Improving cloud based encrypted databases

Eduardo Lopes Cominetti[1], **Marcos Antonio Simplicio Junior**[1]

[1]Departamento de Engenharia de Computação e Sistemas Digitais (PCS)
Escola Politécnica – Universidade de São Paulo (USP)

{ecominetti,mjunior}@larc.usp.br

Abstract. *Databases are a cornerstone for the operation of many services, such as banking, web stores and even health care. The cost of maintaining such a large collection of data on-premise is high, and the cloud can be used to share computational resources and mitigate this problem. Unfortunately, a great amount of data may be private or confidential, thus requiring to be protected from both internal and external agents. Moreover, this data needs to be manipulated to provide useful information to its owner. In this dissertation, we propose modifications to CryptDB, a state-of-the-art encrypted cloud database, aiming to enhance its efficiency, flexibility and security; this is accomplished by improving or changing its underlying cryptographic primitives.*

1. Introduction

A database is a usually large collection of data organized especially for rapid search and retrieval [Merriam-Webster 2017]. Databases are used in processes that need to correlate information, such as maintaining and operating a web store, which requires the association of a person, his/her address, a purchase, and a payment method to process a sale.

Generally, a database is stored on-premise, which means that its owner is responsible for providing the infrastructure and for maintaining the system. However, on-premise costs for large databases are quite high [Buckel 2013]. In comparison, using the cloud to store a database can help mitigate the costs by sharing resources among different companies [Roggero 2013]. Unfortunately, though, the data stored may be private or confidential, which is the case for credit card numbers or medical history, for example. Therefore, it must be protected from internal and external agents. As a result, privacy-preserving cloud databases become essential to the deployment of confidential data in the cloud environment.

1.1. Motivation

Security and privacy concerns remain among the major cornerstones for the widespread adoption of cloud solutions [Oracle 2015, Schulze 2016]. These concerns are legitimate, since the number of online attacks that try to recover confidential data is considerable: only in the United States, more than 9 million records were compromised since 2005 [Privacy Rights Clearinghouse 2017].

Data must be encrypted to prevent its disclosure in case of attacks. However, it is not always possible to use traditional encryption schemes in databases, as then the encrypted data cannot be manipulated and correlated without decryption. This limitation of traditional schemes brings forward a challenge: is it possible to encrypt data and still be able to compute on it without decryption?

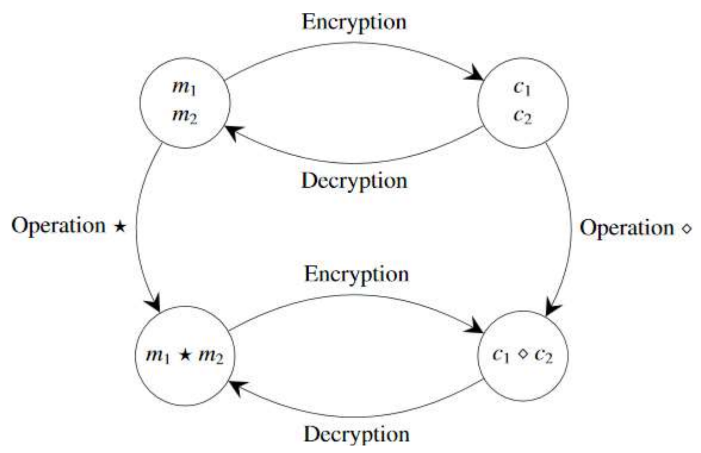

Figure 1. Homomorphic Encryption: morphism between operation $\star$ performed in plaintexts m_1 and m_2 and operation $\diamond$ performed in ciphertexts c_1 and c_2.

In 1978, Rivest, Adleman, and Dertouzos proposed a class of special encryption functions they called "privacy homomorphisms" [Rivest et al. 1978]. These special encryption functions allow encrypted data to be operated without decryption. Figure 1 presents a visual reference of homomorphic encryption. Given two plaintexts m_1 and m_2 and their encryptions c_1 and c_2, there is an operation $\diamond$ performed on the ciphertexts that is equivalent to an operation $\star$ performed on the plaintexts. In other words, the decryption of $c_1 \diamond c_2$ is equal to $m_1 \star m_2$. If they can be used on a database, the underlying data can be protected and operated on the cloud without revealing classified information. For instance, an addition of multiple rows can be simply achieved by homomorphically adding these rows. Furthermore, convoluted database operations can be expressed as logic gate digital circuits. Hence, if there is an homomorphic encryption that allows any operation to be performed, it can be used to create logic gates and any database operation can be performed homomorphically. Following this concept, many Partially Homomorphic Encryption (PHE) schemes that allow one function to be computed over encrypted data were proposed. Unfortunately, schemes that allow any operation to be performed on encrypted data, called Fully Homomorphic Encryption (FHE), are still impractical for real world applications [Bajaj and Sion 2011].

FHE's poor performance compels cloud database developers to adopt new and creative strategies. One of such strategies is to use a collection of different encryption functions, each one allowing a specific database operation to be performed. An example is CryptDB, a cloud database designed by MIT in 2011 [Popa et al. 2011]. It uses a myriad of encryption schemes and modes to permit the database to operate on data without publicly exposing it. As the solution relies on many different algorithms, overall database performance, functionality and even security is greatly affected by their individual behaviour. Thus, the study, improvement and modification of these algorithms is essential to enhance cloud databases and make them a better alternative to on-premise systems.

1.2. Goals and Contributions

Our main goal is to improve the cryptographic schemes used on privacy preserving cloud databases in order to enhance their security, functionality and efficiency. To accomplish this, we use CryptDB as a basis, for it is considered one of the state-of-the-art privacy preserving cloud databases and its framework and source code are publicly available. In this dissertation, we:

1. enhanced CryptDB's homomorphic layer performance by up to 1300 times through the replacement of the layer's algorithm by a novel PHE scheme developed by us presented in Section 3, at the cost of additional storage space. This algorithm was later published on IEEE Transactions on Information Forensics and Security [Cominetti and Simplicio 2020];
2. enhanced CryptDB's deterministic layer performance by up to 7.4 times through the replacement of the layer's algorithm by a hash function, at the cost of additional storage space;
3. enabled wildcard search, thus expanding the system's functionality, through the modification of the CryptDB's search layer algorithm, at the cost of additional storage space and additional performance overhead; and

1.3. Outline

The rest of this paper is organized as follows. Section 2 briefly presents the CryptDB database system, the basis state-of-the-art privacy preserving cloud database used in this work. Section 3 briefly introduces a novel symmetric PHE, created specifically for the scenario of privacy-preserving cloud databases. Section 4 presents our modifications to CryptDB. Finally, we present our concluding remarks in Section 5.

2. Related Works

Although there are some solutions for privacy-preserving cloud databases (e.g., [CipherCloud 2015, Egorov and Wilkison 2016]), we focus our attention on the CryptDB system.

CryptDB [Popa et al. 2011] is an encrypted database designed by MIT. CryptDB uses a (User)-(Secure Proxy)-(Server) framework, so that (1) the user interacts with the secure proxy as if the proxy was a plain database, and (2) the secure proxy is responsible for encrypting data and storing it on the server. The data encryption by the secure proxy in done in "onion layers". There are multiple onions for each data and each onion has multiple layers. Data is encrypted from "more information revealing" layers to "less information revealing" layers. Each layer uses a different algorithm and is responsible for a specific database operation. Moreover, CryptDB's secure proxy is responsible for data encryption, storage of keys and conversion of a plain SQL query provided by the user to an encrypted query sent to the server.

Improvements to CryptDB's security, functionality and efficiency can be achieved by changing individual layer algorithms, since each layer deals with specific SQL operations. This modularity is an important feature of CryptDB, since it allows a more structured analysis when looking for improvement opportunities. In this work, we performed changes in 3 specific layers: the Deterministic layer (DET), the Search layer (SEARCH), and the Homomorphic Addition Layer (HOM).

DET provides the equality check functionality. It enables the computation of selects with equality operators, GROUP BY, COUNT, DISTINCT, among others. This layer uses a pseudorandom permutation (PRP) in CMC mode [Halevi and Rogaway 2003] with a zero Initialization Vector (IV). The outcome is a deterministic encryption. This allows the equality check between values without exposing the unencrypted value.

SEARCH allows a word to be searched on the database without revealing it. This layer enables the SQL operand LIKE in the database. It uses the SWP encryption scheme [Song et al. 2000].

Finally, HOM permits the sum of values without first decrypting them. This allows the database to perform the SQL SUM operation and averages. The layer is implemented by the Paillier algorithm [Paillier 1999].

3. New Fast Additive Partially Homomorphic Encryption

To improve CryptDB's HOM efficiency, we present two novel additive, partially homomorphic encryption schemes built upon the Approximate Common Divisor (ACD) Problem [Howgrave-Graham 2001]:

Definition 1: The Approximate Common Divisor (ACD) Problem
Let p be a prime number of size η, in bits. Let q be an integer in the interval $[0, 2^{\gamma}/p)$, where the γ parameter gives the number final size. And let r be a positive or negative random noise whose size in bits is defined by the ρ parameter. Define the efficiently sampleable distribution $\mathcal{D}_{\gamma,\rho}(p)$ as

$$\mathcal{D}_{\gamma,\rho}(p) = \{p \cdot q + r \mid q \leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p), r \leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho})\}.$$

The ACD problem consists in computing p from polynomially-many samples x_i drawn from $\mathcal{D}_{\gamma,\rho}(p)$.

We name them Fast Additive Homomorphic Encryption (FAHE) 1 and FAHE2. One of the main particularities of the proposed solutions, which enable relevant simplifications and optimizations, is that they rely on symmetric keys for data encryption and decryption. Hence, on one hand, just a trusted party can encrypt and decrypt data. Homomorphic additions, on the other hand, can be performed very efficiently by any entity (e.g., cloud servers).

In a nutshell, FAHE1 and FAHE2 are symmetric probabilistic encryption algorithms that use a prime number as private key. Both schemes rely on the ACD as underlying security problem. However, whereas FAHE1 is a simple application of the ACD, FAHE2 provides shorter ciphertexts but requires slightly stronger security assumptions. A detailed description of these algorithms is available at [Cominetti and Simplicio 2020].

The main idea for FAHE1 is to use the ACD problem and append the message m to be encrypted at the end of the noise `noise`, before adding the result to $p \cdot q$. Since the resulting string containing the message m and `noise` remains smaller than the η-bit prime p, the corresponding plaintext can be recovered via modular reduction, during the decryption procedure. Figure 2 illustrates FAHE1's ciphertext structure.

For FAHE2's design, the basic idea is to create an open space at a given position `pos` inside the noise employed in the ACD problem. Then, we embed the message in that position before adding the result to $p \cdot q$. As a result, the difference between the noise size ρ (composed of `noise1`, `noise2`, and the message) and the key size η is smaller than in

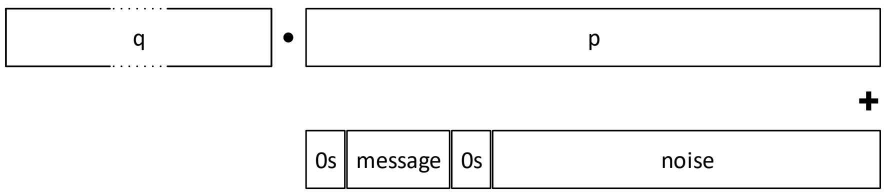

Figure 2. A visual description of FAHE1's encryption process.

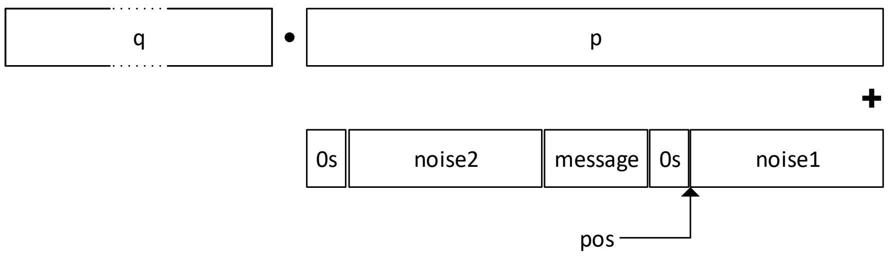

Figure 3. A visual description of FAHE2's encryption process.

FAHE1. Consequently, as the ciphertext size γ is dependent of this difference, it is also smaller than in the previous variant. Figure 3 illustrates FAHE2's structure.

Table 1 presents the parameters size in bits for FAHE1 and FAHE2, messages of 32 and 64 bits, security parameter of $\lambda = 128$ and $2^{\alpha-1}$ allowed additions.

4. Modifications to CryptDB

The main result of this dissertation is the modification performed on CryptDB's HOM. We propose the substitution of the Paillier Encryption (PHPE) on HOM by FAHE2.

HOM uses PHPE to provide additive homomorphism in the encrypted database values. PHPE's underlying security problem requires a public key n of size similar to RSA. For a desired security level $\lambda = 128$ bits, $n = 3072$ [Barker and Dang 2016, Table 2]. Since PHPE uses the square of the key, n^2, to operate, the ciphertext final size is 6144 bits. As a result, the encryption and decryption require modular exponentiations and multiplications over a large cyclic group. Moreover, PHPE is not secure against quantum computer attacks, as the integer factorization of n allows the computation of the private key [Shor 1997].

In opposition, our algorithms allow considerable speed ups in every process (key generation, encryption, decryption, and homomorphic sum). These speed ups are presented in Table 2. The gains are computed using a message size of 64 bits. Furthermore,

Table 1. Parameters for FAHE1 and FAHE2 for $\lambda = 128$ (in bits).

	FAHE1				FAHE2			
	$\lvert m_{max}\rvert = 32$		$\lvert m_{max}\rvert = 64$		$\lvert m_{max}\rvert = 32$		$\lvert m_{max}\rvert = 64$	
	$\alpha = 6$	$\alpha = 33$	$\alpha = 6$	$\alpha = 33$	$\alpha = 32$	$\alpha = 33$	$\alpha = 29$	$\alpha = 33$
ρ	128	128	128	128	192	193	221	225
η	172	226	204	258	224	226	250	258
γ	35402	175616	105619	309029	25921	27683	23866	31359

Table 2. FAHE2 results (in cycles) compared to Paillier at the same (pre-quantum) $\lambda = 128$ security level.

Process	$\|m_{max}\| = 32$ $\alpha = 32$	$\|m_{max}\| = 64$ $\alpha = 33$	Pailler	Gain (Paillier/FAHE2)
KeyGen	17651582	23832773	2253611712	94.56
Enc	254619	294839	351458572	1192.04
Add	1820	2384	211829	88.85
Dec	198096	262719	347251790	1321.76

Table 3. AES-CMC and peppered SHA2 results (in cycles).

		AES-CMC	Peppered SHA2
Hiding (entire dictionary)		697580460	93961082
Search (single entry)	Positive	2110	444
	Negative	2018	290

our schemes are based on the Approximate Common Divisor (ACD) problem, believed to be resistant to quantum computer attacks.

We also modified CryptDB's DET by substituting the CMC mode by a simple hash with a pepper. A pepper is a constant secret string concatenated with the hash's input. This modification enhanced DET's performance.

Since a hash output is different for every different input, even plaintexts that share all bits but one have different hash values. The pepper's function is to be an affix to randomize the hash's output. Because the pepper is a secret, an attacker cannot build a lookup table to discover the hash input. Additionally, an attacker cannot recover a secure hash preimage, by definition. Also, the hash's output has a fixed length, preventing inference attacks due to the database entry size.

As a result, we improved DET's performance. As presented in Table 3, it is 7.4 times quickker to hide a 49057 entries dictionary and up to 6.95 times faster to perform a match operation. The drawback of the method is that the peppered SHA2 method cannot be decrypted and needs an additional layer on the database. The additional layer adds 256 bits for each hidden row. To decrypt data using our method, another onion of CryptDB must be used.

Finally, we present a modification to SEARCH. Currently, SEARCH is able to search only for full words. In a database, a fragment word search is desirable, as the user may want to look for data patterns. To allow word fragments to be searched, we propose a modification on how data is encrypted by SWP.

Instead of encrypting the whole word, we propose that the user chose a token size. This token size is the fragment word size to be encrypted. The word will be divided in multiple tokens. Appended to the token, the position of the fragment in the word is added, forming the final plaintext to be encrypted. The last token also has a second copy with the special terminator, $\vdash$. Moreover, if the last token is smaller than the token size, the final token repeats part of the previous fragment to reach the token size. For instance, the word "cryptography", with a token size 2, will be divided in the fragments "cr1", "yp2",

Table 4. Traditional and modified Search results (in cycles) to encrypt a 49057 entries dictionary, to generate the search token and to search all encrypted entries.

	Traditional	Modified
	(Full words)	(Single character, fixed position)
Encrypting (entire dictionary)	2486218658	59039053083
Generate search token	9140	8420
Search (all encrypted entries)	1366376385	21827182139

"to3", "gr4", "ap5", "hy6", and "hy⊢". The word "Alice", with the same token size, will be divided into "Al1", "ic", "ce3", and "ce⊢". These fragments will be then encrypted instead of the whole word.

When the user searches for a fragment, the proxy rewrites the search query and adds the appropriate positions to the fragment. One search query may have to be rewritten as many times as there are possible positions for the fragment. This modified query is then used to perform the search.

Unfortunately, the solution has drawbacks and one limitation. The fragment word search is still limited to fragments of a giving size. Moreover, if the fragment is partially in one token and partially in other (e.g. searching for the fragment "ry" in our example), this result will not be produced. Because of this, the wildcard operator is not fully enabled. To fully enable the wildcard operator, the token size must be made equal to 1. This brings forth the important drawback of our solution.

The major drawback of our solution is that it increases the ciphertext size. Since each fragment must be individually encrypted to allow its search, there is a ciphertext expansion compared to the unmodified protocol. Therefore, the performance of a search decreases as the token size is made smaller. An example with toke size 1 is presented in Table 4.

5. Conclusion

In this dissertation, we presented methods to improve the CryptDB system, a cloud based encrypted database. We accomplished this by: (1) designing a new partially additive homomorphic algorithm which is more efficient than the current state of the art used on CryptDB; (2) exchanging the deterministic layer's AES-CMC algorithm for the SHA2 hash function to improve the layer's performance; (3) modifying the search layer's SWP algorithm to allow wildcard searches to be executed, adding a new functionality to the system.

According to our experimental results and analysis, all our solutions are satisfactory. Solutions 1 and 2 are able to improve the system performance by increasing the system storage. Additionally, solution 3 enables wildcard search if the size of the words to be stored are decreased to a still reasonable value.

As a direct result from this work, solution 1 was published on IEEE Transactions on Information Forensics and Security[Cominetti and Simplicio 2020].

The full version of this dissertation is available at https://doi.org/10.11606/D.3.2019.tde-29052019-072659.

References

Bajaj, S. and Sion, R. (2011). Trusteddb: A trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 205–216, New York, NY, USA. ACM.

Barker, E. and Dang, Q. (2016). Nist special publication 800–57 part 1, revision 4.

Buckel, C. (2013). The real cost of enterprise database software. Accessed April 3, 2017.

CipherCloud (2015). Guide to cloud data protection.

Cominetti, E. L. and Simplicio, M. A. (2020). Fast additive partially homomorphic encryption from the approximate common divisor problem. *IEEE Transactions on Information Forensics and Security*, pages 1–1.

Egorov, M. and Wilkison, M. (2016). Zerodb white paper. *arXiv preprint arXiv:1602.07168*.

Halevi, S. and Rogaway, P. (2003). A tweakable enciphering mode. In *Annual International Cryptology Conference*, pages 482–499, Santa Barbara, CA, USA. Springer.

Howgrave-Graham, N. (2001). Approximate integer common divisors. In *CaLC*, volume 1, pages 51–66, Providence, RI, USA. Springer.

Merriam-Webster (2017). Database. Accessed March 29, 2017.

Oracle (2015). Cloud computing comes of age.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology - EUROCRYPT'99*, pages 223–238, Prague, Czech Republic. Springer.

Popa, R. A., Redfield, C., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, Cascais, Portugal. ACM.

Privacy Rights Clearinghouse (2017). Chronology of data breaches. Accessed March 29, 2017.

Rivest, R., Adleman, L., and Dertouzos, M. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180.

Roggero, H. (2013). Sample pricing comparison: On-premise vs. private hosting vs. cloud computing. Accessed March 29, 2017.

Schulze, H. (2016). Cloud security spotlight report.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Berkeley, CA, USA. IEEE.