Efficient Format Preserving Encrypted Databases

Prakruti C
IIIT- Bangalore
Email: prakruti@iiitb.org

Sashank Dara
Cisco Systems India Pvt. Ltd
Email: sadara@cisco.com

V. N. Muralidhara
IIIT- Bangalore
Email: murali@iiitb.ac.in

***Abstract*—We propose storage efficient *SQL-aware* encrypted databases that preserve the format of the fields. We give experimental results of storage improvements in *CryptDB* using *FNR* encryption scheme.**

Keywords Format preserving Encryption, *CryptDB*, computational privacy, block ciphers, *FNR*

I. Introduction

Cloudification of application stacks, legacy systems, databases etc. has many benefits but ensuring privacy of sensitive fields while retaining their *computability* is quite challenging. Such privacy preserving efforts involve identification of sensitive fields in the system and re-engineering such data fields in order to encrypt them. Many companies are skeptical from moving their critical applications to the cloud due to privacy issues associated with outsourcing sensitive data to third party cloud providers.

For example, a data field might be well-defined for accommodating IPv4 address of size 32 bits with number of validations throughout the application stack (and/or database) to ensure the input data is in fact IPv4 address. In order to preserve the privacy of this field using traditional AES-128 scheme it has to be re-engineered to accommodate 128 bits of memory instead of 32 bits. Also any validations that identifies the type of the data (dotted notation of IPv4 address) needs to be removed in order to accommodate a random cipher string of encrypted IPv4 address.

While the length expansion of the field needs more storage (and/or bandwidth) the removal of validations might open up serious security vulnerabilities in the form of injection attacks. The same issue is applicable while ensuring privacy of any sensitive fields like MAC addresses, Email addresses, Usernames, Account numbers, Serial Numbers, SSN, Credit Card numbers etc.

Recent advances in *SQL-aware* encrypted database systems like *CryptDB*[1] attempts to solve this problem. *CryptDB is a system that provides practical and provable confidentiality for applications that are backed by SQL databases*. It allows running most standard SQL queries over encrypted data without ever decrypting it.

CryptDB uses conventional Symmetric Encryption schemes like *AES* and *BlowFish* to encrypt the data that is to be stored in the database. Cryptographic Schemes like *AES* and *BlowFish* do not preserve the format of the plaintext after encryption. As a result of which there is a change in database field length and field types. For example, encrypting a 16 digit credit card number(of data type Integer) using *AES* produces a 128 bit ciphertext. The major drawback involved in using conventional encryption schemes like *AES* and *BlowFish* is that it adds up to the storage space required to store the encrypted data.

A. Key Contributions

In this paper we propose usage of a *Format Preserving Encryption(FPE)* to encrypt data fields in *CryptDB*. This preserves the length and formats of the plaintext upon encryption. Hence we name our implementation as *FP-CryptDB*. We provide experimental results on improvements in storage gains with minor impact on performance.

II. Preliminaries

A. Format Preserving Encryption

Format Preserving Encryption(FPE) retains the formats of the input domains upon encryption. Encryption of an IPv4 address (represented in dotted notation format) using conventional encryption techniques like AES would result in a random string but not an IPv4 address format again. Whereas encrypting using *FPE* encryption schemes would ensure that resultant cipher text is again an IPv4 address.

We use simplified definitions of *FPE* scheme without loss of generality. More rigorous definitions and proofs are available in [2]. *FPE* algorithms have additional parameters tweak t and domain d. Tweak t is similar to cryptographic salt to provide additional randomness. Domain d defines the domain in which the plain text belongs to. For example input domains could be *IPv4 address or Credit card number* etc.

The basic algorithms of an *FPE* encryption scheme are defined as below

1) $KeyGen(\sigma)$ Generates a key k, tweak t based on a security parameter σ.
2) $Encryption(p, k, t, d)$ Given a plaintext p, key k and a tweak t the algorithm produces cipher text c such that both the plaintext p and ciphertext c are in the same domain d. As shown in the Fig.4, it internally uses *rank* and *de-rank* methods along with a length preserving block cipher enc. Detailed examples of ranking functions are discussed later.
3) $Decryption(c, k, t, d)$ Given a ciphertext c, key k and a tweak t the algorithm returns the corresponding plain text p. Here both c, p are in same domain.

FNR is a arbitrary length small domain block cipher proposed in [3]. *FNR* scheme could be used to preserve the format and length of arbitrary length small domain sensitive

data like MAC addresses, IPV4(32), IPV6(32) etc. FNR uses *Pair wise Independent Permutations* along with classic *Feistel Networks* which is proven to provide additional security to its alternatives.

B. SQL Aware Encryption

SQL Aware Encryption is a technique that defines an encryption scheme for each of the pre-defined set of SQL operations like equality checks, joins, aggregates, inequality checks etc. As a result of this every data item is encrypted in a way such that it is possible to perform computations on the resultant ciphertext[1].

III. Technical Architecture

A. FP-CryptDB

The fundamental architecture of our modified version is same as *CryptDB*. We differentiate in the encryption techniques used internally as discussed in below sections.

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 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(ciphertext).

Fig.1 describes the architecture of CryptDB.

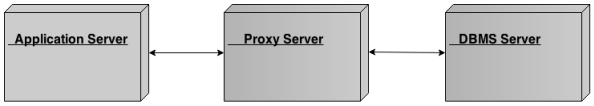

Fig. 1: *CryptDB* Architecture

Below we describe the steps involved in processing a query inside *CryptDB*.

1) 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).
2) 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.
3) The DBMS server performs computations on the encrypted data and forwards the encrypted results back to the proxy server.
4) The proxy server decrypts the encrypted query result obtained and returns it to the application server.

B. Encryption Techniques

CryptDB uses different Encryption techniques based on the operations desired by the issued query. For example if the query does not involve any equality or inequality checks then the query is encrypted using random encryption scheme(RND layer). If the query involves any SUM aggregate calculation then *CryptDB* uses homomorphic encryption scheme[4](HOM layer). For performing equality checks deterministic encryption scheme(DET layer) is used. *CryptDB* uses separate encryption schemes for performing inequality checks and joins.

In this paper we have focused on two encryption layers Random and deterministic(DET).

1) Random (RND)
   It is a probabilistic encryption scheme which means that two equal plaintexts map to two different ciphertexts. Due to this probabilistic property it provides the maximum security and is IND-CPA secure. The random encryption layer in *CryptDB* is implemented using *AES-CBC-128* for strings and *Blowfish (CBC mode)* for integers. Both these schemes use a random initialization vector(IV).
2) Deterministic (DET)
   It is a deterministic encryption scheme which means that two equal plaintexts map to the same ciphertext. Due to deterministic property this layer is less secure than random (RND) layer. The deterministic encryption layer in *CryptDB* is implemented using AES-CMC-128 and Blowfish, using a zero initialization vector. This layer is used by the server to compute select queries with equality checks, equality joins, COUNT and DISTINCT.

Fig.2 describes the various encryption layers of *CryptDB* for a simple INSERT query. As shown in the Fig.2, the columns in the query are encrypted twice using DET layer followed by encryption using RND layer. The output of `RND_int`(for a column with Integer Input type) and `RND_str`(for a column with varchar input type) is the final ciphertext that is stored in the database.

FP-CryptDB achieves both the above described properties of *Format Preserving Encryption (FPE)* schemes. The following Fig.3 describes the various onion layers of *FP-CryptDB*.

In addition to the above two described layers *CryptDB* [5] has many other onion layers like OPE(order-preserving encryption layer) to support range queries, HOM(homomorphic encryption layer)[6] [4] to support aggregation and counts, SEARCH layer performs full word keyword searches on encrypted data[7].

These additional layers aide to perform different SQL operations on the encrypted data without decrypting the data. These layers are left as is in our modified implementation *FP-CryptDB*.

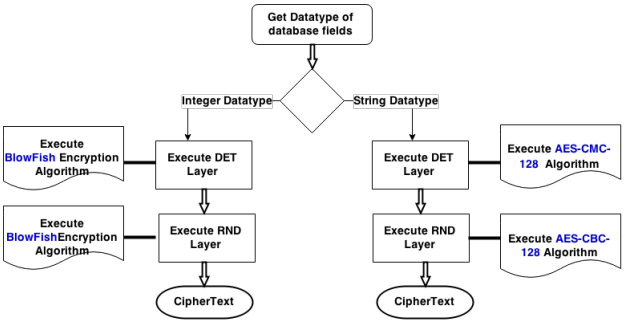

Fig. 2: Onion Layers of *CryptDB* for an SQL INSERT query

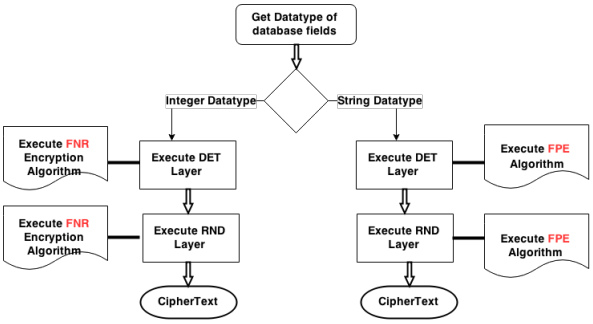

Fig. 3: Onion Layers of *FP-CryptDB* for an SQL INSERT query

IV. Implementation

A. Toy Application

We take network monitoring application as reference. The data used in our implementation is used by security analysts to monitor and analyze network traffic. Attributes of the data are *Source IP address, Destination IP address,Protocol, Number of packets, Number of Records, No of Bytes, Start date, End date, Sensor*. These fields are useful for network monitoring applications to perform traffic analysis, intrusion detection and packet filtering. For our analysis, we have taken five fields from this dataset, which are described in Fig.5.

B. Encryption of Data Types

In our modified *FP-CryptDB* we preserve the formats and lengths of the input strings. We chose *Flexible Naor and Reingold (FNR)* is length preserving block cipher for inputs 32 to 128 bits [3], [8]. We preserve the lengths and formats while encrypting IPv4 (Table I) , Time Stamps (Table II) as discussed below.

1) IPv4 Addresses: An IPv4 address is *ranked* as a 32 bits integer. The ranked integer is then encrypted using a block cipher like FNR to result in another 32 bits integer. The resultant cipher text is converted back (*de-ranked*) to dotted notation of IPv4 address in order to preserve the format. The outline of the algorithm is show in Fig.4.

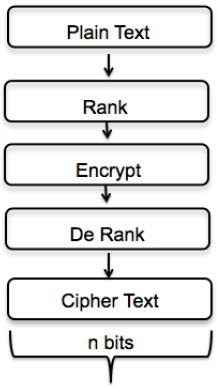

Fig. 4: Format-preserving encryption

Plain Text		Cipher Text	
Raw(Dotted)	*Ranked(Integer)*	*Raw(Integer)*	*De-ranked(Dotted)*
64.243.129.86	941079480	1226870871	73.32.144.87
56.23.187.184	2213763856	1067498731	63.160.188.235
131.243.91.16	4026531837	2739475379	163.73.19.179
239.255.255.253	905584639	2223369266	132.133.236.50
53.250.31.255	3222780570	2000079960	119.54.204.88

TABLE I: Samples for IPv4 Addresses

2) Time Stamps: To preserve the format of the input time stamp, first the date field is *ranked* to epoch value [1]. The obtained epoch value is then encrypted using *FNR* encryption scheme. The encrypted epoch value is then again is *de-ranked* back to date. The obtained date is the ciphertext and is in the same format as that of the plaintext.

3) Integers: Few fields in the application like *Port Number*, *Packets transfered* are plain integers. On encrypting integers with *FNR* scheme we get back an integer as the ciphertext. These fields do not need any special *ranking* functions. In our implementation we have used unsigned 64-bit integer data type to store the ciphertexts.

V. Experiments

A. Setup

We used Ubuntu 12.04 setup with 4GB RAM, 2.60 GHz Processor, Intel(R) Core(TM) i5-320M. We have taken reference data derived from anonymized enterprise packet header traces obtained from Lawrence Berkeley National Laboratory and ICSI [9][10]. This data covers the network traffic in

[1]*Epoch* is the date and time with reference to which a computer's clock and time-stamp values are determined. Most versions of Unix, use January 1, 1970 as the *Unix Epoch*. *Unix Time-stamp* is basically the number of seconds between a particular date and the *Unix Epoch*.

Plain Text		**Cipher Text**	
Raw(String)	**Ranked(Integer)**	**Raw(Integer)**	**De-Ranked(String)**
Date String	*Unix Timestamp*	*Unix timestamp*	*Date String*
2004/11/16T21:04:05	1100639045	1531152620	2018/07/09T16:10:20
2004/11/15T15:44:38	1100533478	627185476	1989/11/16T02:11:16
1988/01/14T09:27:05	569150825	3645016902	2085/07/03T16:41:42
2005/01/7T41:33:07	1105205587	2685279782	2055/02/03T15:03:02
2005/01/6T22:07:06	1105049226	3275728681	2073/10/20T12:38:01

TABLE II: Samples for Time stamps

particular hours on dates in late 2004 and early 2005. The data sets are in SILK flow record format. We have encrypted up to 1 million SILK flow records[11], using *FNR* encryption scheme and analyzed the performance.

B. Results

1) Storage: Our experimental results show approximately 50% [2] storage efficiency in *FP-CryptDB* for our application chosen. The storage gain is due to length preserving nature of *FNR* scheme used in *FP-CryptDB*. The database schema of our application is shown in Fig.5. Fig.6 is a graph that shows the estimated storage space for data encrypted in *CryptDB* (using *AES*) and *FP-CryptDB* (using *FNR*).

The X-axis represents the *Number of records* inserted into the database and the Y-axis represents the *Database Size* in MB.

Column Id	Field	Plaintext (char)	AES (char)	FNR (char)
1	Source ip	15	48	15
2	Destination ip	15	48	15
3	Start-Date	19	64	19
4	End-Date	19	64	19
5	Sensors	2	48	2

Fig. 5: Database Schema of *FP-CryptDB*

In our implementation, we have five VARCHAR fields in our database as shown in Fig.5 and the encrypted database also stores the salt values which is used to encrypt queries and onion layers in *CryptDB*. Therefore the estimated average row length is the sum of variable data length(size of ciphertext stored) and the total size required to store the salt value for each field.

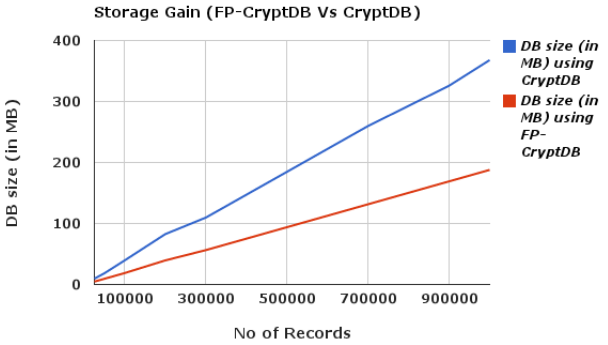

Fig. 6: Storage in MB for CryptDB and FP-CryptDB

2) Performance: The following graph shows the performance of *FNR* when compared to *AES*-128. The X-axis represents the *No of records* inserted and Y-axis represents the *time* (in milliseconds) to execute n number of SQL INSERT queries.

[2]Note that this varies based on the data fields of the application chosen.

The performance degrades in the case of FP-CryptDB as compared to CryptDB. It could be noticed that the performance degrades $y \approx 7x$ times. This is in lines with theory, FNR scheme internally uses 7 rounds of AES by design [3].

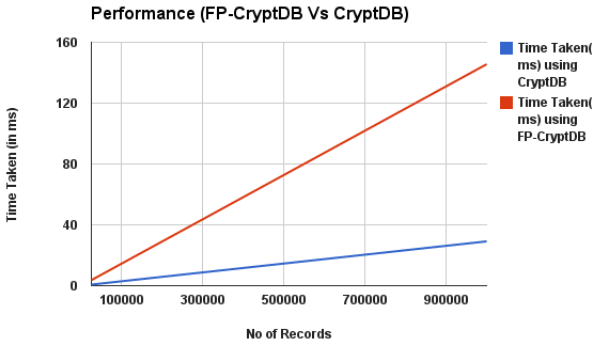

Fig. 7: Performance of *CRYPTDB* versus *FP-CRYPTDB*

VI. Conclusions

In this paper we have proposed Format Preserving Encrypted Databases *FP-CryptDB*. We took network monitoring as reference application. We provided experimental results on storage gains that could be achieved using *FPE* schemes with minor degrade in performance.

References

[1] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 85–100.

[2] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-preserving encryption," in *Selected Areas in Cryptography*. Springer, 2009, pp. 295–312.

[3] S. Dara and S. Fluhrer, "Fnr : Arbitrary length small domain block cipher proposal," in *Security, Privacy, and Applied Cryptography Engineering*. Springer, 2014.

[4] P. Paillier, in *Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 1999.

[5] Y. A. Boldyreva, N. Chenette and A. O'Neill, "Order-preserving symmetric encryption," in *Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. (EUROCRYPT), 2009.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices." in *STOC*, vol. 9, 2009, pp. 169–178.

[7] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 44–55.

[8] https://github.com/cisco/libfnr.

[9] https://tools.netsa.cert.org/silk/referencedata.html.

[10] http://www.icir.org/enterprise-tracing/Overview.html.

[11] T. Shimeall, S. Faber, M. DeShon, and A. Kompanek, "Using silk for network traffic analysis," tech. rep., CERT Network Situational Awareness Group, Tech. Rep., 2010.