), but might be an issue for others (e.g. financial data).

CryptDB security related papers

Even though security is not an official part of this thesis, security is still an important topic when it comes to usability and whether it is worth the additional coasts. One question that had in the beginning was whether a curious database administrator could still draw conclusions from the encrypted data sets and whether he would able to take advantage of that, either by getting interesting insights or by actually being able to manipulate things in a way that would gain him further access to data. For these questions like to feature the following two papers: The first one is "On the Difficulty of Securing Web Applications using CryptDB" by Ihsan H. Akin and Berk Sunar and the second one is "Inference Attacks on Property Preserving Encrypted Databases" by Muhammad Naveed, Seny Kamara and Charles V. Wright.

VII. Expermental Evolution

Cosine Similarity is calculated by encrypting the files using tf-idf values and the results are evaluated by knowing the search time required to search the keyword.



VIII. Conclusion

In the face of snooping Database Administrators (DBAs) and compromise from attackers, confidentiality in Database Management Systems (DBMS) should still remain. CryptDB is a system which acts as a proxy to secure the communication between the database server and the applications server. an application is built over cryptdb that receives query from the user and returns the most relevant documents to the user. Vector based model is employed for information retrieval. In the first step, all the keywords in the document collection and corresponding tf-idf values are send to cryptdb.

CryptDB receives them from the application server secures them and sends them to the database server. When the user gives query, application will invoke the CryptDB for tf-idf values. CryptDB will receive the encrypted data from the database decrypts it and sends it to the application server. From these tf-idf values document vectors are created. Then using cosine similarity, the relevant documents are retrieved and ranked and returned to the user. Thus, the entire retrieval process is secured.

IX. References

1. Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, October 2011. (This is the main paper describing CryptDB.)
2. Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich.An Ideal-Security Protocol for Order-Preserving Encoding.In Proceedings of the 34th IEEE Symposium on Security and Privacy (IEEE S&P/Oakland), San Francisco, CA, May 2013.(This paper constructs the encryption scheme that computes order queries in CryptDB.)
3. Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing Analytical Queries over Encrypted Data.In Proceedings of the 39th International Conference on Very Large Data Bases (VLDB), Riva del Garda, Italy, August 2013.(This paper extends CryptDB's basic design to complex analytical queries and large data sets.)
4. Raluca Ada Popa and Nickolai Zeldovich.Cryptographic treatment of CryptDB's Adjustable Join.Technical Report MIT-CSAIL-TR-2012-006, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, March 2012.(A formal description and analysis of CryptDB's adjustable join cryptographic scheme.)
5. Carlo Curino, Evan P. C. Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich.Relational Cloud: A Database-as-a-Service for the Cloud.In Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011), Pacific Grove, CA, January 2011.(A paper describing how CryptDB can help with hosting databases in the cloud.)
6. Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: improved securityanalysis and alternative solutions.
7. Seny Kamara. Attacking encrypted database systems, blog post, Outsourced bits, snapshot as of Sept 7, 2015.
8. Vladimir Kolesnikov and Abdullatif Shikfa. On the limits of privacy provided by order-preserving encryption. Bell Labs Technical Journal, 17(3):135–146, 2012.
9. Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. 2015.

10. MySQL AB, "MySQL performance benchmarks," A MySQL Technical White Paper, 2005. [Online]. Available: http://www.jonahharris.com/osdb/mysql/mysql-performance-whitepaper.pdf
11. Google,"Encryptedbigqueryclient,"https://github.com/google/encrypted-bigquery-client, 2015.
12. P. Grofig, M. Haerterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schroepfer, and W. Tighzert, "Experiences and observations on the industrial implementation of a system to search over out-sourced encrypted data." in Sicherheit, 2014, pp. 115–125.
13. I. H. Akin and B. Sunar, "On the difficulty of securing web applications using cryptdb," in Big DataB and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on. IEEE, 2014, pp. 745–752.
14. M.Kantarcioglu and C. Clifton. Security issues in queryingencrypted data. Technical Report CSD TR 04-013, PurdueUniversity, Department of Computer Sciences, 2004.
15. L. Xiong, S. Chitti, and L. Liu. Preserving data privacy foroutsourcing data aggregation services. Technical report,
16. Savoy, J. (1997) Statistical inference in retrieval effectiveness evaluation, Information Processing & Management, 33(4):495-512.
17. Chen, H., Tobun, D.N., Martinez, J., & Schatz, B. (1997). A Concept Space Approach to Addressing the Vocabulary Problem in Scientific Information Retrieval: An Experiment on the Worm Community System. Journal of the American Society for Information Science.
18. Hofmann, T. (1999). Probabilistic Latent Semantic Indexing. In Proceedings of the 22st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99) (pp. 50-57). ACM.
19. Raghavan, V.V., & Wong, S.K.M. (1986). A Critical Analysis of Vector Space Model for Information Retrieval. Journal of the American Society for Information Science, 37, 279- 87.
20. Gotlieb, S.D.J, & Avinash, S. (1968). Semantic clustering of index terms. Journal of the ACM, 15(4), 493-513.