Secure Database Techniques in Encrypted Database Systems

Ryan Su
pinwensu@cs.ucsb.edu

June 2018

Abstract

In the era of big data, there is an increasing need to store huge amount of private data. Database security becomes extremely crucial and many Encrypted Database Systems (EDB) has been proposed. CryptDB is widely used in these systems. This paper explores the performance of such systems as well as some of the key schemes in secure database techniques, i.e. CryptDB, the threats it addresses, SQL-aware Encryption including Deterministic Encryption(DET) and Order-Preserving Encryption (OPE). Additionally, we looked into interference attacks on DET-encrypted attributes.

1 Introduction

With an increasing amount of private data being collected nowadays, database is widely used by companies, government and even individuals. Thus, database security is a crucial topic in the area of Cryptographic Engineering. As a result, many encrypted database systems have been proposed. Most of them are based on the design of CryptDB.

CryptDB[8] provides confidentiality for applications on database management systems (DBMS). It consists of a DBMS server and a separate application server and executes queries over encrypted data. CryptDB prevents database administrators from learning private data and stops adversaries from taking complete control of application and DBMS servers. It overcomes the challenges of minimizing the amount of confidential information revealed to the DBMS server and the amount of data leaked when an adversary compromises the application server in addition to the server.

Popular approaches to search on encrypted data includes searchable symmetric encryption (SSE)[3, 10], fully-homomorphic encryption (FHE) [9], oblivious RAMs

(ORAM)[5], and property-preserving encryption (PPE)[1, 2]. There is a trade-off between security and efficiency. CryptDB is largely based on property-preserving encryption(PPE). PPE is an encryption scheme that leaks a certain property of the plaintext. Some of the popular property-preserving encryption methods include Deterministic(DET) and order preserving encryption(OPE).

This tutorial paper is focused on some of the key schemes in secure database techniques. Chapter 2 explores CryptDBs architecture and the key ideas used in overcoming the challenges as well as the two threat models addressed by CryptDB. Chapter 3 discusses encryption types used by CryptDB and the security property. Case study and performance analysis are included in Chapter 4. Chapter 5 deals with concrete inference attacks against DET based systems on the CryptDB design.

2 CryptDB

CryptDB 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.

2.1 Architecture

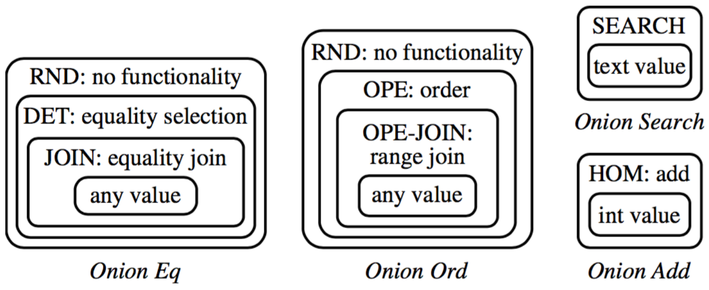


Figure 1: CryptDB Onion encryption layers[8]

CryptDB is a layered architecture; each has 4 layers (onions). The encryption of a data in the database is computed in a layered way. Figure 1 shows the Onion encryption layers and the classes of computation they allow. The terms will be explained in Chapter 3. Onion names stand for the operations they allow at some of their layers. 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. In practice, some onions or onion layers may be omitted, depending on column types or schema annotations provided by application developers.

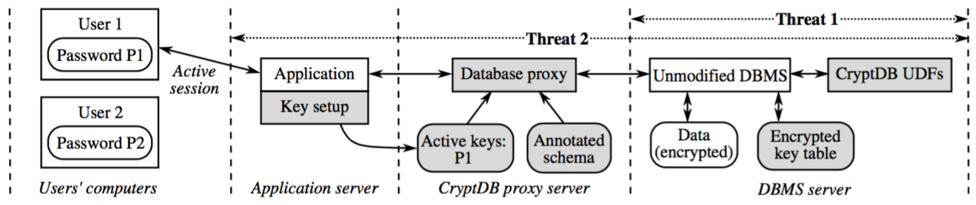


Figure 2: CryptDBs architecture[8]

CryptDBs architecture and threat models are shown in Figure 2. It consists of a database proxy and an unmodified DBMS. In the DBMS, user-defined functions (UDFs) are used to perform cryptographic operations. In Figure 2, processes are represented by rectangular boxes and data is represented by rounded boxes. Components added by CryptDB are shaded. The separation between users computers, the application server, CryptDB proxy server and the DBMS server is represented by dashed lines. CryptDB addresses two threats. It prevents database administrators from learning private data and stops adversaries from taking complete control of application and DBMS servers.

2.2 Key Ideas in Addressing Threats

The two kinds of threats are shown as dotted lines in Figure 2. In threat 1, a curious database administrator(DBA) with complete access to the DBMS server snoops on private data. In threat 2, an adversary gains complete control over both the software and hardware of the application, proxy, and DBMS servers.

Threat 1: DBMS Server Compromise[8]. The assumption is that DBA wants to get private data without changing queries. This threat is increasingly important with the rise of database in companies and outsourcing of databases. It includes DBMS software compromises, root access to DBMS machines, and even access to the RAM of physical machines. The approach that prevents such threat is to executing SQL queries over encrypted data on the DBMS server. CryptDB prevents the DBA from learning private data. The proxy uses secret keys to encrypt all data inserted or included in queries issued to the DBMS. By using SQL-aware encryption that adjusts dynamically to the queries presented, only relationships among data items that correspond to the classes of computation that queries perform are revealed. Such queries include comparing equality, sorting or searching. CryptDB guarantees that sensitive data is never available in plain text at DBMS. The type of queries determines how much information is revealed to the DBMS server. Nothing about the data content will be revealed if no relational predicate filtering on a column is requested. If the application requests equality checks on a column, CryptDBs proxy reveals which items repeat in that column, but not the actual values. If the application requests order checks on a column, the proxy only reveals the order of the elements in the column. Furthermore, DBMS server cannot compute the encrypted results for queries that involve computation classes not requested by the application.

Threat 2: Adversary gaining complete control of servers. In this type of threat, application server, proxy, and DBMS server infrastructures may be compromised arbitrarily. Thus, the approach for previous threat is not applicable here. The solution is to encrypt different data with different keys. Developers annotate their SQL schema to define different principals, whose keys will allow decrypting different parts of the database. CryptDB leaks at most the data of currently active users for the duration of the compromise. Data of currently inactive users would remain confidential because it has been encrypted with keys not available to the adversary.

There are two challenges in combating these threats. The first one is to minimize the amount of confidential information revealed to the DBMS server and the second one is to minimize the amount of data leaked when an adversary compromises the application server in addition to the server. The application must be able to access decrypted data since arbitrary computation on encrypted data is not practical. CryptDB exploits the following three key ideas to overcome these challenges.

- **Executing SQL queries over encrypted data[8].** This is achieved by using a using a SQL-aware encryption strategy. Such encryption strategies will be discussed later in this tutorial. All SQL queries are made up of equality checks. Based on this fact, CryptDB encrypts each data item in a way that allows the DBMS to execute on the transformed data. This could be achieved by adapting known encryption schemes for equality and additions and using a new privacy preserving method for joins. The reason why CryptDB is efficient is that it mainly exploits symmetric-key encryption and runs on unmodified DBMS platforms.

- **Adjustable query-based encryption[8].** There are certain encryption schemes that are required to process certain types of queries, even though they might leak more information about the data to the DBMS server. CryptDB carefully adjusts the SQL-aware encryption scheme for any given data item, depending on the queries observed at run-time. Doing so makes sure that possible encryptions of data will not be all revealed to the DBMS. Onions of encryption are used to address these adjustments, which is a novel way to avoid expensive re-encryptions. It can also store multiple ciphertexts within each other in the database.

- **Chain encryption keys to user passwords[8].**This enables that each data item in the database can be decrypted only through a chain of keys rooted in the password of one of the users with access to that data. Compromised as the server is, there is no way that the adversary can decrypt the users data if he doesn't know the password or if the user is not logged into the application. CryptDB allows the developer to provide policy annotations over the applications SQL schema. It also specifies which users have access to each data item.

3 SQL-aware Encryption

Random (RND) This is the scheme that provides the maximum security in CryptDB. In this probabilistic scheme, two equal values are mapped to different ciphertexts with overwhelming probability. In addition, it 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).

Deterministic (DET) A symmetric DTE scheme $DTE = (Gen, Enc, Dec)$ is a symmetric encryption scheme. Enc is not randomized and each message m is mapped by Enc to a single ciphertext under a key K. DET scheme provides a slightly weaker guarantee compared to RND but it still provides strong security. The main reason is that it leaks only which encrypted values correspond to the same data value. This is achieved by generating the same ciphertext for the same plaintext deterministically. This encryption layer allows the server to perform equality checks. DET is a pseudo-random permutation (PRP) [4].

Homomorphic encryption (HOM) This is a secure probabilistic encryption scheme that allows server to perform computations on encrypted data. The result will be decrypted at the proxy after computation. Unfortunately, fully homomorphic encryption[6], is very inefficient. What it did in CryptDB is to enable HOM for specific operations. HOM can also be used for computing averages by having the DBMS server return the sum and the count separately.

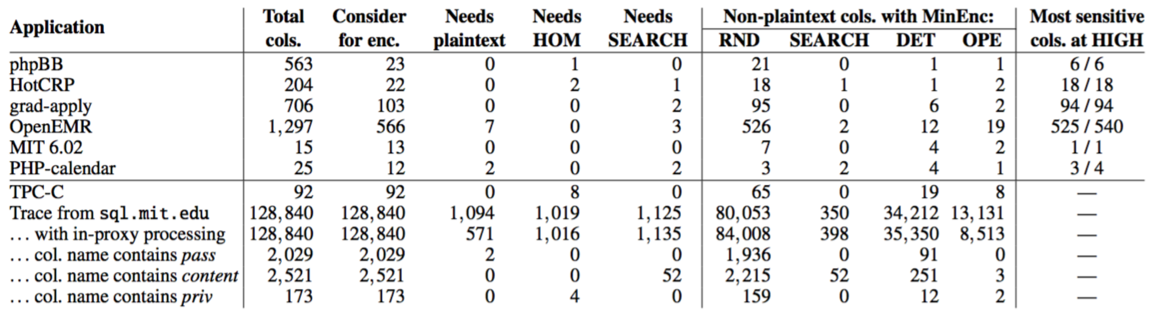
Order-preserving encryption (OPE) OPE is a weaker encryption scheme than DET because it reveals order. A symmetric OPE scheme $OPE = (Gen, Enc, Dec)$ is a symmetric encryption scheme with the following property: if $m1 > m2$, then $Enc_K(m1) > Enc_K(m2)$; if $m1 = m2$, then $Enc_K(m1) = Enc_K(m2)$;if $m1 < m2$, then $Enc_K(m1) < Enc_K(m2)$. CryptDB proxy will only reveal OPE-encrypted columns to the server if users request order queries on those columns. It also includes a hypergeometric sampler that lies at the core of OPE.

Join (JOIN) Since we use different keys for DET to prevent cross-column correlations, we need a separate encryption scheme to allow equality joins between two columns. Thats why we have Join. Moreover, it also supports all operations allowed by DET, and also enables the server to determine repeating values between two columns.

Word search (SEARCH)[8] SEARCH is almost as secure as RND. It does not reveal if a certain word repeats in multiple rows, but leaks the number of keywords encrypted. An adversary may estimate the number of distinct or duplicate words. SEARCH is used to perform searches on encrypted text to support operations. For each column needing SEARCH, the text are split into keywords using standard delimiters before removing repetition in these words. This is achieved by randomly permuting the positions of the words and encrypting each of the words to the same size. SEARCH allows CryptDB to only perform full-word keyword searches and it does not support arbitrary regular expressions.

4 CryptDB Performance Analysis

Popa et al.[8] implemented implemented versions for both Postgres 9.0 and MySQL 5.1. The CryptDB proxy consists of a C++ library and a Lua module. In this chapter, we will dive into the performance of CryptDB in different applications. They evaluated TPC-C queries, SQL queries from `sql.mit.edu` to evaluate what columns, operations and queries CryptDB supports. See the results in Figure 3.



Application	**Total cols.**	**Consider for enc.**	**Needs plaintext**	**Needs HOM**	**Needs SEARCH**	**Non-plaintext cols. with MinEnc:**				**Most sensitive cols. at HIGH**
						RND	**SEARCH**	**DET**	**OPE**	
phpBB	563	23	0	1	0	21	0	1	1	6 / 6
HotCRP	204	22	0	2	1	18	1	1	2	18 / 18
grad-apply	706	103	0	0	2	95	0	6	2	94 / 94
OpenEMR	1,297	566	7	0	3	526	2	12	19	525 / 540
MIT 6.02	15	13	0	0	0	7	0	4	2	1 / 1
PHP-calendar	25	12	2	0	2	3	2	4	1	3 / 4
TPC-C	92	92	0	8	0	65	0	19	8	—
Trace from `sql.mit.edu`	128,840	128,840	1,094	1,019	1,125	80,053	350	34,212	13,131	—
… with in-proxy processing	128,840	128,840	571	1,016	1,135	84,008	398	35,350	8,513	—
… col. name contains *pass*	2,029	2,029	2	0	0	1,936	0	91	0	—
… col. name contains *content*	2,521	2,521	0	0	52	2,215	52	251	3	—
… col. name contains *priv*	173	173	0	4	0	159	0	12	2	—

Figure 3: Steady-state onion levels for database columns[8]

Functional

The number of columns in the needs plaintext column counts columns that cannot be processed in encrypted form by CryptDB. According to the table, it is quite small relative to the total number of columns. For the applications in the top group of rows, sensitive columns were determined manually, and only these columns were considered for encryption. For the bottom group of rows, all database columns were automatically considered for encryption. We can also see that CryptDB will only support queries on certain sensitive fields that perform string or data manipulation if they were precomputed standalone columns. needs HOM and needs SEARCH means the number of columns for which that encryption scheme is needed to process some queries. It indicates that CryptDB is not able to support those queries without the schemes. Furthermore, CryptDB is able to process queries over encrypted data over about 99.5% of the columns.

Security

To evaluate security of CryptDB, amount of information that would be revealed to the adversary is examined. This is achieved by examining the steady-state onion levels of different columns among applications. The MinEnc of a column is defined to be the weakest onion encryption. Such definition makes possible quantifying the level of security. As described in Chapter 3. RND and HOM are considered the strongest schemes, followed by SEARCH, DET, JOIN and OPE. Now let's look back Figure 3 above. The right side shows the MinEnc onion level for a range of applications and query traces. It is clear that most of the fields remain at RND. Quite some remains at DET, mainly those for looking up and joins. OPE occurs with least frequency mainly because leaks orders. We can conclude that there is an improvement By CryptDB in encryption scheme confidentiality.

RND, HOM and DET for columns without repetition are refined to be security level HIGH. The rightmost column shows the applications most sensitive database columns and the number of them that have MinEnc in HIGH. From the quieres that they picked, 93% of them remain at DET or above, which is a very high number. In addition, CryptDB reveals much less information about these columns containing password or private information, almost all of them are at RND or DET.

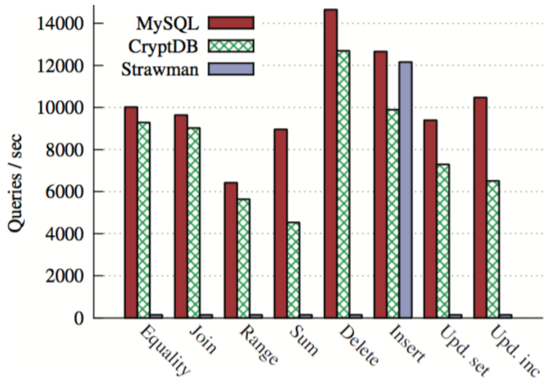


Figure 4: Throughput of different types of SQL queries[8]

Performance

It is found that the overall throughput with CryptDB is around 25% lower than MySQL, depending on the exact number of cores. Figure 4 includes the throughput given different types of SQL queries for MySQL, CryptDB, and strawman design. Upd. inc stands for UPDATE that increments a column, and Upd. set is when columns are set to a constant. We can conclude from the figure that CryptDB performs the worst on queries involves HOM additions, and relative better on other types of queries. Regarding the latency, there is an overall server latency increase of 20% with CryptDB proxy. Since most of our schemes are efficient, the cryptographic overhead is relatively small.

5 Inference Attacks on DET-Encrypted Attributes

There are two common types of attacks on DET-Encrypted Columns, Frequency analysis and l_p-optimization attacks.

Frequency Analysis is the basic and well-known inference attack. It is used to break deterministically-encrypted columns. Given a DET-encrypted column c over C_k and an auxiliary dataset z over M_k, the attack works by assigning the ith most frequent element of c to ith most element of z. Define ψ= Hist($\mathbf{c}$) and π= Hist($\mathbf{z}$) Assumptions are that for all $i \neq j$,$\psi_i \neq \psi_j$ and $\pi_i \neq \pi_j$. In the worst-case, each tie will be broken erroneously and induce an error in the assignment. The attack is defined as:

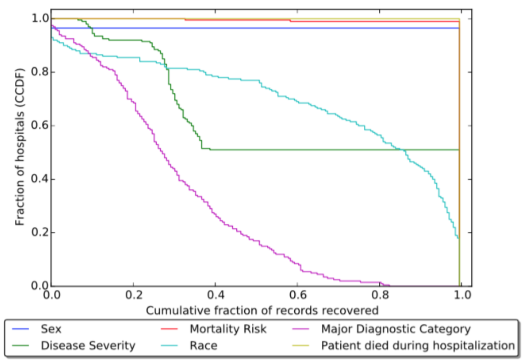
```
Frequency-An(c,z)[7]:
1:  compute $\psi \leftarrow$ vSort(Hist($\mathbf{c}$))
2:  compute $\pi \leftarrow$ vSort(Hist($\mathbf{z}$))
3:  output $\alpha : C_k \to M_k$ such that
```

$$\alpha(c) = \begin{cases} \pi[Rank_\psi(c)], & \text{if c} \in \mathbf{c} \\ \perp, & \text{if c} \notin \mathbf{c} \end{cases} \quad (1)$$

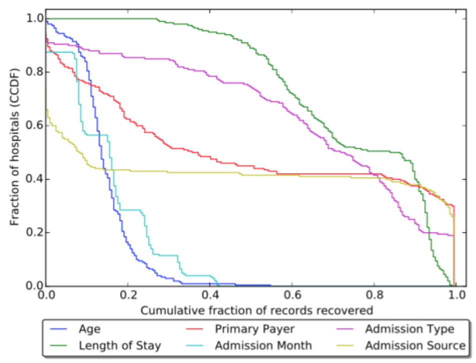
l_p-optimization is parameterized by the l_p norms. The basic idea is to minimize the total mismatch in frequencies by finding an assignment from ciphertexts to plaintexts that minimizes l_p distance between the histograms of the datasets. Given a DET-encrypted column c over C_k and an auxiliary dataset z over M_k. Define ψ= Hist($\mathbf{c}$) and π= Hist($\mathbf{z}$). It then finds the permutation matrix X that minimizes the l_p distance between the ciphertext histogram ψ and the permuted auxiliary histogram $X \cdot \pi$.The attack is defined as follows:

```
l_p-optimization(c,z)[7]:
1:  compute $\psi \leftarrow$ vSort(Hist($\mathbf{c}$))
2:  compute $\pi \leftarrow$ vSort(Hist($\mathbf{z}$))
3:  output $argmin_{X \in P_N} \|\psi - X \cdot \pi\|_p$
```

Naveed et al.[7] focused on l_p-optimization attacks on data from the National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP), which includes database includes attributes such as age, drugs, procedures, diagnosis, length of stay, etc[11]. In their experiment, subset of 1050 hospitals in the 2009 HCUP/NIS database is used. They identify whether a column is DET based by checking whether OpenEMR supported equality queries on the attribute. Frequency and l_p-optimization attacks on sex, mortality risk, admission source, major diagnostic category, age and length of stay. All the attacks take less than a fraction of a second per hospital. The accuracy of the attack is computed as the number of encrypted cells for which the recovered plaintext matches the ground truth, divided by the total number of column cells.



(a) First set of attributes



(b) Second set of attributes

Figure 5: Results of l_p-optimization on DET-encrypted columns on 200 largest hospitals[7]

Figure 5 shows the result of the attack on DET-encrypted columns on 200 largest hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data. The plot shows the empirical Complementary Cumulative Distribution Function, which means a point at location (x, y) indicates that we correctly recovered at least x fraction of the records for y fraction of the hospitals in the target data. Morality passes (0.99, 1.0), which means it is able to recover 99% of the patients for all of the hospitals. For attribute Race, at least 60% of the patients for at least 69.5% of the hospitals is gained. Admission Source for at least 90% of the patients for 38% of the hospitals, etc. According to the plot, attacks recover a substantial fraction of the encrypted columns, even for attributes with a large number of distinct values such as Age and Length of Stay. It recovered Age for at least 10% of patients for 84.5% of the hospitals. The key point for the good performance results from auxiliary information.

6 Conclusions

This tutorial paper deals with secure database techniques in Encrypted Database Systems. CryptDB is an implementation that allows query processing over encrypted databases. It provides confidentiality for applications on those systems by addressing two threats, DBMS Server Compromise and Adversary that gains complete control of application and DBMS servers. The key ideas are executing SQL queries over encrypted data, adjustable query-based encryption and chain encryption keys to user passwords. A group of SQL-aware Encryptions are widely used these systems.

RND and HOM are considered the strongest schemes, followed by SEARCH, DET, JOIN and OPE. CryptDB can support operations over encrypted data for 99.5% of the columns seen in the performance test while maintaining modest throughput penalty. CryptDB also protects most sensitive attributes with highly secure encryption schemes. The study of interference attacks shows that l_p-optimization can decrypt a large fraction of cells from DET-encrypted columns as long as appropriate auxiliary information is available.

References

[1] Mihir Bellare, Alexandra Boldyreva, and Adam ONeill. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*, pages 535–552. Springer, 2007.

[2] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam Oneill. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 224–241. Springer, 2009.

[3] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[4] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

[5] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[6] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *Annual International Cryptology Conference*, pages 482–499. Springer, 2003.

[7] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[8] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[9] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[10] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[11] Claudia Steiner, Anne Elixhauser, and Jenny Schnaier. The healthcare cost and utilization project: an overview. *Effective clinical practice: ECP*, 5(3):143–151, 2002.