NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF GRADUATE STUDIES
SPECIALIZATION: INFORMATION AND COMMUNICATIONS TECHNOLOGIES

MASTER THESIS

Advanced Cryptographic Techniques For Database Security

Alexios T. Katsadouris

Supervisor: **Konstantinos Limniotis,** External Professor

ATHENS

SEPTEMBER 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΙΔΙΚΕΥΣΗ: ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προηγμένες Τεχνικές Κρυπτογράφησης Για Ασφάλεια Βάσεων Δεδομένων

Αλέξιος Θ. Κατσαδούρης

Επιβλέπων: **Κωνσταντίνος Λιμνιώτης,** Εξωτερικός Διδάσκων

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

MASTER THESIS

Advanced Cryptographic Techniques For Database Security

Alexios T. Katsadouris
A.M.: M1579

SUPERVISOR: **Konstantinos Limniotis, External Professor**

September 2021

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προηγμένες Τεχνικές Κρυπτογράφησης Για Ασφάλεια Βάσεων Δεδομένων

Αλέξιος Θ. Κατσαδουρης
Α.Μ.: M1579

ΕΠΙΒΛΕΠΩΝ **Κωνσταντίνος Λιμνιώτης,** Εξωτερικός Διδάσκων

Σεπτέμβριος 2021

ABSTRACT

There is a modern need to store information in databases to external providers, while maintaining integrity and security. This paper is a review of the issue concerning data security and the solutions that have been proposed and implemented so far. More specifically, to address the problem, various systems have been proposed at a theoretical and practical level. These systems, by combining different encryption methods, succeed to address the problem to some extent, by encrypting the exchanged and stored data. Moreover, in these systems, the correct execution of several SQL query clauses on encrypted databases is achieved. Firstly, the existing encryption methods that can be used to create complex systems are listed and discussed. These methods are commonly accepted as they are currently used in various implementations outside the scope of our study. Then, we mention systems that have been generated at a theoretical level as well as all the analysis that have been done in terms of performance, complexity of SQL query execution and resilience against security attacks. Furthermore, we provide studies that have taken advantage of these methodologies and have been implemented on a practical level. In addition, we discuss the use and application of these systems in practice, along with their various configurations, their analysis at security levels, performance and complexity in SQL query execution. Finally, we refer to possible extensions of these systems as well as possible new implementations that can take advantage of the existing methodologies.

SUBJECT AREA: Database Cryptography

KEYWORDS: security, cryptography, database, SQL, database

ΠΕΡΙΛΗΨΗ

Είναι γεγονός ότι υπάρχει μια σύγχρονη ανάγκη αποθήκευσης πληροφοριών σε βάσεις δεδομένων σε εξωτερικούς παρόχους, με τη διατήρηση της ακεραιότητας και της ασφάλειας τους. Η παρούσα διπλωματική εργασία αποτελεί μια ανασκόπηση πάνω στο ζήτημα της ασφάλειας των δεδομένων και τις λύσεις που έχουν κατά καιρούς προταθεί και εφαρμοστεί. Πιο συγκεκριμένα, για την αντιμετώπιση του προβλήματος έχουν προταθεί, σε θεωρητικό και πρακτικό επίπεδο, διάφορα συστήματα, τα οποία, συνδυάζοντας διάφορες μεθόδους κρυπτογράφησης, κρυπτογραφούν τα δεδομένα που ανταλλάσσονται και αποθηκεύονται, επιτυγχάνοντας, έως έναν βαθμό, να αντιμετωπίσουν το πρόβλημα. Επίσης, μέσω αυτών των συστημάτων επιτυγχάνεται η σωστή εκτέλεση αρκετών SQL query clauses πάνω σε κρυπτογραφημένες βάσεις. Αρχικά, παρατίθενται και συζητούνται οι υπάρχουσες μέθοδοι κρυπτογράφησης οι οποίες μπορούν να χρησιμοποιηθούν για τη δημιουργία περίπλοκων συστημάτων. Οι μέθοδοι αυτές είναι κοινώς αποδεκτές καθώς χρησιμοποιούνται σήμερα σε διάφορες υλοποιήσεις εκτός του σκοπού της μελέτης μας. Στη συνέχεια, γίνεται λόγος για συστήματά που έχουν συσταθεί σε θεωρητικό επίπεδο και παρατίθενται αναλυτικά όλες οι αναλύσεις που έχουν γίνει σε επίπεδο χρόνου εκτέλεσης, πολυπλοκότητας εκτέλεσης πάνω σε SQL queries και ανθεκτικότητας ενάντια σε επιθέσεις ασφαλείας. Έπειτα, ακολουθεί η παράθεση μελετών σε συστήματα που έχουν εκμεταλλευτεί τις μεθοδολογίες αυτές και έχουν υλοποιηθεί σε πρακτικό επίπεδο. Επιπρόσθετα, γίνεται αναφορά στη χρήση και εφαρμογή αυτών των συστημάτων στην πράξη, μαζί με τις διάφορες παραμετροποιήσεις τους, καθώς και στις αναλύσεις τους σε επίπεδα ασφαλείας, χρόνων εκτέλεσης και πολυπλοκότητας σε SQL queries. Τέλος, γίνεται αναφορά σε πιθανές επεκτάσεις των συστημάτων αυτών καθώς και σε πιθανά νέα συστήματα που μπορούν να εκμεταλλευτούν τις υπάρχουσες μεθοδολογίες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κρυπτογράφηση Βάσεων Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ασφάλεια, κρυπτογραφία, βάσεις δεδομένων, SQL

CONTENTS

LIST OF FIGURES

FOREWORD

The thesis was created during the master educational level on Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. The context of the thesis is composed primarily of publications and projects available online. During the study we also used the aid of virtualization technology by taking advantage of already created Docker images.

1. PRIVACY ISSUES IN MODERN APPLICATIONS

1.1 Defining the problem

Nowadays with the emergence of the “Software as a Service” model for enterprise computing has introduced several challenges to overcome with one of the most important this of data privacy. “Database as a Service” model gives the ability to everyone who has access to the internet to retrieve or alter data from anywhere. This creates several major privacy issues. For example, the data stored by customers on the services providers sites need to be protected from security leaks. Also, the services providers themselves need to be able to provide some advanced level of security and trust to the data owners for the use of their products. There are a lot of studies that their approach focuses on the theoretical and practical techniques in order to tackle the aforementioned privacy problems. These techniques are based on the execution of SQL queries on encrypted data with the prime objective to execute as much as possible of those on the client site without having to decrypt the actual data. There are several encryption methodologies explored based on the algebraic framework as well as their practical implementations and their corresponding results.

In the database-service-provider model the data are hosted on the premises of the database providers. Using this model, organizations provide their customers with hardware and software solutions without the need for them to develop their own. It is self-explanatory for most organizations that data are essential for their business flow. As a result, organizations need to provide sufficient security measures to guard data privacy. H. Hacigümüs [1] in his study stated that hardware encryption is superior to software encryption. Bulk data encryption had a significant positive impact on per-byte encryption cost. Also, row encryption was found preferable to field encryption. Moreover, in order to achieve data privacy in is stated that the whole data may not be decrypted at the provider site. A viable approach is to transmit the needed encrypted tables from the server to the client and then decrypt the tables and execute the query on the client. Unfortunately, this approach mitigates almost every advantage of the service-provider-model because now primary data processing needs to occur on client machines.

The system as shown in the aforementioned work consists of three primary entities. A user that poses the query to the client. A server that is hosted by the service provider who stores the encrypted database. The encrypted database is augmented with additional information allows certain amount of query processing to occur at the server without jeopardizing data privacy. Finally, a client that stores the data at the server. The client also maintains metadata for translating user queries to the appropriate representation on the server and performs post-processing on server query results. The basic idea here is that based on the auxiliary information stored, there are certain developed techniques that allow us to split an original query over unencrypted relations, into a corresponding query over encrypted relations to run on the server and a client query for post-processing results of the server query. At first, for achieving this goal an algebraic framework has been developed that enabled query rewriting over encrypted representation. Then, the performance of various manifested strategies was tested over numerous queries and the results shown that privacy from service providers can be achieved with reasonable overheads establishing the feasibility of the model. Several people managed to use the methodology H. Hacigümüs proposed in his paper [1] in order to develop custom constructions on both theoretical and practical level [4].

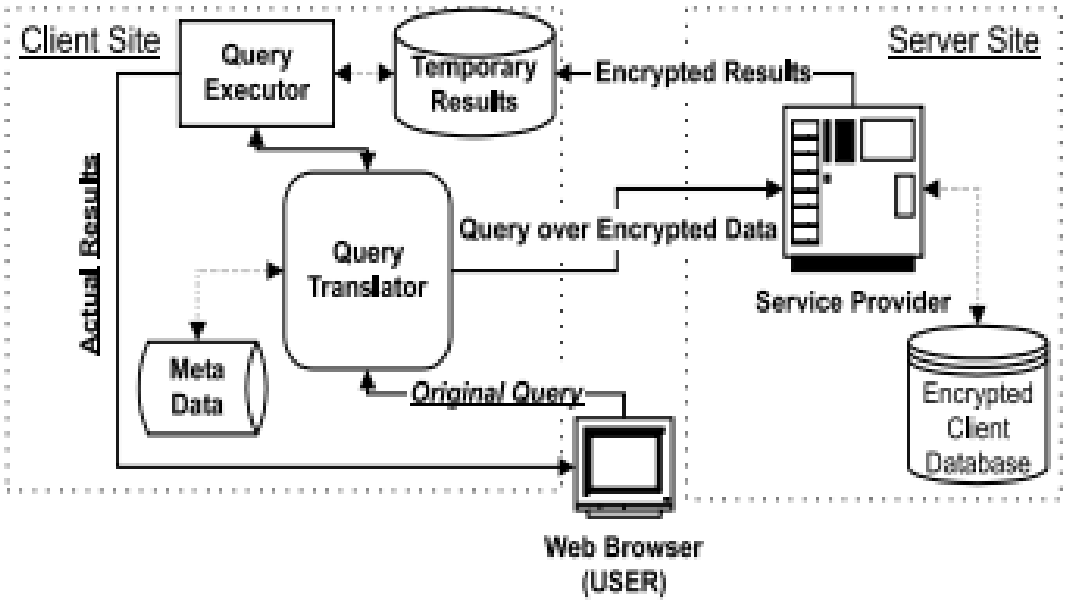


Figure 1 Service Provider Model [1]

1.2 Overview of the Study

The purpose of this study is the research and investigation of the possible theoretical and practical solutions with respect to the matter of privacy and integrity of data stored and exchanged on service providers. Also, we focus on all the various encryption methods that can be used to form complex constructions, the constructions themselves as well as their assets, flaws and possible future enhancements. As it poses a large problem now and will surely continue to pose a problem in the future much work needs to be done on the specific field.

In the chapter that follows we explore the most important encryption schemes that are widely used nowadays. These encryption schemes, when used in conjunction, they can protect the database servers against possible attacks. Their usage can be beneficial as they have already been tested separately in several application systems. In the third chapter we explore in depth the custom theoretical and practical constructions that have been proposed to work on the service provider model. We also inspect the construction designs, their advantages and disadvantages concerning performance, information leakage and complexity. In the fourth chapter we study the custom generated system created using the custom construction and then we review their published the results. Finally, we cite a summary of all the matters discussed in the paper. We also discuss further research that is needed on the field.

2. WELL-DOCUMENTED DATA SECURE METHODS

There are plenty of encryption methodologies that can be used to achieve database encryption over unencrypted queries. The advantage of this kind of encryption methodologies is that they obscure data, even after a breach, and satisfy privacy regulations. It is also important to mention that they can obstruct application performance, especially when applied to data in cloud services. A lot of companies nowadays have internal policies or regulatory compliance standards that require data to be encrypted, with encryption keys managed by the company, rather than external providers before they send them elsewhere. Security experts need to look for encryption schemes with strong data protection features. No encryption scheme can be perfect as it leaks some data in the process. This enables attackers to find plaintexts from ciphertexts by using various techniques. Below are mentioned some scheme approaches that are used and tested in various models, their corresponding strengths and weaknesses as well as:

2.1 Regular Key Encryption

As T. Ristenpart states in his article [7] Regular Key Encryption goals are confidentiality, data integrity, and sender authenticity. It provides the ability to hide all critical information about the data. Schemes can also provide data integrity and sender authenticity, meaning an attacker cannot create a valid ciphertext or modify a legitimate ciphertext without the user noticing. It should be used for any data that requires the highest security, even at the price of losing search and other functionality. The way this encryption scheme works is very simple. Using an encryption algorithm, the plaintext is converted in an unreadable ciphertext and then it is sent over a communication channel. Because the message is encrypted attackers cannot read it. When the ciphertext is received, it is decrypted using a decryption algorithm and then the receiver can read it. Also, Regular key encryption distinguishes itself in two technique types. Symmetric and Asymmetric Key Encryption.

As mentioned by A. Singhal in his article [8] in Symmetric Key Encryption both the receiver and the sender use the same encryption key. The sender uses the key to

encrypt the data and then the receiver uses it to decrypt them and be able to read their content. What is good about this method of encryption is that it is very efficient, and it take less time to encrypt and decrypt each message. On the other hand, the number of keys required is very large. Also, the sharing of the common key between the sender and receiver is of critical importance because an attacker might intrude in the process. The most common encryption algorithm that is being used in Symmetric Key Encryption is Advanced Encryption Standard (AES).

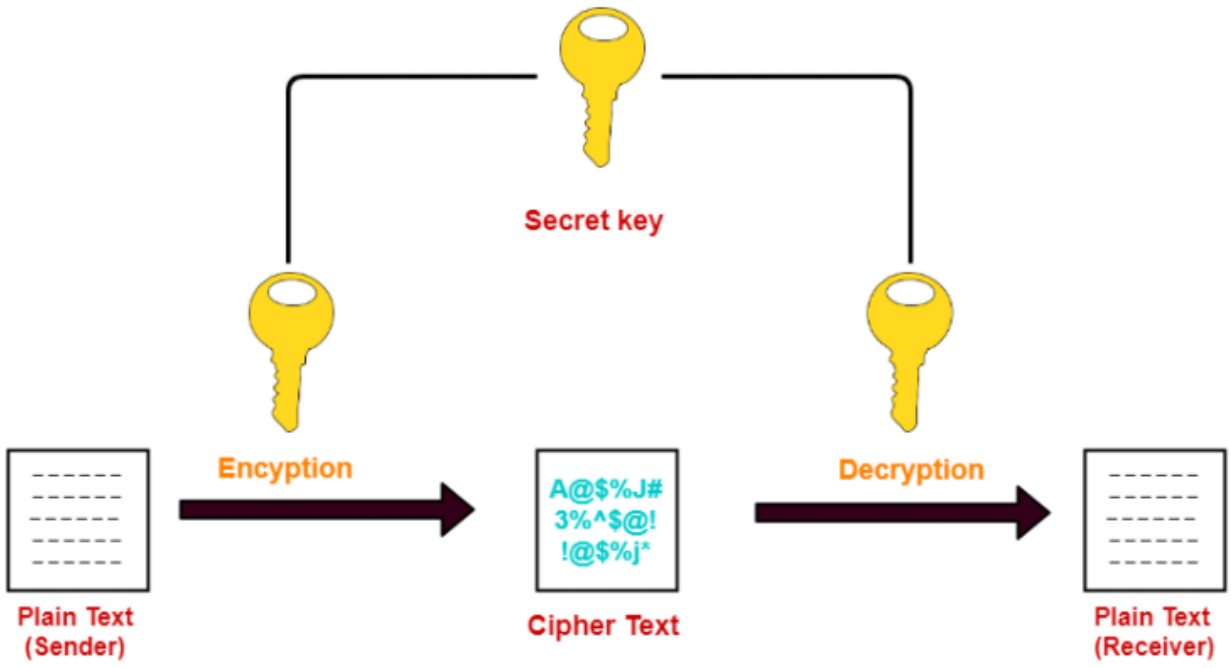


Figure 2 Symmetric Encryption Example [8]

As mentioned by A. Singhal in his article [9] in Asymmetric Key Encryption communication parties use different keys to encrypt and decrypt the data, a Public Key and a Secret Key (that's why they are also being called as public key encryption schemes). The receiver issues the two keys and then send the Public to the sender. In general, the Public Key of the receiver, as its name implies, can be publicly available to everyone. The Sender then encrypts the plain data using the recipient's Public Key and then sends the encrypted data. The produced ciphertext can only be decrypted using the Private Key. By using the Public Key, it is not possible for attackers to figure the receiver's Private Key. Asymmetric key is more vigorous method as well as less susceptible to third party security breach attempts. However, it requires more computational power, and it is slower than Symmetric Key Encryption. Two of the most common encryption algorithms that are used in Asymmetric Key Encryption are the RSA Algorithm and the Diffie-Hellman Key Exchange technique.

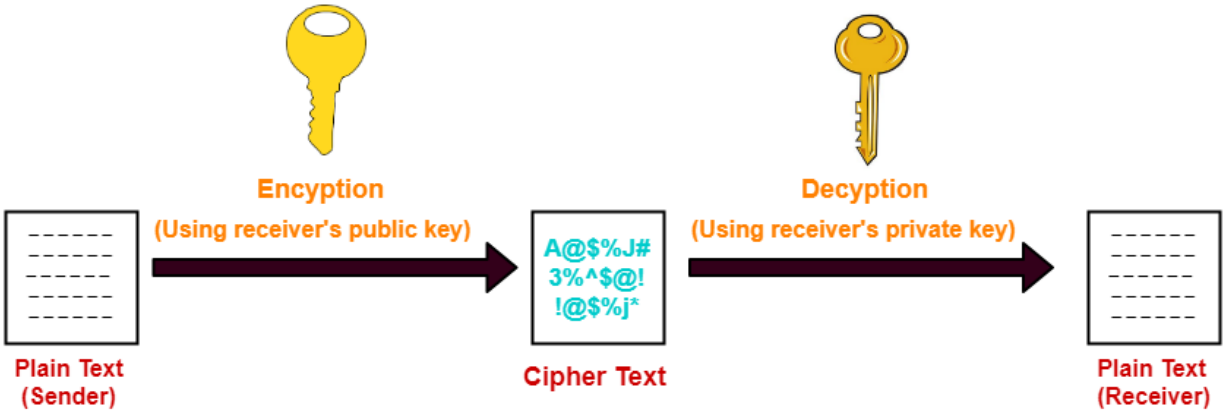


Figure 3 Asymmetric Encryption Example [9]

2.2 Searchable Key Encryption

As stated by R. Curtmola, J. Garay and S. Kamara in their paper [3], Searchable Symmetric Key Encryption (SSE) allows for the ability to selectively search over encrypted data that are stored into a service provider. For a secure and efficient data retrieval it has to be ensured that a search over encrypted data can be performed without revealing the contents and the search keyword to the server. In this scheme the data owner generates and encrypts the data and then store them inside a datasource of a Service Provider, in a way that data users are given searching capabilities on them over an application. The data owner can be either a company or an individual. The application users send queries to the datasource by using the data owner's application and are able to get unencrypted results over a completely encrypted datasource. There are certain privacy requirements which are to be met while creating a searchable encryption scheme. For instance, the Service Provider cannot maintain any knowledge about the exchanged data or the performed queries. Searchable Key Encryption can have two approaches, Symmetric and Asymmetric.

In Symmetric Searchable Encryption (SSE) [16] users use symmetric private key encryption schemes to encrypt the data before outsourcing the to the Service Provider. This approach is preferable when the data owner wants also to search over them. This approach is very efficient because it is based on symmetric primitives like block-ciphers and pseudo random functions thus requiring less computational power. On the other hand, it can only be used for a single user scenario in order to remain a secure approach, so it is not preferable for multiple user scale. Also, this approach leaks access patterns in such a way for an attacker to figure the sequences and frequency of accessed data.

In Asymmetric Searchable Encryption (ASE) [16] approach users use asymmetric public key encryption schemes to encrypt the data before outsourcing the to the Service Provider. This approach is preferable when the entity that outsources the data is different that the one that wants to access them. Multiple entities can use a public key to encrypt and upload the data and only the entity that owns them can use the corresponding private key can perform a search over the encrypted data. This approach has a great advantage in functionality because it can be used in a variety of applications where data owner and user can be different. On the other hand, it also has a major drawback in inefficiency because all known schemes that can be used depend on the evaluation f pairings on elliptic curves which can be very slow and costly compared to ordinary hash functions and block ciphers.

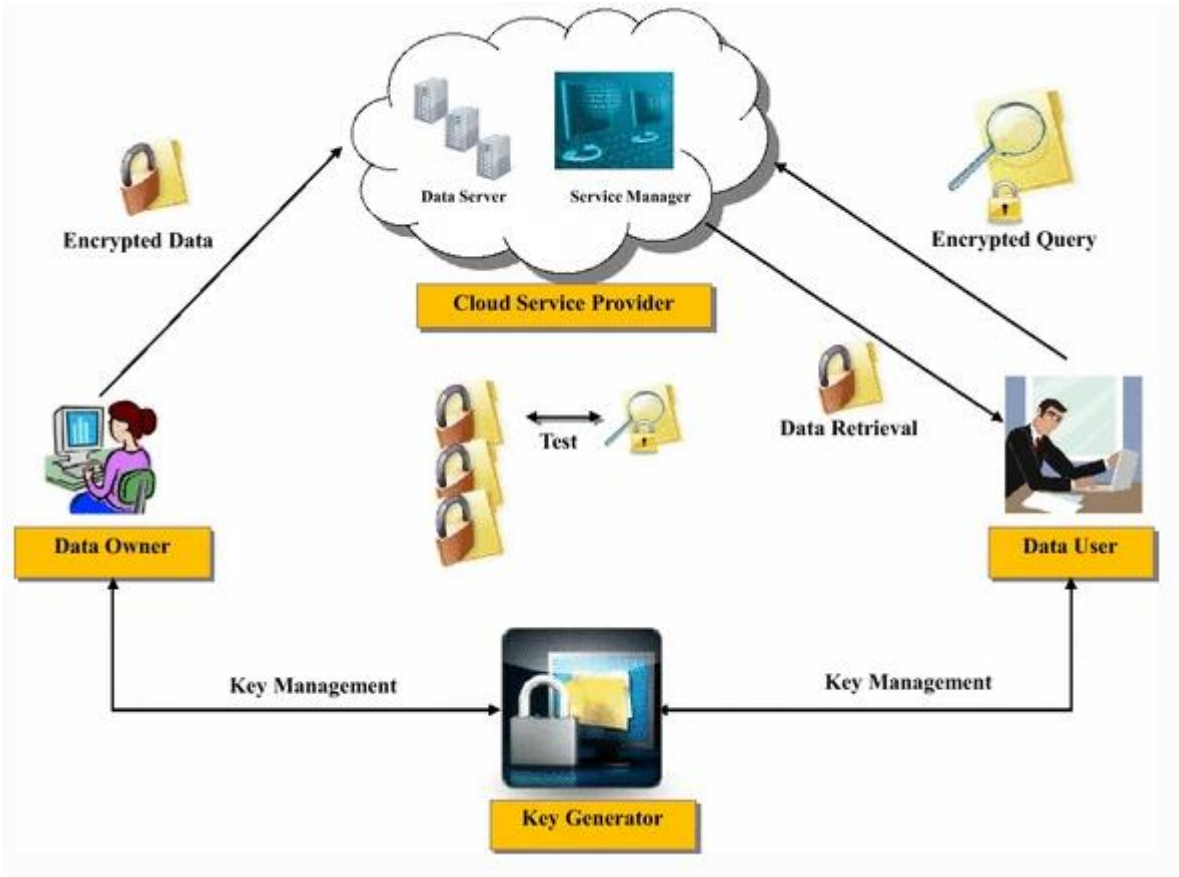


Figure 4 Searchable Key Encryption Model [16]

2.3 Selective Encryption

Selective encryption as stated in the related work of J. Oh, D. Yang and K. Chon. [10] is used when the sender wants to encrypt noncompliant substrings of a larger piece of data. This method can be used to encrypt sensitive data to ensure regulatory compliance while leaving other data unencrypted to preserve as much functionality as possible, thus maintaining both good speed and performance over constant transactions. The problem this encryption scheme tries to solve is this of access control enforcement policies. The data owner in most database encryption solutions needs to be involved in the process in order for queries to be applied or data to be encrypted by a key provided by the data owner. Different users with different keys have different access rights. This solution is presented as very expensive and not easily applicable in real world scenarios. The proposed solutions for this methodology offer the advantage of not requiring the constant online presence of the data owner as well as reducing the number of private keys that each client needs. In essence this encryption scheme exploits hierarchical key derivation methods to achieve a more robust and dynamic access control policy.

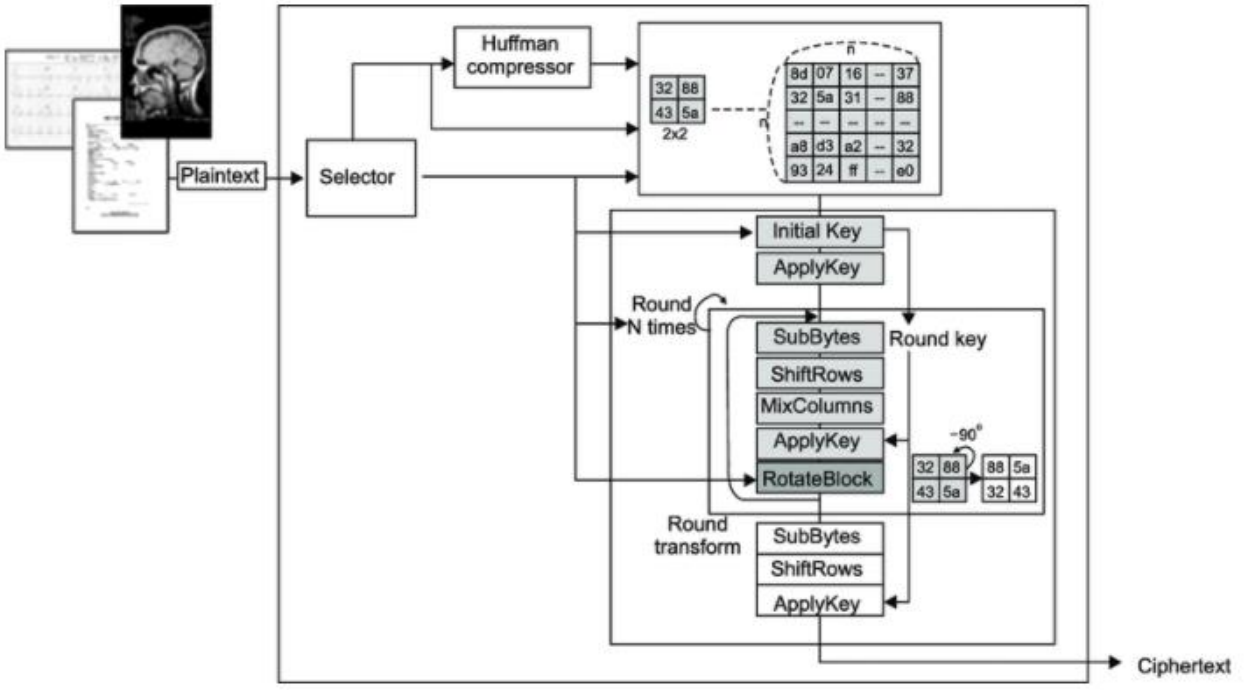


Figure 5 Selective Encryption Algorithm [10]

2.4 Format-Preserving Encryption

Format-preserving encryption (FPE) is used when we want our ciphertext to preserve the format of the original plaintext. This method's existence arises from the problem that encrypting data might be challenging if data models are prone to changes, as it usually involves changing field length limits or data types where the cipher from a typical block would turn them into a hexadecimal or Base64 value. For example, using AES-128-CBC encryption algorithm a simple card number the cypher text will consist of both numbers and letters as well as the produced cyphertext will always be different.

FPE is commonly used for protection of credit card numbers, Social Security numbers and emails. This method has the advantage of allowing an application to apply field validation rules on data while they remain encrypted. Unfortunately, FPE leaks equality, thus failing to provide data integrity and sender authenticity. Equality leakage allows statistical attacks, in which attackers, take advantage of frequency information observed in large sets of ciphertexts to make guesses about plaintexts. A common block cipher, such as AES, is used as a base to take the place of an ideal random function. This has the advantage of easy and fast incorporation of a secret key into an algorithm.

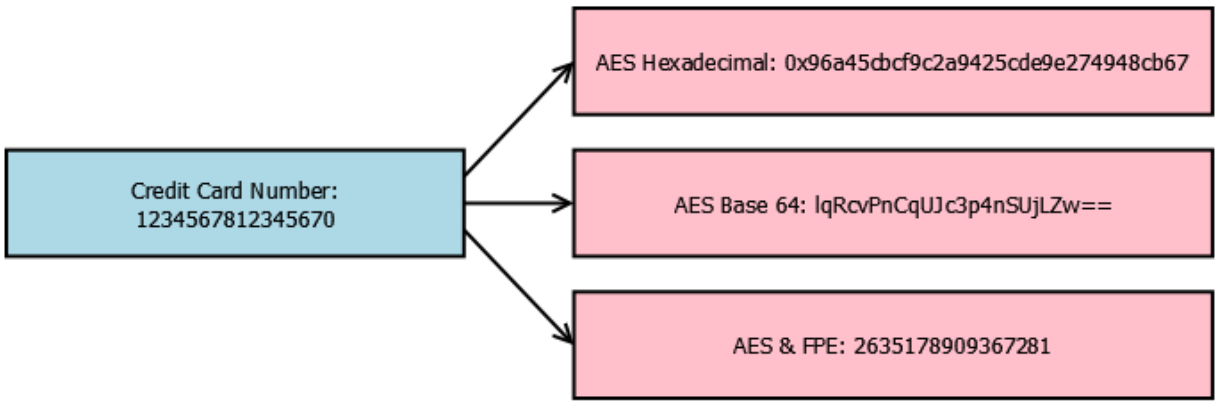


Figure 6 Format Preserving Encryption Example

2.5 Deterministic Encryption

In Deterministic Encryption scheme (DTE) [12] the same ciphertext is always produced for a given plaintext and key even over consecutive operations of the selected encryption algorithm. Examples of this scheme include the RSA cryptosystem (in its original form) and the ECB mode of operation of symmetric block ciphers. In this encryption scheme an adversary in order to get information about various ciphertexts may perform statistical analysis over transmitted encrypted data or attempt to correlate ciphertexts with specific actions. An adversary may also build a large dictionary and then observe the communication channel for matching plaintext-ciphertext pairs. In order to counter this problem cryptographers proposed notion of probabilistic encryption method. In this method a given plaintext can encrypt to one of a very large set of possible ciphertexts who are chosen randomly through encryption the process. Using this notion an attacker will not be able to match any two encryptions of the same message. One primary advantage of using DTE is the efficient searching over encrypted data. If each entry in a database is encrypted using a public key anyone can insert records in the database while only the receiver who has a secret key will be able to decrypt the database entries. However, it is very difficult for the receiver to search for specific records. While such schemes, that allow keyword search exist, they require search time linear in the database size. If database entries were encrypted with deterministic encryption scheme and then sorted, then a specific row of the database could be retrieved in logarithmic time.

2.6 Order-Preserving Encryption

In searchable symmetric encryption (SSE) as stated in the related work of A. Boldyreva, N. Chenette and Y. Lee [15], a user stores his data on the server provider encrypted and when he wants to access them, he uses specific keywords. This encryption method leaks equality of keywords, meaning that it is prone to statistical attacks. One such searchable encryption method that is proposed to tackle this problem is Order-Preserving Encryption by which ciphertexts maintain the order of plaintexts. Its provides the ability to index, search and sort encrypted data in such way that a company can protect numeric or alphanumeric fields while preserving functionality such as sorting and range queries. Because this method leaks order it becomes easy for an attacker to order the ciphertexts and then know that the first ciphertext depends on the first plaintext, the second cipher text on the second plaintext and so on. Even with small

amounts of data encrypted some OPE algorithms have been shown to leak up to half the plaintext so it should carefully be used to protect high value data.

2.7 Fully Homomorphic Encryption

The Homomorphic Encryption Scheme is analyzed by C. Gentry [22], S. Kamara, M. Raykova [23] and F. Armknecht, C. Boyd and C. Carr [24] in their related works. An encryption scheme is called Homomorphic if it supports computation an encrypted data. In addition to the standard encryption and decryption algorithms it has also an evaluation algorithm that can take an encryption of a message x and a function f and returns an encryption of $f(x)$. Homomorphic Encryption schemes can of two types. The first is the 'arithmetic' Homomorphic Encryption schemes which have an extra add or multiply operation feature that takes as inputs of messages x_1 and x_2 and returns encryptions of x_1+x_2 and $x_1 \bullet x_2$, respectively. If an 'arithmetic' Homomorphic Encryption supports both addition and multiplication. By supporting both addition and multiplication it can evaluate any arithmetic circuit over encrypted data, and we say that it is a Fully Homomorphic Encryption (FHE) scheme. The second type of Homomorphic Encryption Schemes is referred as 'non-arithmetic' since it does not provide addition or multiplication operations. Fully Homomorphic Encryption Scheme has multiple applications in the real world. It allows private queries to a data source. Also, it enables searching on encrypted data and lets the client send queries to the server for any function while the server is not learning anything in the process. Even when using the scheme, the later process becomes feasible to use, linear search times are not achieved for large databases. At a high level fully homomorphic encryption given ciphertexts that encrypt a set of data enables both data users and data owners to output a ciphertext that encrypts the aforementioned set of ciphertexts for a desired function as long as that function can be efficiently computed. This encryption scheme can be viewed as an extension of symmetric key encryption. Another derivative of Fully Homomorphic Encryption proposed for computation to a cluster of machines is the Parallel Homomorphic Encryption Scheme [23] which supports computations over encrypted data using an evaluation algorithm that can efficiently be executed in parallel. Using a Parallel Homomorphic Encryption scheme, it is stated that a client can outsource the evaluation of a function on some private input to a cluster of machines as follows. The client encrypts the plaintext and sends the ciphertext and the function to the controller. Using the ciphertext, the controller generates several jobs that it distributes to the workers and then the workers execute their jobs in parallel. When the

entire computation is finished, the client receives a ciphertext which it decrypts to recover the function relative to the plaintext. Moreover, to handle cases where even the function must be hidden, a second variant of Parallel Homomorphic Encryption is introduced which is referred as delegated Parallel Homomorphic Encryption. This variant includes an additional token generation algorithm that takes as input a function and outputs a token that reveals no information about the function but that can be used by the evaluation algorithm to return an encryption of the function of the plaintext.

2.8 Structured Encryption

The Structured Encryption scheme as shown in the related work of M. Chase and S. Kamara [21] is used to solve the problem of hiding and changing the full properties of the data in order to achieve maximum confidentiality and security. It is proposed as a generalization of Searchable Encryption. This encryption scheme encrypts the data of the data owner in such a way that it can be queried with the use of query-specific token, which can only be generated by a Private Key [21]. Moreover, the no useful information is revealed about either the query or the data. When using this encryption scheme the efficiency of the query on the server side should be taken into consideration. Even in cloud storage where actions are happening over massive datasets, even linear time operations can be infeasible [21]. The most common application of Structured Encryption is of course the usage of private queries over encrypted data. In this application the data owner encrypts its structured data resulting in a encrypted data structure and a sequence of ciphertexts. Whenever a data user needs to query the data, he sends a token to the server and then the server uses the token to recover pointers to the appropriate ciphertext. Another application of Structured Encryption is the Controlled Disclosure for local algorithms. In this setting the client wants the server to also perform some computation over the encrypted data. This is achieved with a mechanism that allows the client to encrypt the data and later hide the parts of it that are necessary for the server to perform its tasks.

2.9 Data Tokenization

Data Tokenization [11] is a non-mathematical approach that it alters the plain data with equivalent in length cipher data much like in Property Preserving Encryption. In this methodology a token is created for each plaintext which is then stored in a token vault. Then the tokens are passed to the corresponding application for usage. This approach helps the perseverance of a lot of the application’s functionality such as searching for

keywords. In addition, this methodology satisfies compliance rules for data residency. Tokenization can render it more difficult for attackers to gain access to sensitive data outside of the tokenization system or service. Nonetheless data stored in token vaults need to be protected because in case of leakage all the data are exposed. For this reason, certain security rules are needed to apply. For instance, only the tokenization system can tokenize data to create tokens or detokenize back to redeem sensitive data. Also, the token generation method must be proven to not pose the risk of a direct attack, cryptanalysis, side channel analysis, token mapping table exposure or brute force techniques to reverse tokens back to plaintext. Tokenization can be used to protect data like bank accounts, financial statements, medical records, criminal records, driver's licenses, loan applications, stock trades, voter registrations, and other types of personally identifiable information. It is also used in transactions between bank cards to secure the physical card PAN. Tokenization and classic encryption methods can effectively protect data and may be both used in a security system. At first glance they might both appear similar to each other, but they differ in a few points. While both methods have the same function, they use different processes on the data they are securing. The generated tokens require less computational resources to process than the ciphertexts produced by encryption methods. This can be a great advantage for systems that rely on high performance for database operations.

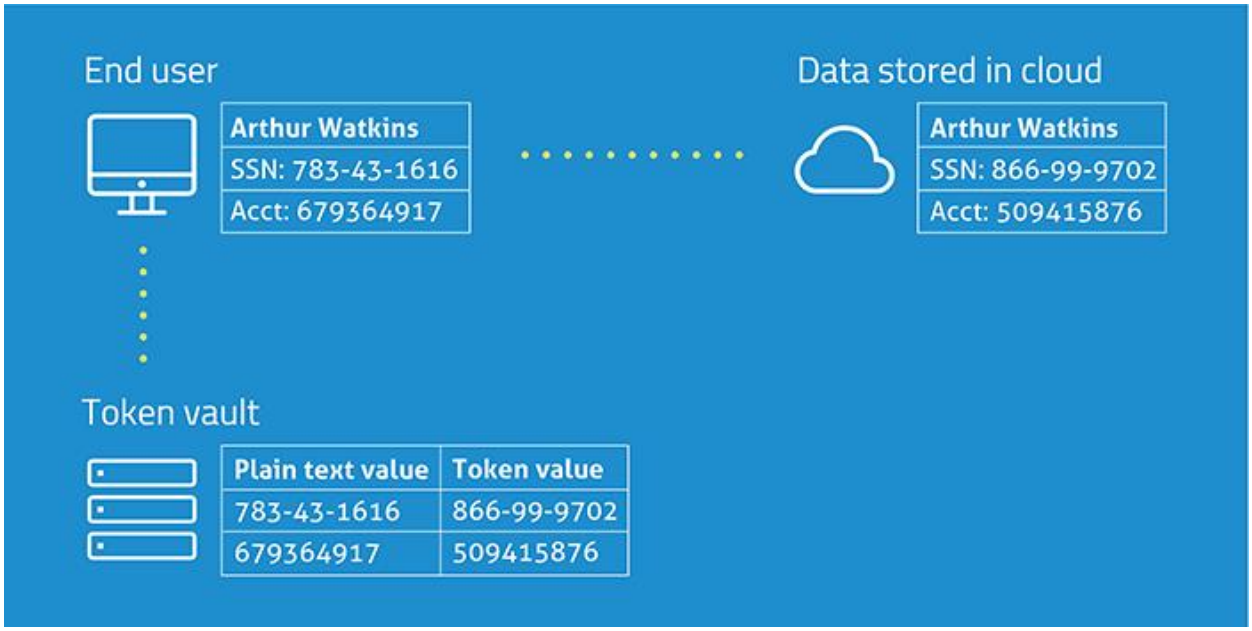


Figure 7 Data Tokenization Example [11]

3. CUSTOM DATABASE DRIVEN ENCRYPTION SCHEMES

There are many encrypted database solutions designed that are based on primary encryption schemes. All these solutions were created to provide security and confidentiality for systems that require the use of relational databases. Moreover, these systems are designed with the support for data searching over encrypted databases while allowing for good performance as well as query and storage complexities. They also study the leakage profiles of transferred data between end-to-end communication and possible attacking techniques and the data an attacker can retrieve by using them. Finally, some of these custom encryption constructions are dynamic, meaning that possible future enhancements are proposed.

3.1 Property Preserving Encryption

In the process of encrypted searching in relational databases, solutions are focused on schemes like Property Preserving Encryption (PPE) like Deterministic and Order Preserving encryption. A PPE scheme is an encryption scheme that leaks a certain property of the plaintext. It was the CryptDB system by R. Popa, C. Redfield and N. Zeldovich [14] which first demonstrated how to use PPE to construct an encrypted database system that supports a subject of SQL. In this system all operations are outsourced to the database server and not at the client. CryptDB will be discussed later on as a practical implementation of the privacy and security problems that are already mentioned. Another system is the Cipherbase system, which uses both DTE and OPE and supports all of SQL. At a high level in this system database operations can be either done in a secure co-processor or over encrypted data in a similar manner as in CryptDB. Other encryption systems that are based in PPE are BigQuery Demo [32] and Always Encrypted [33], which both use DTE and not OPE. PPE-base encrypted database schemes are competitive with real world RDBMs as they require a small number of changes to the standard database infrastructure. They main reason why they are efficient and is the actual use of PPE which allows for the same fast operations on encrypted data with the same standard algorithms and optimizations as of plaintext data. For the implementation of Property Preserving encrypted databases a high-level architecture of the CryptDB system is recalled. The system is composed of an application, a proxy and a server. The application and proxy are considered as trusted

while the server is considered not trusted. The way it works is that the proxy generates a master secret key and uses it to encrypt each tables as follows. First an anonymized scheme is created where the attributes of each column are replaced with random labels. The mapping between the attributes and their labels are stored at the proxy. Then each cell is encrypted using four different onions (Equality, Order, Search and Add). To support queries on encrypted data, the encrypted cells in the encrypted database are decrypted down to a certain layer [13] and every cell in a given column is peeled to the same level. The proxy keeps track of the layer at which each column is peeled. In order to query an encrypted database, the application issues a SQL query that is then rewritten by the proxy before being sent to the server. In the changed query every column name is replaced with its random label and each constant is replaced with a ciphertext determined as a function of the semantics of the query. Then the proxy checks the onion level of the relevant columns to determine if they need to be peeled further. If so, it the appropriate decryption keys are sent to the server so that it peels the columns down to the appropriate level. In the Cipherebase implementation, columns are encrypted directly with the Property Preserving Encryption scheme needed to support the query and onions are not involved in the process.

An encrypted database system should be able to protect against a variety of security threats. It is stated that there are at least two kinds of attacks against that. Firstly, there are the individual attacks. In this kind of attack, the attacker wants to recover information about a row in the database. Secondly there are the aggregate attacks. In an aggregate attack the attacker wants to recover statistical information about the database. It noted that depending on the context, aggregate attacks can be very harmful. In the related work by M. Naveed, S. Kamara and C. V. Wright [2] the focus is on weak attacker who has access to the encrypted database but not to the encrypted queries. Also, it is assumed that the attacker has access to the encrypted database in a state after the application has been running for a while, so the onions of each cell are peeled down to the lowest layer that is needed to support the queries that are generated by the application. Lastly it is assumed that the attacker has access to auxiliary information about the system and the data. It is standard in an adversarial model since the adversary can always consult public information sources to carry out the attack. This includes application details, public statistics and prior versions of the database.

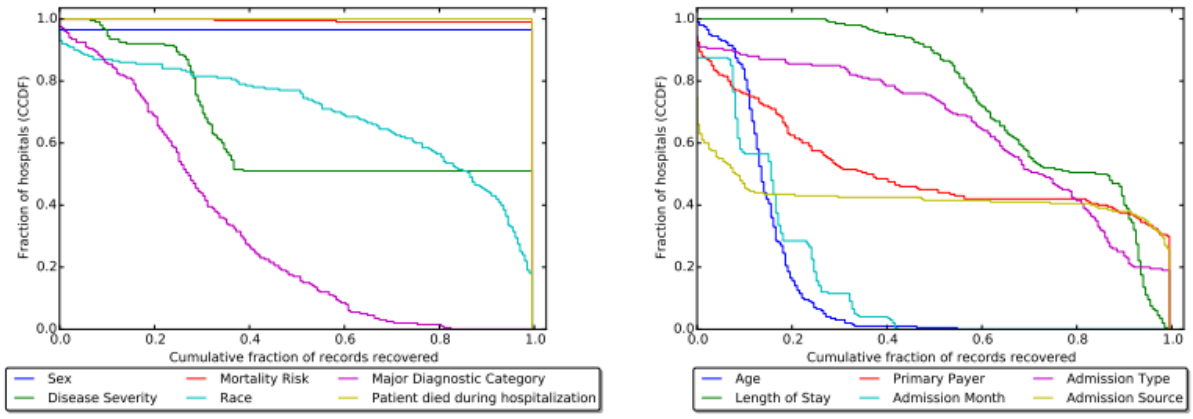
For an attack to be successful against an encrypted database it is only required for the attacker to recover partial information about a single cell of the database. Four different

attacks are tested on this encryption methodology and then their empirical results are discussed. For the empirical results electronic medical records were used because they provide large amount of private and sensitive information about both patients and hospitals that treat them.

Firstly, the most basic and well-known Frequency Analysis attack was studied. It was developed in the 9th century and is used to break classical cyphers. Frequency Analysis is well known to break Deterministic Encryption and in particular deterministic encryption columns. In this attack the encrypted columns, which were encrypted with the Deterministic Encryption, are decrypted using an auxiliary dataset that is correlated with the plaintext column. Because the extent of the correlation needed is not significant, many publicly available datasets can be used on encrypted columns by the attackers. This attack managed to recover the mortality risk and patient death attributes for 100% of the patients for at least 99% of the 200 large hospitals. It also recovered the disease severity for 100% of the patients for at least 51% of the same hospitals.

Secondly comes the 'ℓ_p-optimization' family of attacks. It is a newly introduced family of attacks which also works by decrypting the Deterministic Encryption columns. This family is parameterized by the ℓ_p-norms [25] and it is based on combinatorial optimization techniques. The basic idea behind this family of attacks is an assignment from ciphertexts to plaintexts that minimizes a given cost function. This has the effect of minimizing the total mismatch in frequencies across all plaintext/ciphertext pairs. This attack had the same results as the 'frequency analysis' attack. In fact, both attacks for a fixed encrypted column and auxiliary dataset, they decrypted same exact ciphertexts. However, frequency analysis did consistently better than ℓ_1-optimization and this a statement that raises interesting theoretical and practical questions. It is noted that from a theoretical perspective it would be interesting to understand the exact relationship between frequency analysis and ℓ_p-optimization. The experiments that were performed tell us that ℓ_1-optimization is different from frequency analysis since they generated different results, but they did not distinguish between frequency analysis and ℓ_2 and ℓ_3 optimization. From a practical perspective, the question raised is what the motivation for is using ℓ_p-optimization over frequency analysis and the main reason behind this is that ℓ_p-optimization not only decrypts an encrypted column but, while doing so, also produces cost information about the different solutions it finds. This is due to its use of combinatorial optimization. It turns out that this extra information can be leveraged to

attack columns which we do not know their attributes something we cannot always do with frequency analysis.



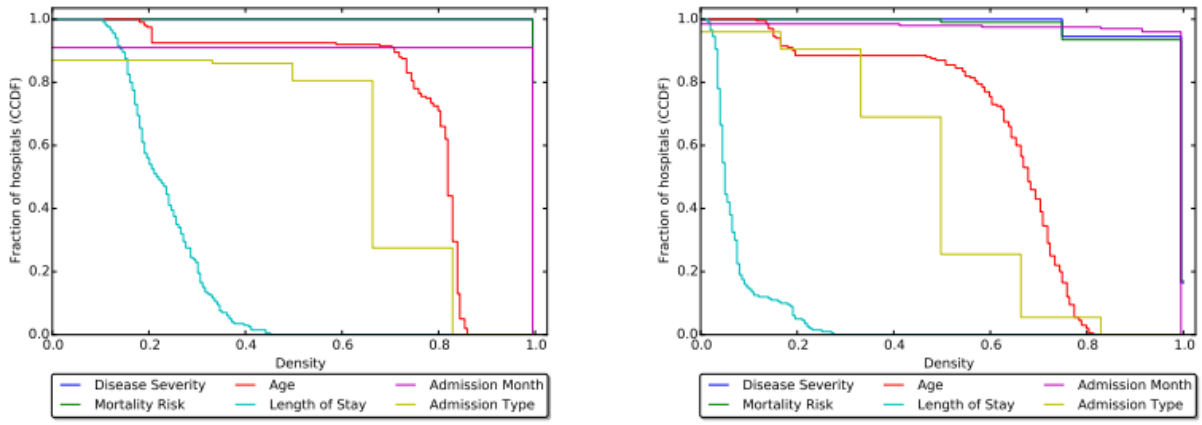
(a) First set of attributes (b) Second set of attributes

Figure 8 Results of ℓ_2-optimization on DTE-encrypted columns on 200 largest hospitals with 2009 HCUP/NIS as target data and 2004. HCUP/NIS as auxiliary data [2]

Thirdly is the Sorting Attack. This attack decrypts Order Preserving encrypted columns. This attack is very simple yet very powerful in practice. It is applicable to all columns that satisfy a condition that Is called density. An OPE-encrypted column is called δ-dense if it contains the encryptions of at least a δ fraction of its message space. If δ=1 it is said that that column is dense. Moreover, this attack requires access to auxiliary information and can recover a large fraction of column cells. Also, it holds for many real-world scenarios. This attack managed to recover the admission month and mortality risk of 100% of patients for at least 90% of the 200 largest hospitals.

Lastly is the Cumulative Attack. It also a newly introduced attack which also decrypts Order Preserving encrypted columns. It is introduced as a more efficient attack than Sorting Attack because Sorting Attack is only applicable to dense columns. This attack is applicable even to low-density columns and makes use of combinatorial optimization techniques as in 'ℓ_p optimization' attack. Furthermore, this attack requires access to auxiliary information and can recover a large fraction of column cells. In this attack an attacker learns the sample frequency of each ciphertext in the column. These samples compose a histogram for the encrypted column which the attacker can use to match the Deterministic Encrypted ciphertexts to their plaintexts by finding (c, m) pairs where c and m have similar frequencies. After testing this attack managed to recover disease severity, mortality risk, age, length of stay, admission month, and admission type of at least 80% of the patients for at least 95% of the largest 200 hospitals. Also, for small

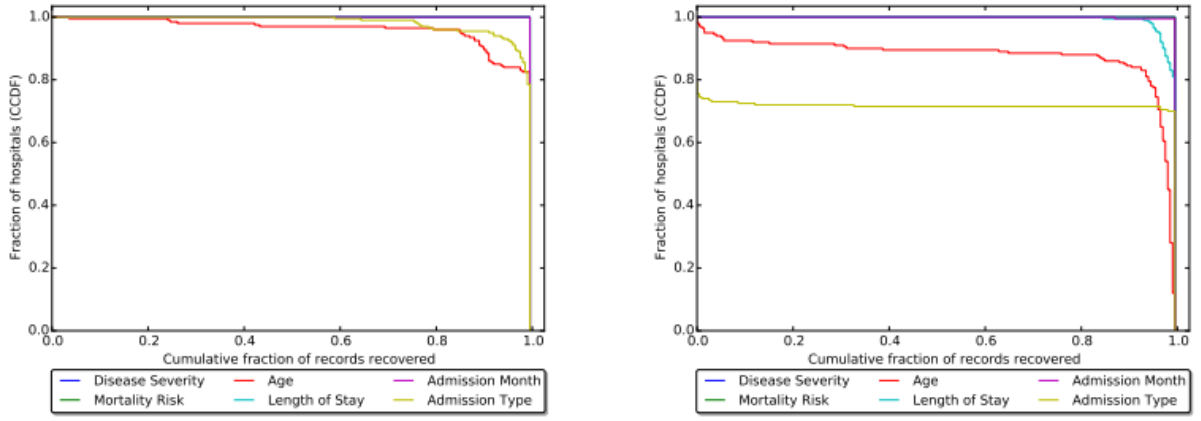
200 hospitals this attack managed to recover admission month, disease severity, and mortality risk for 100% of the patients for at least 99:5% of the hospitals.



(a) Density for large hospitals

(b) Density for small hospitals

Figure 9 Density - Ratio of the number of values of an attribute present in a column to the total number of values of the attribute. [2]



(a) Results for the 200 largest hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data.

(b) Results for the 200 small hospitals with 2009 HCUP/NIS as target data and 2004 HCUP/NIS as auxiliary data.

Figure 10 Results of Cumulative attack on OPE-encrypted columns. [2]

Most of the aforementioned inference attacks need an auxiliary source of information and their success highly depends on how well correlated the auxiliary data are with the plaintext column. The choice of auxiliary data is therefore a critical factor when evaluating an inference attack. While a strongly correlated auxiliary dataset may yield better results it may not be available to the attacker. On the other hand misjudgment of which datasets are available to the attacker can lead to overestimating the security of the system. An additional difficulty is that the 'quality' of the auxiliary dataset is application dependent. The results of the study of the attacks showed that they can recover a large number of Property Preserving based medical encrypted databases.

The performance of these attacks in other datasets like human resource or accounting is leaved as important future work but is specified that it will be as successful as in datasets from medical databases as they use many similar correlated data. Moreover the presented results from the studies on the attacks should be considered as a lower bound on what can be extracted from Property Preserving encryption databases for two main reasons. Firstly, these attacks only make use of leakage from the encrypted database and do not exploit the considerable amount of the queries to the encrypted database. Secondly the attacks do not target the weakest encryption schemes that can be used by in these systems such as schemes that support ‘equi-joins’ and ‘range-joins’.

3.2 SPX Encryption Scheme

The SPX Encryption scheme is proposed by S. Kamara and T. Moataz [4] as an improvement over Property Preserving Encryption scheme and the CryptDB structure. This encryption scheme is based solely on Structured Encryption and does not make use of schemes such as Deterministic Encryption an Order Preserving Encryption. As a result, this approach leaks considerably less information than Property Preserving Encryption based solutions. As stated, this construction is efficient and under certain conditions on the database and the queries, can have optimal query complexity. This approach also is shown to be dynamic and can be extended to support traditional operations such as row addition and row deletion while maintaining the scheme’s optimal query complexity. It mainly focuses on conjunctive queries with ‘Where’ predicates that their attributes are not the same across terms. In addition, this approach can handle a sub-class of SQL queries and an even larger class if a small amount of post-processing is allowed at the client.

This approach to encrypted databases replaces the plaintext execution of a SQL query with an encrypted execution of the query by executing the server’s low-level operations directly on the encrypted cells. This is possible due to the properties of Property Preserving Encryption which guarantee that operations on plaintexts can also be done on ciphertexts. This easy integration approach makes the design of encrypted databases relative straightforward since the only requirement is to replace plaintext cells with property preserving encrypted cells Because relational databases and SQL queries can be rather complex it is stated that it is not clear how to solve this problem

without resorting to schemes like Property Preserving Encryption, Fully Homomorphic Encryption or ORAM. There are several approaches that are descripted towards solving this problem.

The first approach proposed is the isolation of the conceptual difficulties of the problem. Since relational databases consist of a set of two-dimensional arrays are considered relatively simple from a data structure perspective. The difficult challenge is posed from the complexity of SQL and also the fact that it is declarative. It is stated that to overcome this a simpler but widely applicable and well-studied subset of SQL queries is used and a more procedural view is taken. Specifically, the work is done with the 'relational algebra' formulation of SQL which is more amenable to cryptographic techniques. The first who introduced relational algebra was Codd [34] as w way to formalize queries in relational databases. Relational algebra consists of all the queries that can be expressed form a set of basic operations. As it was later shown by Chandra and Merlin [35] three of these operations, specifically selection, projection and cross product capture a large class of useful queries called 'conjunctive queries' that have particularly nice theoretical properties. The subset of the relational algebra expressed by the aforementioned 'conjunctive queries' is also called the 'SPC algebra'. In this way only a procedural representation of SQL queries is supported but the problem is also reduced to handling just three basic operations. Also, another important advantage of working in the SPC algebra is that every SPC query can be written in a standard form. In a result a single construction can be designed which can handle all SPC queries. Lastly it is noted that SPC normal form is not always guaranteed to be the most efficient.

One main difficulty in the case of relational databases and in handling SPC queries, is that queries are 'constructive', meaning, that they produce new data structures from the original base structure. In this case, the structures that must be generated by the server to answer the corresponding query depend on the query itself and as a result they cannot be constructed by the client in an initial pre-processing phase. However, while SPC queries are constructive, they are not arbitrarily so. That means that the tables needed to answer an SPC query are not completely arbitrary but are structured in a way that can be predicted at setup. As it has been stated, the challenge that remains on the matter is to provide the server with the means to construct the appropriate intermediate and final tables and to design encrypted structures that will allow it to efficiently find the encrypted content it needs to create those tables. SPC normal form queries can be

rewritten in a different and optimized form that is introduced as heuristic normal form (HNF). It ordered for queries in HNF form to be handled a set of encrypted structures that store different representations of the database need to be created. Then these representations when used and combined in a appropriate way, tokens can be generated for the server to recover the encrypted database rows needed for it to process the query in its HNF form.

In the case of dynamism, it is stated that it poses a challenge to maintain the schemes query complexity while not introducing additional leakage. The related work focused only on row additions and deletions and leave as an open problem the handling of more complex update operations. A two-party protocol is introduced to solve this challenge without the need for the client to get access to the entire structure an without leaking too much information on the server.

In the case of leakage, the scheme is analyzed by using algorithms that make black-box use of several lower-level STE schemes such as multimap and dictionary encryption schemes. As mentioned, this approach has several advantages. Firstly, it results in modular constructions that are easier to describe and analyze. Secondly the schemes can benefit from any improvement in the underlying building blocks. Also, this approach holds with respect to efficiency but also with respect to leakage because it is proved with respect to a black-box leakage profile.

The first step of the generation of SPX construction was to build different representations of the database, each designed to handle a particular operation of the SPC algebra. Four representations were used. Firstly, was a row-wise representation of the database instantiated as a map that maps the coordinate of every row in the database to the contents of the row. Secondly was a column-wise representation in which also an instantiated map, maps the coordinate of every column to the contents of that. Thirdly contrary to the previous representations no content of the table is stored, but the equality relation among values in the database instead. Again, a multi-map is created that maps each value in every column to all the rows that contain the same value. Lastly the fourth representation was a set of multi-maps one for every column in the database. Each of them mapped a pair of column coordinates to all the rows that have the same value in both those columns. Then by using multimap and dictionary encryption schemes, encrypted multi-maps and dictionary was created. All the different representations are designed so that, given an SPC query, the server can generate the intermediate encrypted tables needed to produce the final encrypted result. The server

will need to make further intermediate queries on encrypted tables in order to do that. It is stated that this type of query is ‘constructive’, meaning that the intermediate and final encrypted tables are not the result of pre-processing at setup time but are constructed at query time by the server as a function of the query and the underlying database. For this operation, a chaining technique is used. The idea is to store query tokens for one encrypted structure as the responses of another encrypted structure and by chaining the various encrypted multi-maps constructive queries can be handled by first querying some subset of the encrypted multi-maps to recover either tokens for encrypted multi-maps further down the chain or encrypted content which we will be used to populate intermediate tables later. This process continues until the final result is constructed.

The chose database representation for this scheme provides the users with a way to control both efficiency and security. While intermediate results will vary depending on the query, the chaining sequence remains the same for any SPC query written in our heuristic normal form. The chaining sequence is important because it determines the leakage profile of the construction. The security of the scheme is analyzed by providing a black-box leakage profile that is a function of the leakage profile of the underlying encrypted multi-map and encrypted dictionaries used. This allows for isolating the leakage that is coming from the underlying building blocks and the leakage that is coming directly from the SPX construction. Moreover, this helps to reason about and decide which concrete instantiations to use as building blocks so that the appropriate kind of leakage/performance tradeoff can be chosen.

At a high level, the SPX construction makes use of a response-revealing multi-map encryption scheme Σ_{MM} = (Setup, Token, Get), of a response-revealing dictionary encryption scheme Σ_{DX} = (Setup, Token, Get) and of a symmetric-key encryption scheme SKE = (Gen, Enc, Dec). The Setup algorithm takes as input a database DB = (T1, . . ., Tn), creates the multi-maps MM_R, MM_C, MM_V, $\{MMc\}_{c \in DB^T}$ and the dictionary DX and then proceeds by encrypting each structure with the appropriate structured encryption scheme. The Token algorithm parses the heuristic normal form query and generates appropriate tokens for each structure in order to enable the server to perform an indexed execution of the query. More specifically takes as input a secret key and a query in SPC normal form. The query algorithm makes use of the plaintext indexed HNF query evaluation algorithm. Lastly the decryption algorithm takes as input a secret key and the response table returned by the server and then decrypts each cell of the response table. It is proven that the Search complexity of the construction is optimal. As

for the storage complexity in the DB is it shown to be more complex and highly dependable on the number of columns and column domains of each table.

Let $\Sigma_{\mathsf{DX}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a response-revealing dictionary encryption scheme, $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a response-revealing multi-map encryption scheme and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric-key encryption scheme. Consider the DB encryption scheme $\mathsf{SPX} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Dec})$ defined as follows [a]:

- $\mathsf{Setup}(1^k, \mathsf{DB})$:
  1. initialize a dictionary DX;
  2. initialize multi-maps MM_R, MM_C and MM_V;
  3. initialize multi-maps $(\mathsf{MM}_{\mathsf{att}})_{\mathsf{att} \in \mathbb{S}(\mathsf{DB})}$;
  4. for all $\mathbf{r} \in \mathsf{DB}$ set
     $$\mathsf{MM}_R[\chi(\mathbf{r})] := \Big(\mathsf{Enc}_{K_1}(r_1), \ldots \mathsf{Enc}_{K_1}(r_{\#\mathbf{r}}), \chi(\mathbf{r})\Big);$$
  5. compute $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_R)$;
  6. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$, set
     $$\mathsf{MM}_C[\chi(\mathbf{c})] := \Big(\mathsf{Enc}_{K_1}(c_1), \ldots \mathsf{Enc}_{K_1}(c_{\#\mathbf{c}}), \chi(\mathbf{c})\Big);$$
  7. compute $(K_C, \mathsf{EMM}_C) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_C)$;
  8. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
     (a) for all $v \in \mathbf{c}$ and $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$,
        i. compute $\mathsf{rtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \chi(\mathbf{r})\Big)$,
     (b) set
     $$\mathsf{MM}_V\Big[\big\langle v, \chi(\mathbf{c})\big\rangle\Big] := \Big(\mathsf{rtk}_{\mathbf{r}}\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}};$$
  9. compute $(K_V, \mathsf{EMM}_V) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_V)$;
  10. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
      (a) for all $\mathbf{c}' \in \mathsf{DB}^{\intercal}$ such that $\mathsf{dom}(\mathsf{att}(\mathbf{c}')) = \mathsf{dom}(\mathsf{att}(\mathbf{c}))$,
         i. initialize an empty tuple $\mathbf{t}$;
         ii. for all $i, j \in [m]$ such that $\mathbf{c}[i] = \mathbf{c}'[j]$,
            A. compute $\mathsf{rtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \Big\langle \mathsf{tbl}(\mathbf{c}), i \Big\rangle\Big)$;
            B. compute $\mathsf{rtk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \Big\langle \mathsf{tbl}(\mathbf{c}'), j \Big\rangle\Big)$;
            C. add $(\mathsf{rtk}_i, \mathsf{rtk}_j)$ to $\mathbf{t}$;
         iii. set
         $$\mathsf{MM}_{\mathbf{c}}\Big[\Big\langle \chi(\mathbf{c}), \chi(\mathbf{c}') \Big\rangle\Big] := \mathbf{t};$$
      (b) compute $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_{\mathbf{c}})$;
      (c) set $\mathsf{DX}[\chi(\mathbf{c})] = \mathsf{EMM}_{\mathbf{c}}$;
  11. compute $(K_D, \mathsf{EDX}) \leftarrow \Sigma_{\mathsf{DX}}.\mathsf{Setup}(1^k, \mathsf{DX})$;
  12. output $K = \big(K_R, K_C, K_V, K_D, \{K_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^{\intercal}}\big)$ and $\mathsf{EDB} = \big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$;

[a] Note that we omit the description of Dec since it simply decrypts every cell of R.

Figure 11 SPX: a relational DB encryption scheme (Part 1). [4]

- $\mathsf{Token}(K, q)$:
  1. rewrite q as $\pi_{\mathsf{att}_1,\cdots,\mathsf{att}_h}\Big([a_1] \times \cdots \times [a_f] \times \varphi_1 \times \cdots \times \varphi_d\Big)$;
  2. for all $i \in [h]$,
     (a) if att_i is not in $\varphi_1 \times \cdots \times \varphi_d$, compute $\mathsf{ptk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_C, \chi(\mathsf{att}_i)\Big)$, and set $\mathsf{ytk}_i := (\mathsf{ptk}_i, \mathsf{out})$;
     (b) else set $\mathsf{ptk}_i := (\mathsf{ptk}_i, \mathsf{in})$, where $\mathsf{ptk}_i = \mathsf{pos}_i$ and pos_i is att_i's position in the tables in $\varphi_1 \times \cdots \times \varphi_d$;
  3. for all $i \in [d]$,
     (a) if φ_i has form $\sigma_{\mathsf{att}_1 = a_1}(\mathsf{T}) \cap \cdots \cap \sigma_{\mathsf{att}_p = a_p}(\mathsf{T})$, set $\mathsf{stk}_i := (\mathsf{itk}_1, \ldots, \mathsf{itk}_p)$, where, for all $j \in [p]$, $\mathsf{itk}_j := \Sigma_{\mathsf{MM}}.\mathsf{Token}\big(K_V, \langle a_i, \chi(\mathsf{att}_i)\rangle\big)$;
     (b) if φ_i has form $\sigma_{\mathsf{att}_1 = \mathsf{att}_2}(\mathsf{T}_1 \times \mathsf{T}_2)$, set $\mathsf{stk}_i := (\mathsf{dtk}_i, \mathsf{jtk}_i)$ where

$$\mathsf{dtk}_i := \Sigma_{\mathsf{DX}}.\mathsf{Token}\big(K_D, \chi(\mathsf{att}_1)\big) \quad \text{and} \quad \mathsf{jtk}_i := \Sigma_{\mathsf{MM}}.\mathsf{Token}\big(K_e, \langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\rangle\big);$$

  4. for all $i \in [f]$, compute $\mathrm{e}_i \leftarrow \mathsf{Enc}_{K_1}(a_i)$;
  5. output $\mathsf{tk} = \big((\mathsf{ytk})_{i\in[h]}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\big)$;
- $\mathsf{Query}(\mathsf{EDB}, \mathsf{tk})$:
  1. parse EDB as $\big(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, \mathsf{EDX}\big)$ and tk as $\big((\mathsf{ytk})_{i\in[h]}, (\mathrm{e}_i)_{i\in[f]}, (\mathsf{stk}_i)_{i\in[d]}\big)$;
  2. for $i \in [d]$,
     (a) if stk_i has form $(\mathsf{itk}_1, \ldots, \mathsf{itk}_p)$,
        i. for $j \in [p]$,
           A. instantiate an empty set R'_j,
           B. compute $(\mathsf{rtk}_1, \ldots, \mathsf{rtk}_s) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_V, \mathsf{itk}_j)$,
           C. for all $m \in [s]$, set $R'_j := R'_j \bigcup \{\Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_m)\}$,
        ii. set $R_i = R'_1 \cap \cdots \cap R'_p$;
     (b) if stk_i has form $(\mathsf{dtk}_i, \mathsf{jtk}_i)$,
        i. compute $\mathsf{EMM} := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EDX}, \mathsf{dtk}_i)$,
        ii. compute $\big((\mathsf{rtk}_1, \mathsf{rtk}'_1), \ldots, (\mathsf{rtk}_s, \mathsf{rtk}'_s)\big) := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}, \mathsf{jtk}_i)$,
        iii. compute

$$\mathbf{ct}_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_1), \ldots, \mathbf{ct}_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}_s),$$

        and

$$\mathbf{ct}'_1 := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_1), \ldots, \mathbf{ct}'_s := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_R, \mathsf{rtk}'_s),$$

        iv. set $R_i = \Big\{\mathbf{ct}_j \times \mathbf{ct}'_j\Big\}_{j\in[s]}$;
  3. set $\mathsf{S} = \mathrm{e}_{a_1} \times \cdots \times \mathrm{e}_{a_f} \times R_1 \times \cdots \times R_d$;
  4. for all the z sub-tokens ytk with form $(\mathsf{ptk}_i, \mathsf{in})$, compute,

$$\mathsf{W} := \pi_{\mathsf{ptk}_1,\cdots,\mathsf{ptk}_s}(\mathsf{S});$$

  5. for all the $(h - z)$ sub-tokens ytk with form $(\mathsf{ptk}_i, \mathsf{out})$, compute,

$$\mathbf{ct}_i := \Sigma_{\mathsf{MM}}.\mathsf{Get}(\mathsf{EMM}_C, \mathsf{ptk}_i);$$

  6. output $\mathsf{R} := \mathbf{ct}_1 \times \cdots \times \mathbf{ct}_{h-z} \times \mathsf{W}$;

Figure 12 SPX: a relational DB encryption scheme (Part 2). [4]

Property Preserving solutions can handle a large amount of SQL queries including conjunctive queries. However, in order to support conjunctive queries, these solutions need to rely on Deterministic Encryption. This will reveal the frequency information on entire columns to the server and depending on the setting, frequency patterns can be particularly dangerous. The SPX solutions leaks considerably less information for several reasons. First it does not leak any information on entire rows and columns. Also, if the underlying multi-map and dictionary schemes are instantiated with standard constructions, the information leaked about the attributes and matching rows is “repetition” type of information.

3.3 OPX Encryption Scheme

The OPX Encryption Scheme which was proposed by S. Kamara, T. Moataz, S. Zdonik and Z. Zheguang [5], comes as an extension to the SPX Encryption Scheme. Though SPX was practical it was not efficient enough to yield a system that was competitive with commercial plaintext database management systems. This was based on several reasons.

First the query processing and optimization take place. SQL queries are processed by Database Systems in a series of steps. A SQL query is converted into a logical query tree which is a tree-based representation of the query where each node is a relational algebra operator. Then Query trees are evaluated bottom up by evaluating the operators at the leaves on the appropriate database tables. The intermediate table that results from an operation is then passed on to its parent node until the result table is the output by the root. After that the initial query tree is converted by a query optimizer to an equivalent but optimized query tree using various optimization techniques. It is stated that query optimizers are one of the most important components of a Database Management System and a large part of why commercial systems are so efficient. For encrypted database systems to be competitive with commercial systems, they must support some form of query optimization. However, the SPX construction does not allow for query optimization because it only handles queries in heuristic normal form which is a very specific form of query tree. Given a query tree, SPX evaluates leaf operations by querying one of its encrypted multi-maps directly and then uses various algorithms to process the internal operations on intermediate results. While the leaf operations are handled optimally thanks to the encrypted multi-maps, internal operations are not necessarily handled in optimal or even sub-linear time.

Second is the Sub-optimality of correlated queries. A conjunctive SQL query is uncorrelated if the terms of its ‘where’ clause include columns that are in different tables. The query trees of uncorrelated queries are relatively simple. They have height 1 with leaves that are either join or filter operations and a root that is a Cartesian product. So, SPX can handle these queries very efficiently since leaf operations are evaluated optimally by directly querying the encrypted multi-maps. Correlated queries, on the other hand, have query trees of height 2 or more which means they have internal operations which are not necessarily handled optimally.

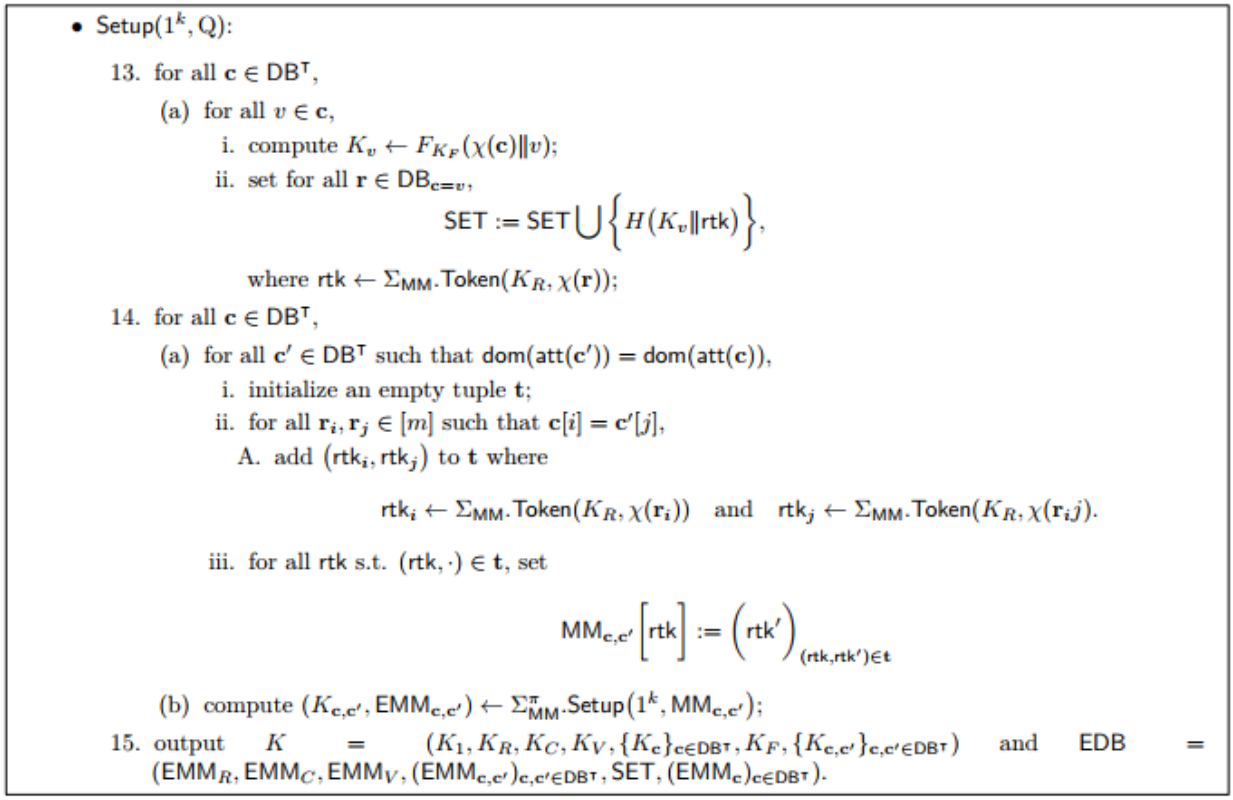
OPX is a response-hiding STE scheme that supports query optimization and can handle all conjunctive SQL queries optimally. This is achieved by using additional encrypted structures that are designed to optimally handle internal operations. These additional structures include an encrypted set structure to handle internal filters as well as an additional set of encrypted multi-maps to handle internal joins. Moreover, these additional structures increase the storage overhead but only concretely. It is stated that OPX has the same storage overhead as SPX. The leakage profile of OPX is also close to that of SPX. In addition to executing internal operations more efficiently, OPX has the advantage that it can handle any query tree, not only heuristic normal form trees. It is mentioned that this is an important feature because it means that OPX can be used to query trees that have been optimized by standard query optimizers. At a high level OPX uses a response-revealing multi-map encryption scheme Σ_{MM}, the adaptively secure encrypted multi-map scheme Σ^{π}_{MM} a symmetric encryption scheme SKE a pseudo-random function F, and a random oracle H. For a given database OPX produces three encrypted multi-maps that are named EMM_R, EMM_C and EMM_V respectively, two collections of encrypted multi-maps and a set structure. Furthermore, the token algorithm in this construction takes as input a key and a query tree and generates a token tree. The token tree is a copy of the query tree and first initialized with empty nodes. It performs a post-order traversal of the query tree and perform several operations for each visited node. The query algorithm takes as input the encrypted database and the token tree and then it performs post-order traversal of the token tree for each visited node. The query complexity is shown to be asymptomatically optimal. The storage complexity of OPX is shown to be like that of SPX asymptotically, but larger concretely. This is because OPX needs two additional encrypted structures: a collection of encrypted multi-maps and an encrypted set.

Let $\Sigma_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be a response-revealing multi-map encryption scheme, $\Sigma^{\pi}_{\mathsf{MM}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Get})$ be the response-revealing multi-map encryption scheme in [6], $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, $F : \{0,1\}^k \times \{0,1\}^\star \to \{0,1\}^m$ be a pseudo-random function, and $H : \{0,1\}^\star \to \{0,1\}^m$ be a random oracle. Consider the DB encryption scheme $\mathrm{OPX} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Dec})$ defined as follows [a]:

- $\mathsf{Setup}(1^k, \mathsf{DB})$:
  1. initialize a set SET;
  2. initialize multi-maps MM_R, MM_C and MM_V;
  3. initialize multi-maps $(\mathsf{MM}_{\mathsf{att}})_{\mathsf{att} \in \mathsf{DB}^\intercal}$;
  4. initialize multi-maps $(\mathsf{MM}_{\mathsf{att},\mathsf{att}'})_{\mathsf{att},\mathsf{att}' \in \mathsf{DB}^\intercal}$ such that $\mathsf{dom}(\mathsf{att}) = \mathsf{dom}(\mathsf{att}')$;
  5. sample two keys $K_1, K_F \xleftarrow{\$} \{0,1\}^k$;
  6. for all $\mathbf{r} \in \mathsf{DB}$ set
     $$\mathsf{MM}_R\big[\chi(\mathbf{r})\big] := \Big(\mathsf{Enc}_{K_1}(r_1), \ldots \mathsf{Enc}_{K_1}(r_{\#\mathbf{r}}), \chi(\mathbf{r})\Big);$$
  7. compute $(K_R, \mathsf{EMM}_R) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_R\big)$;
  8. for all $\mathbf{c} \in \mathsf{DB}^\intercal$, set
     $$\mathsf{MM}_C\big[\chi(\mathbf{c})\big] := \Big(\mathsf{Enc}_{K_1}(c_1), \ldots \mathsf{Enc}_{K_1}(c_{\#\mathbf{c}}), \chi(\mathbf{c})\Big);$$
  9. compute $(K_C, \mathsf{EMM}_C) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_C\big)$;
  10. for all $\mathbf{c} \in \mathsf{DB}^\intercal$,
      (a) for all $v \in \mathbf{c}$ and $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$,
          i. compute $\mathsf{mtk}_{\mathbf{r}} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \chi(\mathbf{r})\Big)$,
      (b) set
          $$\mathsf{MM}_V\Big[\big\langle v, \chi(\mathbf{c})\big\rangle\Big] := \Big(\mathsf{mtk}_{\mathbf{r}}\Big)_{\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}};$$
  11. compute $(K_V, \mathsf{EMM}_V) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_V\big)$;
  12. for all $\mathbf{c} \in \mathsf{DB}^\intercal$,
      (a) for all $\mathbf{c}' \in \mathsf{DB}^\intercal$ such that $\mathsf{dom}(\mathsf{att}(\mathbf{c}')) = \mathsf{dom}(\mathsf{att}(\mathbf{c}))$,
          i. initialize an empty tuple $\mathbf{t}$;
          ii. for all rows $\mathbf{r}_i$ and $\mathbf{r}_j$ in $\mathbf{c}$ and $\mathbf{c}'$, such that $\mathbf{c}[i] = \mathbf{c}'[j]$,
              A. compute $\mathsf{mtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \chi(\mathbf{r}_i)\Big)$;
              B. compute $\mathsf{mtk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_R, \chi(\mathbf{r}_j)\Big)$;
              C. add $(\mathsf{mtk}_i, \mathsf{mtk}_j)$ to $\mathbf{t}$;
          iii. set
              $$\mathsf{MM}_{\mathbf{c}}\Big[\Big\langle \chi(\mathbf{c}), \chi(\mathbf{c}')\Big\rangle\Big] := \mathbf{t};$$
      (b) compute $(K_{\mathbf{c}}, \mathsf{EMM}_{\mathbf{c}}) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Setup}\big(1^k, \mathsf{MM}_{\mathbf{c}}\big)$;

[a] Note that we omit the description of Dec since it simply decrypts every cell of R.

Figure 13 The OPX scheme (Part 1). [5]



- $\mathsf{Setup}(1^k, \mathrm{Q})$:

13. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
    (a) for all $v \in \mathbf{c}$,
        i. compute $K_v \leftarrow F_{K_F}(\chi(\mathbf{c}) \| v)$;
        ii. set for all $\mathbf{r} \in \mathsf{DB}_{\mathbf{c}=v}$,
        $$\mathsf{SET} := \mathsf{SET} \bigcup \left\{ H(K_v \| \mathsf{rtk}) \right\},$$
        where $\mathsf{rtk} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_R, \chi(\mathbf{r}))$;
14. for all $\mathbf{c} \in \mathsf{DB}^{\intercal}$,
    (a) for all $\mathbf{c}' \in \mathsf{DB}^{\intercal}$ such that $\mathsf{dom}(\mathsf{att}(\mathbf{c}')) = \mathsf{dom}(\mathsf{att}(\mathbf{c}))$,
        i. initialize an empty tuple $\mathbf{t}$;
        ii. for all $\mathbf{r}_i, \mathbf{r}_j \in [m]$ such that $\mathbf{c}[i] = \mathbf{c}'[j]$,
        A. add $(\mathsf{rtk}_i, \mathsf{rtk}_j)$ to $\mathbf{t}$ where
        $$\mathsf{rtk}_i \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_R, \chi(\mathbf{r}_i)) \quad \text{and} \quad \mathsf{rtk}_j \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}(K_R, \chi(\mathbf{r}_i j).$$
        iii. for all rtk s.t. $(\mathsf{rtk}, \cdot) \in \mathbf{t}$, set
        $$\mathsf{MM}_{\mathbf{c},\mathbf{c}'}\left[\mathsf{rtk}\right] := \left(\mathsf{rtk}'\right)_{(\mathsf{rtk},\mathsf{rtk}') \in \mathbf{t}}$$
    (b) compute $(K_{\mathbf{c},\mathbf{c}'}, \mathsf{EMM}_{\mathbf{c},\mathbf{c}'}) \leftarrow \Sigma^{\pi}_{\mathsf{MM}}.\mathsf{Setup}(1^k, \mathsf{MM}_{\mathbf{c},\mathbf{c}'})$;
15. output $K = (K_1, K_R, K_C, K_V, \{K_{\mathbf{c}}\}_{\mathbf{c} \in \mathsf{DB}^{\intercal}}, K_F, \{K_{\mathbf{c},\mathbf{c}'}\}_{\mathbf{c},\mathbf{c}' \in \mathsf{DB}^{\intercal}})$ and $\mathsf{EDB} = (\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, (\mathsf{EMM}_{\mathbf{c},\mathbf{c}'})_{\mathbf{c},\mathbf{c}' \in \mathsf{DB}^{\intercal}}, \mathsf{SET}, (\mathsf{EMM}_{\mathbf{c}})_{\mathbf{c} \in \mathsf{DB}^{\intercal}})$.

Figure 14 The OPX scheme (Part 2). [5]

- $\mathsf{Token}(K, Q)$:
  1. initialize a token tree TK with empty nodes and with the same structure as Q;
  2. for every node N accessed in a post-traversal order in Q ,
     (a) if $N \equiv \sigma_{\mathsf{att}=a}(\mathbf{T})$ then set TK_N to

$$\mathsf{stk} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_V, \langle a, \chi(\mathsf{att})\Big);$$

     (b) if $N \equiv \sigma_{\mathsf{att}=a}(\mathbf{R_{in}})$ then set TK_N to $(\mathsf{mtk}, \mathtt{pos})$ where

$$\mathsf{mtk} \leftarrow F_{K_F}\Big(\chi(\mathsf{att}) \| a\Big)$$

     and $\mathtt{pos}$ denotes the position of att in $\mathbf{R_{in}}$.
     (c) if $N \equiv \mathbf{T}_1 \bowtie_{\mathsf{att}_1=\mathsf{att}_2} \mathbf{T}_2$ then set TK_N to $(\mathsf{jtk}, \mathtt{pos})$ where

$$\mathsf{jtk} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_{\mathsf{att}_1}, \Big\langle \chi(\mathsf{att}_1), \chi(\mathsf{att}_2)\Big\rangle\Big),$$

     and $\mathtt{pos}$ is the position of attribute att_1 in $\mathbf{R_{in}}$.
     (d) if $N \equiv \mathbf{T} \bowtie_{\mathsf{att}_1=\mathsf{att}_2} \mathbf{R_{in}}$ then set the corresponding node in TK to $(\mathsf{etk}, \mathtt{pos}_1, \mathtt{pos}_2)$ where

$$\mathsf{etk} := K_{\mathsf{att}_1,\mathsf{att}_2};$$

     and $\mathtt{pos}_1$, $\mathtt{pos}_2$ are the positions of the attributes att_1, att_2 in $\mathbf{R_{in}}$, respectively.
     (e) if $N \equiv \mathbf{R}_{\mathbf{in}}^{(l)} \bowtie_{\mathsf{att}_1=\mathsf{att}_2} \mathbf{R}_{\mathbf{in}}^{(r)}$ then set TK_N to $(\mathtt{pos}_1, \mathtt{pos}_2)$ where $\mathtt{pos}_1$ and $\mathtt{pos}_2$ are the column positions of att_1 and att_2 in $\mathbf{R}_{\mathbf{in}}^{(l)}$ and $\mathbf{R}_{\mathbf{in}}^{(r)}$, respectively.
     (f) if $N \equiv \pi_{\mathsf{att}}(\mathbf{T})$ then set TK_N to ptk where

$$\mathsf{ptk} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Token}\Big(K_C, \chi(\mathsf{att}_i)\Big).$$

     (g) if $N \equiv \pi_{\mathsf{att}_1,\cdots,\mathsf{att}_z}(\mathbf{R_{in}})$ then set TK_N to

$$\Big(\mathtt{pos}_1, \cdots, \mathtt{pos}_z\Big),$$

     where $\mathtt{pos}_i$ is the column position of att_i in $\mathbf{R_{in}}$.
     (h) if $N \equiv [a]$ then set TK_N to $[\mathsf{Enc}_{K_1}(a)]$.
     (i) if $N \equiv \times$ then set TK_N to $\times$.
  3. output TK.

Figure 15 The OPX scheme (Part 3). [5]

- Query(EDB, tk):
  1. parse EDB as $(\mathsf{EMM}_R, \mathsf{EMM}_C, \mathsf{EMM}_V, (\mathsf{EMM}_{\mathbf{c},\mathbf{c}'})_{\mathbf{c},\mathbf{c}' \in \mathsf{DB}^\intercal}, \mathsf{SET}, (\mathsf{EMM}_{\mathbf{c}})_{\mathbf{c} \in \mathsf{DB}^\intercal})$.
  2. for every node N accessed in a post-traversal order in TK,
     - if $N \equiv \mathsf{stk}$, it computes

$$(\mathsf{rtk}_1, \cdots, \mathsf{rtk}_s) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Query}\left(\mathsf{stk}, \mathsf{EMM}_V\right),$$

       and sets $\mathbf{R_{out}} := (\mathsf{rtk}_1, \cdots, \mathsf{rtk}_s)$;
     - if $N \equiv (\mathsf{mtk}, \mathbf{pos})$, then for all rtk in $\mathbf{R_{in}}$ in the column at position pos, if

$$H(\mathsf{mtk} \| \mathsf{rtk}) \notin \mathsf{SET}$$

       then it removes the row from $\mathbf{R_{in}}$. Finally, it sets $\mathbf{R_{out}} := \mathbf{R_{in}}$;
     - if $N \equiv (\mathsf{jtk}, \mathbf{pos})$, then it computes

$$\left((\mathsf{rtk}_1, \mathsf{rtk}'_1), \ldots, (\mathsf{rtk}_s, \mathsf{rtk}'_s)\right) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{jtk}, \mathsf{EMM}_{\mathbf{pos}}),$$

       and sets

$$\mathbf{R_{out}} := \left((\mathsf{rtk}_i, \mathsf{rtk}'_i)\right)_{i \in [s]};$$

     - if $N \equiv (\mathsf{etk}, \mathbf{pos}_1, \mathbf{pos}_2)$, then for each row $\mathbf{r}$ in $\mathbf{R_{in}}$, it computes $\mathsf{ltk} \leftarrow \Sigma^{\pi}_{\mathsf{MM}}.\mathsf{Token}(\mathsf{etk}, \mathsf{rtk})$, and

$$(\mathsf{rtk}_1, \cdots, \mathsf{rtk}_s) \leftarrow \Sigma^{\pi}_{\mathsf{MM}}.\mathsf{Query}(\mathsf{ltk}, \mathsf{EMM}_{\mathbf{pos}_1, \mathbf{pos}_2}),$$

       where $\mathsf{rtk} = \mathbf{r}[\mathsf{att}_{\mathbf{pos}_2}]$, and appends the new rows $\left(\mathsf{rtk}_i\right)_{i \in [s]} \times \mathbf{r}$ to $\mathbf{R_{out}}$;
     - if $N \equiv (\mathbf{pos}_1, \mathbf{pos}_2)$, then it sets

$$\mathbf{R_{out}} := \mathbf{R}^{(l)}_{\mathbf{in}} \bowtie_{\mathbf{pos}_1 = \mathbf{pos}_2} \mathbf{R}^{(r)}_{\mathbf{in}},$$

       where $\mathbf{R}^{(l)}_{\mathbf{in}}$ and $\mathbf{R}^{(r)}_{\mathbf{in}}$ are the left and right input respectively;
     - if $N \equiv \mathsf{ptk}$ then it computes

$$(\mathsf{ct}_1, \cdots, \mathsf{ct}_s) \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Query}\left(\mathsf{ptk}, \mathsf{EMM}_C\right)$$

       and sets $\mathbf{R_{out}} := (\mathsf{ct}_1, \cdots, \mathsf{ct}_s)$;
     - if $N \equiv (\mathbf{pos}_1, \cdots, \mathbf{pos}_z)$, then it computes $\mathbf{R_{out}} := \pi_{\mathbf{pos}_1, \cdots, \mathbf{pos}_z}(\mathbf{R_{in}})$;
     - if $N \equiv \times$ then it computes

$$\mathbf{R_{out}} := \mathbf{R}^{(l)}_{\mathbf{in}} \times \mathbf{R}^{(r)}_{\mathbf{in}};$$

  3. it replaces each cell rtk in $\mathbf{R}^{\mathbf{root}}_{\mathbf{out}}$ by $\mathsf{ct} \leftarrow \Sigma_{\mathsf{MM}}.\mathsf{Query}(\mathsf{rtk}, \mathsf{EMM}_R)$;
  4. output $\mathbf{R}^{\mathbf{root}}_{\mathbf{out}}$.

Figure 16 The OPX scheme (Part 4). [5]

OPX is shown to be adaptively-semantically secure, meaning that only negligible information about the plaintext can be extracted from the generated ciphertext, with respect to a well-specified leakage profile. It is stated that, like the SPX construction, OPX is composed of a “black-box component” in the sense that it comes from the underlying STE schemes, and a “non-black-box component” that comes from OPX directly. The Setup leakage captures what a persistent attacker learns by only observing the encrypted structure and before observing any query execution. The Query leakage captures what a persistent attacker learns when it observes the token and query execution. It is represented as a leakage tree that has the same form as of the query tree.

4. PRACTICAL IMPLEMENTATIONS OF DATABASE ENCRYPTION SCHEMES

4.1 CryptDB PPE Based System

CryptDB [14] [17] [36] is a system that is based on the Property Preserving Encryption construction and was designed to provide confidentiality and protection against attacks for applications that are dependent on SQL databases. It works by executing SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. It has also the ability to 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. In this way a database administrator never gets access to decrypted data and even if every server is compromised an attacker cannot decrypt the data of any user who is not logged in. As mentioned above, CryptDB works by intercepting all SQL queries in a database proxy, which rewrites queries in order to be executed on encrypted data. The proxy encrypts and decrypts all data, and changes some query operators, while preserving the semantics of the query. The database server never receives decryption keys to the plaintext so it never sees sensitive data, ensuring that a curious database admin cannot gain access to private information. To prevent application, proxy and database server compromises, developers who take advantage of the CryptDB implementation need to annotate their SQL schema to define different principals. These principals' keys will allow decrypting different parts of the database. Developers also need to make configurations to their application in order to provide their encryption keys to the proxy. The proxy determines what parts of the database should be encrypted under what key. It is also stated that although CryptDB protects data confidentiality, it does not ensure the integrity, freshness, or completeness of results returned to the application. An attacker can still delete any or all the data stored in the database. Also similar attacks on user machines such as cross-site scripting, are outside of the scope of CryptDB.

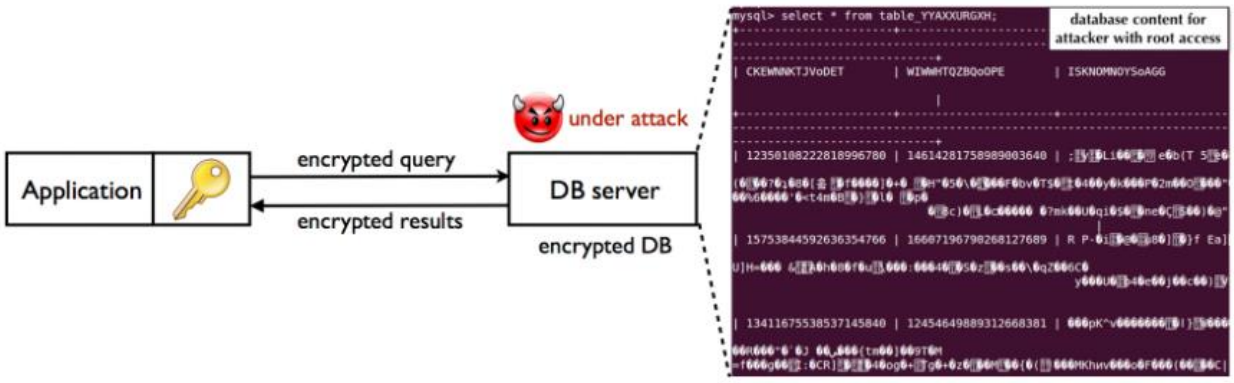


Figure 17 CryptDB System Overview [36]

There are two main threats that the system was designed to counter. The first was the “curious” database administrator who tries to learn private data. The goal here was confidentiality and not integrity or availability. The attacker was assumed to be passive, meaning he wants to learn confidential data but does not change queries issued by the application, query results or the data in the database. This threat includes database server software compromises, root access to database server machines as well as access to the RAM of physical machines. This threat is considered increasingly important with the rise in database consolidation inside enterprise data centers, outsourcing of databases to public cloud computing infrastructures and the use of third-party database admins.

The second threat was an adversary that gains complete control of the application and database management servers. The solution was to encrypt different data items with different keys. To determine the key that should be used for each item, developers need to annotate the application’s database schema to express finer-grained confidentiality Policies. CryptDB cannot provide any guarantees for users that are logged into the application. CryptDB leaks at most the data of currently active users for the duration of the compromise, even if the proxy used for encrypted queries behaves in a Byzantine fashion. It is stated that “duration of a compromise”, means the interval from the start of the compromise until any trace of the compromise has been erased from the system. For a read SQL injection attack, the duration of the compromise spans the attacker’s SQL queries. during an attack but can still ensure the confidentiality of logged-out users’ data.

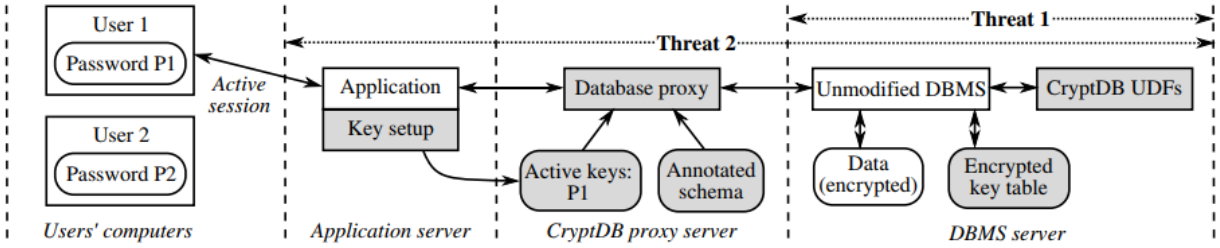


Figure 18 CryptDB's architecture consisting of two parts: a database proxy and an unmodified DBMS. [14]

The first challenge that was faced to counter the above threats was the tension between minimizing the amount of confidential information revealed to the database management server and the ability to efficiently execute a variety of queries, as current approaches for computing over encrypted data were either too slow or do not provide adequate confidentiality. Encrypting data with a cryptosystem, such as AES, would prevent the database management server from executing many SQL queries. In this case, the only practical solution would be to give the DBMS server access to the decryption key, but that would allow an attacker to also gain access to all data. The second challenge was to minimize the amount of data leaked when an adversary compromised the application server. The application must be able to access decrypted data because arbitrary computation on encrypted data is not very practical. In that case, a compromised application should only obtain a limited amount of decrypted data. As mentioned, a solution of assigning each user a different database encryption key for their data does not work for applications with shared data, such as bulletin boards and conference review sites.

The CryptDB system uses three key ideas to address these challenges. The first is to execute SQL queries over encrypted data. For this idea an SQL-aware encryption strategy is used which leverages the fact that all SQL queries are made up of set of primitive operators, such as equality checks, order comparisons, aggregates, and joins. It is mentioned that CryptDB encrypts each data item in a way that allows the database management server to execute on the transformed data by taking advantage of known encryption schemes as well as a new privacy-preserving cryptographic method for joins. Also, CryptDB is efficient because it mostly uses symmetric-key encryption, avoids fully homomorphic encryption, and runs on unmodified database management server software.

The second idea is adjustable query-based encryption. Because some encryption schemes leak more information than others about the data in the database but are

required to process certain queries CryptDB carefully adjusts the SQL-aware encryption scheme for any given data item, depending on the queries observed at run-time. In order to implement these adjustments efficiently, CryptDB uses onions of encryption, which are a novel way to compactly store multiple ciphertexts within each other in the database and avoid expensive re-encryptions.

The third idea is chaining encryption keys to user passwords, in a way that each data item in the database can be decrypted only through another chain of keys rooted in the password of one of the users with access to that data. In this way if the user is not logged into the application or an attacker does not know his password, the attacker cannot decrypt the user's data even if the database server becomes fully compromised. The CryptDB system allows the developer to provide policy annotations over the application's SQL schema to specify which users have access to each data item in order for the chain of keys to be constructed.

CryptDB enables applications to execute SQL queries on encrypted data the same way as if it were executing on plaintext data. In this way existing applications do not need to be changed. The database server query plan for an encrypted query is typically the same as for the original query, except that the operators comprising the query, such as selections, projections, joins, aggregates, and orderings, are performed on ciphertexts, and use modified operators in some cases. As stated, the CryptDB's proxy stores a secret master key, the database schema, and the current encryption layers of all columns. The database server sees an anonymized scheme, encrypted user data, and some auxiliary tables used by CryptDB. CryptDB also equips the server with CryptDB-specific user-defined functions that enable the server to compute on ciphertexts for certain operations. Processing a query in CryptDB involves four steps. At first the application issues a query which then is intercepted by the proxy and gets rewritten in a way that each table and column name is anonymized using the master key and each constant in the query is encrypted with an encryption scheme which is best suited for the desired operation. Then the proxy checks if the database management server should be given keys to adjust encryption layers before executing the query and if so, issues an UPDATE query that invokes a user defined function to adjust the encryption layer of the appropriate column. After that, the proxy forwards the encrypted query to the database server and then the query is executed using standard SQL. Finally, the server returns the encrypted query result, which the proxy decrypts and returns to the application.

Regarding SQL-aware encryption CryptDB uses several encryption types, including a number of existing cryptosystems, an optimization of a recent scheme, and a new cryptographic primitive for joins. This first encryption type is Random (RND). As stated RND provides the maximum security in CryptDB, especially indistinguishability under an adaptive chosen-plaintext attack (IND-CPA); the scheme is probabilistic, meaning that two equal values are mapped to different ciphertexts with overwhelming probability. On the other hand, RND 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). The second encryption type is the Deterministic Encryption Scheme (DET). As stated DET has a slightly weaker guarantee, yet it still provides strong security. It leaks only which encrypted values correspond to the same data value, by deterministically generating the same ciphertext for the same plaintext. This encryption layer allows the server to perform equality checks, which means it can perform selects with equality predicates, equality joins and statements such as GROUP BY, COUNT and DISTINCT. The third encryption type is the Homomorphic Encryption Scheme (HOM). As stated, Homomorphic Encryption is a secure probabilistic encryption scheme (IND-CPA secure), allowing the server to perform computations on encrypted data with the result decrypted at the proxy. While fully homomorphic encryption is prohibitively slow, homomorphic encryption for specific operations is efficient. The fourth encryption type is Join (JOIN and OPE-JOIN). As stated, JOIN as a separate encryption scheme is necessary to allow equality joins between two columns, because different keys are used for Deterministic Encryption to prevent cross-column correlations. JOIN also supports all operations allowed by Deterministic Encryption and enables the server to determine repeating values between two columns. Moreover OPE-JOIN enables joins by order relations. The final encryption type used is Word search known as SEARCH. Its is used to perform searches on encrypted text to support operations such as MySQL’s LIKE operator. The cryptographic protocol of Song et al. [37] is used, which had not been previously implemented by its authors. As state the protocol is used in a different way, which results in better security guarantees. For each column needing SEARCH, the text gets split into keywords using a delimiter. Then repetitions in these words are removed, the positions of the words are randomly permuted and then each of the word are encrypted using Song et al.’s scheme while padding each word to the same size. SEARCH is nearly as secure as RND: the encryption does not reveal to the DBMS server whether a certain word repeats in multiple rows, but it leaks the number of

keywords encrypted with SEARCH. For this reason, an attacker may be able to estimate the number of distinct or duplicate words.

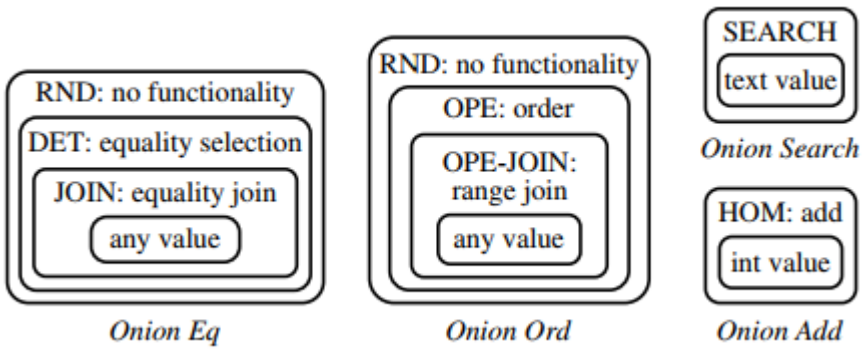


Figure 19 Onion encryption layers and the classes of computation they allow. [14]

Regarding adjustable Query-based Encryption the chosen idea was to encrypt each data item in one or more onions where each value is dressed in layers of increasingly stronger encryption. Each layer of each onion enables certain kinds of functionality as explained above. Outermost layers such as Random and Homomorphic Encryption provide maximum security, whereas inner layers such as Order Preserving Encryption provide more functionality. It is stated that Multiple onions are needed in practice, mainly because the computations supported by different encryption schemes are not always strictly ordered, and because of performance considerations. CryptDB may not maintain all onions for each column. For example, the Search onion does not make sense for integers, and the Add onion does not make sense for strings. For each layer of each onion, the proxy uses the same key for encrypting values in the same column, and different keys across tables, columns, onions, and onion layers. Using the same key for all values in a column allows the proxy to perform operations on a column without having to compute separate keys for each row that will be manipulated. Using different keys across columns prevents the server from learning any additional relations. Each onion starts out encrypted with the most secure encryption scheme. As the proxy receives SQL queries from the application, it determines whether layers of encryption need to be removed. The proxy never decrypts the data past the least-secure encryption onion layer. CryptDB implements onion layer decryption using user defined functions that run on the database server. It is states that each column decryption should be included in a transaction to avoid consistency problems with clients accessing columns being adjusted. Onion decryption is performed entirely by the DBMS server. In

the steady state, no server-side decryptions are needed, because onion decryption happens only when a new class of computation is requested on a column.

Employees

ID	*Name*
23	Alice

Table1

C1-IV	*C1-Eq*	*C1-Ord*	*C1-Add*	*C2-IV*	*C2-Eq*	*C2-Ord*	*C2-Search*
x27c3	x2b82	xcb94	xc2e4	x8a13	xd1e3	x7eb1	x29b0

Figure 20 Data layout of tables created by the application and the equivalent at the server. [14]

Regarding executing over encrypted data, it is stated that once the onion layers in the database server are at the layer necessary to execute a query, the proxy transforms the query to operate on these onions. Particularly the proxy replaces column names in a query with corresponding onion names, based on the class of computation performed on that column. Also, the proxy replaces each constant in the query with a corresponding onion encryption of that constant, based on the computation in which it is used. Finally, the server replaces certain operators with user defined function-based counterparts. For example, the SUM aggregate operator and the '+' column-addition operator must be replaced with an invocation of a user defined function that performs homomorphic addition of ciphertexts. Equality and order operators do not need such replacement and can be applied directly to the Deterministic Encryption and Order Preserving Encryption ciphertexts. Once the proxy has transformed the query, it sends the query to the DBMS server, receives query results, decrypts the results using the corresponding onion keys, and sends the decrypted result to the application.

As stated, there are two kinds of joins supported by CryptDB. First are "equi-joins" in which the join predicate is based on equality. To perform an "equi-join" of two encrypted columns, the columns should be encrypted with the same key so that the server can see matching values between the two columns. At the same time, to provide better privacy, the DBMS server should not be able to join columns for which the application did not request a join, so columns that are never joined should not be encrypted with the same keys. If the queries that can be issued, or the pairs of columns that can be joined, are known a priori, "equi-join" is easy to support: CryptDB can use the Deterministic encryption scheme with the same key for each group of columns that are joined together. The difficult case is when the proxy does not know at the start the set of columns to be joined, and hence does not know which columns should be encrypted with matching keys. In order to solve this problem CryptDB introduces a new cryptographic primitive called JOIN-ADJ which allows the database server to adjust the key of each column at runtime. Intuitively, JOIN-ADJ can be thought of as a keyed

cryptographic hash with the additional property that hashes can be adjusted to change their key without access to the plaintext. JOIN-ADJ is a deterministic function of its input, meaning that if two plaintexts are equal, the corresponding JOIN-ADJ values are also equal. It is also collision-resistant and has a sufficiently long output length to allow to assume that collisions never happen in practice. Each column is initially encrypted at the JOIN layer using a different key, thus preventing any joins between columns. When a query requests a join, the proxy gives the DBMS server an onion key to adjust the JOIN-ADJ values in one of the two columns, so that it matches the JOIN-ADJ key of the other column. After the adjustment, the columns share the same JOIN-ADJ key, allowing the DBMS server to join them for equality. The DET components of JOIN remain encrypted with different keys. For range joins, it is stated that a similar dynamic re-adjustment scheme is difficult to construct due to lack of structure in OPE schemes. Instead, CryptDB requires that pairs of columns that will be involved in such joins be declared by the application ahead of time, so that matching keys are used for layer OPE-JOIN of those columns. If not, the same key will be used for all columns at layer OPE-JOIN.

There are some security improvements that can be implemented in CryptDB. It is stated that application developers can specify the lowest onion encryption layer that may be revealed to the server for a specific column. In this setting, developers can ensure that the proxy will not execute queries exposing sensitive relations to the server. Moreover, it is stated that although CryptDB can evaluate several predicates on the server, evaluating them in the proxy can improve security by not revealing additional information to the server. One such common use case is a SELECT query that sorts on one of the selected columns, without a LIMIT clause on the number of returned columns. Since the proxy receives the entire result set from the server, sorting these results in the proxy does not require a significant amount of computation, and does not increase the bandwidth requirements. Doing so avoids revealing the OPE encryption of that column to the server. Furthermore, it is mentioned that CryptDB provides a training mode feature, which allows developers to provide a trace of queries and get the resulting onion encryption layers for each field, along with a warning in case some query is not supported. Developers can then examine the resulting encryption levels to understand what each encryption scheme leaks. If some onion level is too low for a sensitive field, it should be arranged to have the query processed in the proxy, or to process the data in some other fashion, such as by using a local instance of SQLite. Finally, it is stated that in cases when an application performs infrequent queries

requiring a low onion layer, CryptDB could be extended to re-encrypt onions back to a higher layer after the infrequent query finishes executing. With this approach leakage to attacks happening in the time windows when the data is at the higher onion layer, is reduced.

CryptDB also supports performance optimizations. CryptDB encrypts all fields and creates all applicable onions for each data item based on its type by default. If many columns are not sensitive, developers can instead provide explicit annotations indicating the fields that are considered sensitive and leave the remaining fields in plaintext. Moreover, it is stated that if the developers know some of the queries ahead of time, as is the case for many web applications, they can use the training mode described above to adjust onions to the correct layer, avoiding the overhead of runtime onion adjustments. If the exact query set is provided or annotations that certain functionality is not needed on some columns, CryptDB has the ability to discard onions and onion layers that are not needed or discard the random IV needed for RND encryption for some columns. Furthermore, the proxy spends a significant amount of time encrypting values used in queries with Order Preserving Encryption and Homomorphic Encryption. To reduce this cost, the proxy pre-computes and caches encryptions of frequently used constants under different keys. Since Homomorphic Encryption is probabilistic, ciphertexts cannot be reused. This optimization reduces the amount of CPU time spent by the proxy on Order Preserving encryption, and assuming the proxy is occasionally idle to perform Homomorphic Encryption pre-computation, it removes Homomorphic Encryption from the critical path.

CryptDB was evaluated in four aspects. First the difficulty of modifying an application to run on top of the system, then the queries and applications it can support, the level of security it provides, and lastly the performance impact of using it. To make this evaluation seven applications were used and a large trace of SQL queries. The effectiveness of the provided annotations and the important application changes were analyzed on phpBB, HotCRP, grad-apply, OpenEMR, an electronic medical record application and a web application of an MIT class. Also, for the functionality of security, analysis has been done on TPC-C and a large trace of SQL queries that came from a MySQL server at MIT, sql.mit.edu. The query trace run for about ten days and included approximately 126 million queries. In the end the overall performance of the system and a detailed analysis through microbenchmarks were evaluated on the phpBB application and on the query mix of TPC-C. In the applications except TPC-C only sensitive

columns were encrypted. In the case of TPC-C all the columns on the database were encrypted in single-principal mode in order for the performance of a fully encrypted database to be studied. The results shown that for multi-principal mode, the system required 29 to 111 annotations and of them 11 and 13 were unique and 2 to 7 lines of code for providing user passwords to the proxy. For TPC-C principal no annotations or line of code were used.

Application	Annotations	Login/logout code	Sensitive fields secured, and examples of such fields
phpBB	31 (11 unique)	7 lines	23: private messages (content, subject), posts, forums
HotCRP	29 (12 unique)	2 lines	22: paper content and paper information, reviews
grad-apply	111 (13 unique)	2 lines	103: student grades (61), scores (17), recommendations, reviews
TPC-C (single princ.)	0	0	92: all the fields in all the tables encrypted

Figure 21 CryptDB Experimental Evaluation setup [14]

For the Functional Evaluation, the queries issued from the aforementioned applications were used. On the analysis on the trace of SQL queries was found that CryptDB should be able to support operations over all but 1,094 of the 128,840 columns observed. It is stated that with in-proxy processing, CryptDB should be able to process queries over encrypted data over all but 571 of the 128,840 columns, thus supporting 99:5% of the columns. Also, it is shown that it 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.

For the Security evaluation the steady-state onion levels of different columns were studied. The minimum encryption was defined to be the weakest onion encryption scheme exposed on any of the onions of a column when onions reach a steady state. For the SQL trace approximately 6.6% of columns were at Order Preserving Encryption even with in-proxy processing and the rest encrypted columns remain at Deterministic Encryption or stronger. It is stated that out of the columns that were at Order Preserving Encryption, 3.9% are used in an ORDER BY clause with a LIMIT, 3.7% are used in an inequality comparison in a WHERE clause, and 0.25% are used in a MIN or MAX aggregate operator. Moreover, it was validated that the system's confidentiality guarantees by trying real attacks on phpBB that have been listed in the CVE database, including two known SQL injection attacks, bugs in permission checks and a bug in remote PHP file inclusion. Finally, it was found that, for users not currently logged in, the answers returned from the database management system were encrypted, even with

root access to the application server, proxy, and database server, the answers were not decryptable.

Application	Total cols.	Consider for enc.	Needs plaintext	Needs HOM	Needs SEARCH	Non-plaintext cols. with MinEnc: RND	SEARCH	DET	OPE	Most sensitive cols. at HIGH
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 22 CryptDB Steady-state onion levels for database columns required by a range of applications and traces. [14]

Regarding the performance it is stated that evaluation the tests were run on a machine with 2.4 GHz Intel Xeon E5620 4-core processors and 12 GB of RAM to run the MySQL server with version 5.1.54. For the CryptDB proxy and the clients a machine with eight 2.4 GHz AMD Opteron 8431 6-core processors and 64 GB of RAM was used. The TPC-C query mix was compared when running on an unmodified MySQL server and on a the CryptDB proxy in front of the MySQL server. Also as stated there were no onion adjustments during the TPC-C experiments. In all cases the server spent 100% of its CPU time processing queries. The overall throughput with CryptDB was found to be 21–26% lower than MySQL, depending on the exact number of cores. Along with MySQL and CryptDB proxy a strawman design was used which performed each query over data encrypted with Random Encryption using user defined functions. The strawman design then performed the query over the plaintext and re-encrypted the result. The results had shown that CryptDB's throughput penalty was greatest for Homomorphic Encryption queries that involved a SUM and for incrementing UPDATE statements. For the rest queries, which formed a larger part of the TPC-C mix, the throughput overhead was found to be modest. The strawman design performed poorly for almost all queries because the database indexes on the RND-encrypted data were useless for operations on the underlying plaintext data. It is also stated that it was found that increasing security over strawman method can benefit performance. Moreover, it was shown that there was an overall server latency increase of 20% with CryptDB, which was considered modest. The proxy added an average of 0.60 ms to a query; of that time,

24% is spent in MySQL proxy, 23% is spent in encryption and decryption, and the remaining 53% is spent parsing and processing queries. Also, the cryptographic overhead was relatively small because most of the encryption schemes were efficient. Order Preserving and Homomorphic Encryption were the slowest, but the ciphertext pre-computing and caching optimization masked the high latency of queries requiring these encryption schemes. Finally, it is stated that in all TPC-C experiments the proxy used less than 20 MB of memory. Caching ciphertexts for the 30,000 most common values for Order Preserving Encryption accounts for about 3 MB, and pre-computing ciphertexts and randomness for 30,000 values at Homomorphic Encryption required 10 MB.

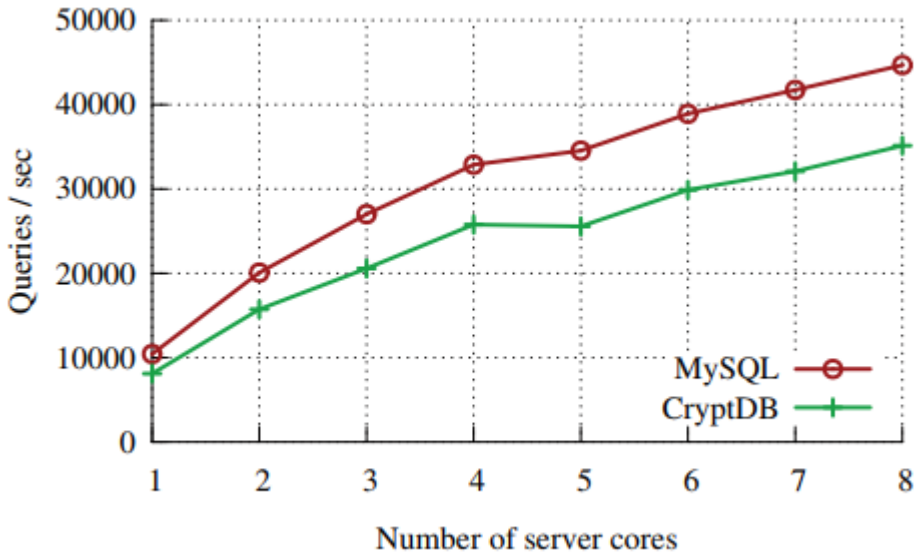


Figure 23 Throughput for TPC-C queries, for a varying number of cores on the MySQL database server. [14]

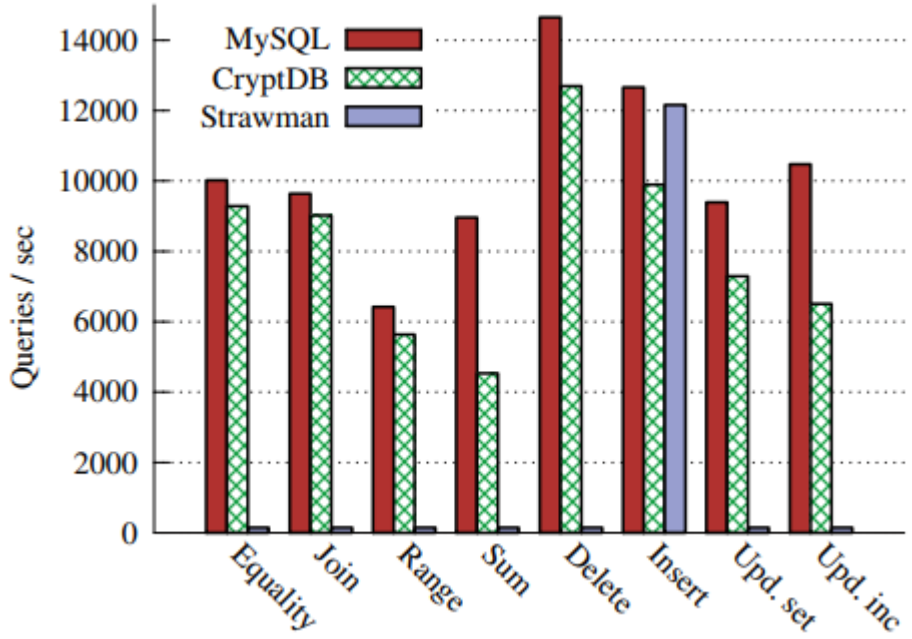


Figure 24 Throughput of different types of SQL queries from the TPC-C query mix running under MySQL, CryptDB, and the strawman design. [14]

Query (& scheme)		MySQL Server	CryptDB Server	CryptDB Proxy	CryptDB Proxy⋆
Select by =	(DET)	0.10 ms	0.11 ms	0.86 ms	0.86 ms
Select join	(JOIN)	0.10 ms	0.11 ms	0.75 ms	0.75 ms
Select range	(OPE)	0.16 ms	0.22 ms	**0.78** ms	28.7 ms
Select sum	(HOM)	0.11 ms	0.46 ms	0.99 ms	0.99 ms
Delete		0.07 ms	0.08 ms	0.28 ms	0.28 ms
Insert	(all)	0.08 ms	0.10 ms	**0.37** ms	16.3 ms
Update set	(all)	0.11 ms	0.14 ms	**0.36** ms	3.80 ms
Update inc	(HOM)	0.10 ms	0.17 ms	**0.30** ms	25.1 ms
Overall		0.10 ms	0.12 ms	**0.60** ms	10.7 ms

Figure 25 Server and proxy latency for different types of SQL queries. [14]

Scheme	Encrypt	Decrypt	Special operation
Blowfish (1 int.)	0.0001 ms	0.0001 ms	—
AES-CBC (1 KB)	0.008 ms	0.007 ms	—
AES-CMC (1 KB)	0.016 ms	0.015 ms	—
OPE (1 int.)	9.0 ms	9.0 ms	Compare: 0 ms
SEARCH (1 word)	0.01 ms	0.004 ms	Match: 0.001 ms
HOM (1 int.)	9.7 ms	0.7 ms	Add: 0.005 ms
JOIN-ADJ (1 int.)	0.52 ms	—	Adjust: 0.56 ms

Figure 26 Microbenchmarks of cryptographic schemes, per unit of data encrypted, measured by taking the average time over many iterations. [14]

CryptDB system is already adopted by many known companies. SAP AG has developed a system called SEED which uses most of the components of CryptDB

including adjustable encryption strategy with onions. Also, Google was motivated from CryptDB and developed an extension of BigQuery known as Encrypted BigQuery, which offers client-side encryption for a subset of query types using similar encryption schemes with that of CryptDB. Furthermore, Microsoft has incorporated in Always Encrypted SQL Server a service to enable users to encrypt several fields with CryptDBs encryption schemes to improve security. They developed Cipherbase which is a successor of CryptDB, which is enhanced with trusted hardware support for queries not supported on encryption. There already many more companies that have incorporated CryptDB over their database servers.

CryptDB was developed at MIT in 2011 and was maintained until 2014 so the original system needs an old machine in order to run properly. After 2014 the work was continued by Y. Shao [29]. As part of our study, we managed to setup a modified version of the original CryptDB system. In this version Y. Shao added new features and fixed bugs over the old implementation, thus making it more practical. This version of CryptDB can run on a newer system now on Ubuntu 16.04 and MySQL server version 5.7.32. The MySQL server and the proxy are both incorporated in the same server. We the help of another github user named Agribu [30] we managed to run a Docker image of the improved CryptDB system made by Yiwen Shao. Once the Docker was built and run, we needed to run two scripts. One to run the MySQL server and one to connect to the proxy client. Then, by accessing the proxy client we created a test database named “di”. As shown in Figure 28 by running the “SHOW DATABASES” query from the proxy client we were able to see all the databases from the proxy client. Next, we used the proxy client to create a table named “test” in “di” database and inserted a row in it as seen in Figure 29. In the SELECT query results all the data appear to be normal when executed through the proxy client. After that we connected on the MySQL server directly through the official client as shown in Figure 30. When we executed the query “SHOW DATABASES”, again we were able to see all databases. When we used the “SHOW TABLES” query the results were different this time. Our table had a different name “table_NMWVDTXRJE”. Finally, we executed a SELECT query to see the data inside the table. Again, the result set returned encrypted data in a peculiar form as shown in Figure 31. This was because all the data inside any created database in reality are encrypted. So, in case an attacker finds a way to read the real database he will not be able to read the actual data. Also it is important to mention here that the proxy client needs to be used in a secure manner because if an attacker manage to access the proxy client he can have access to all the data.

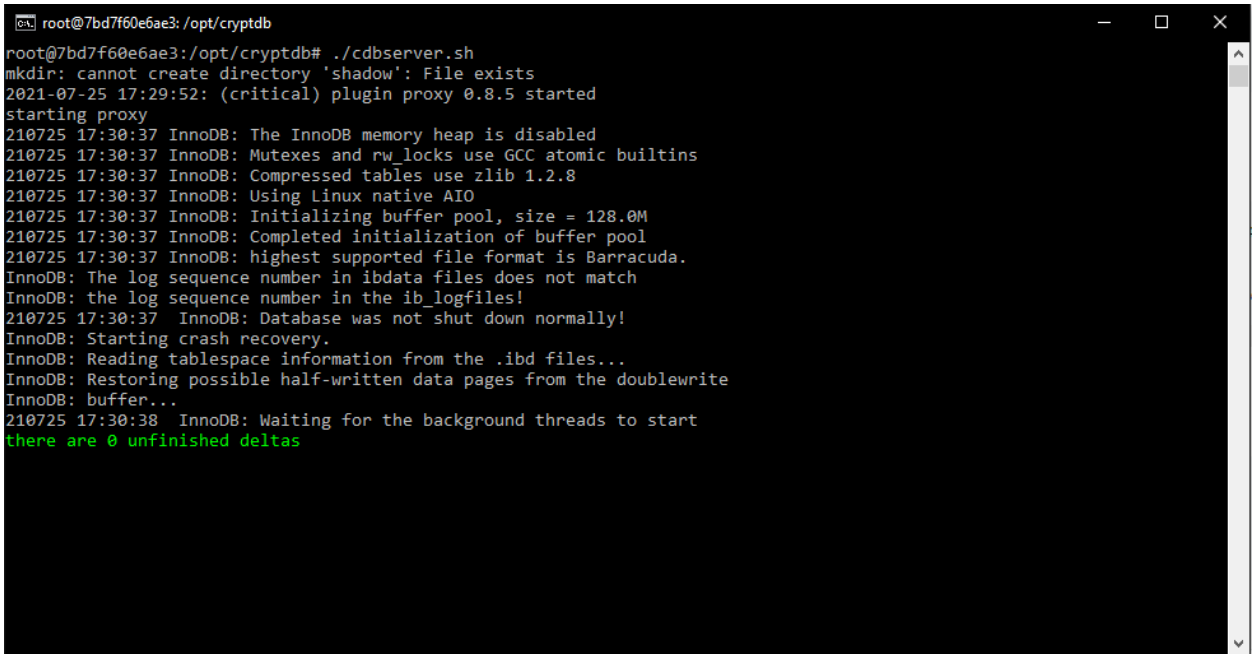


Figure 27 Server Script Log Once MySQL server is started, and an instance of proxy client is connected.

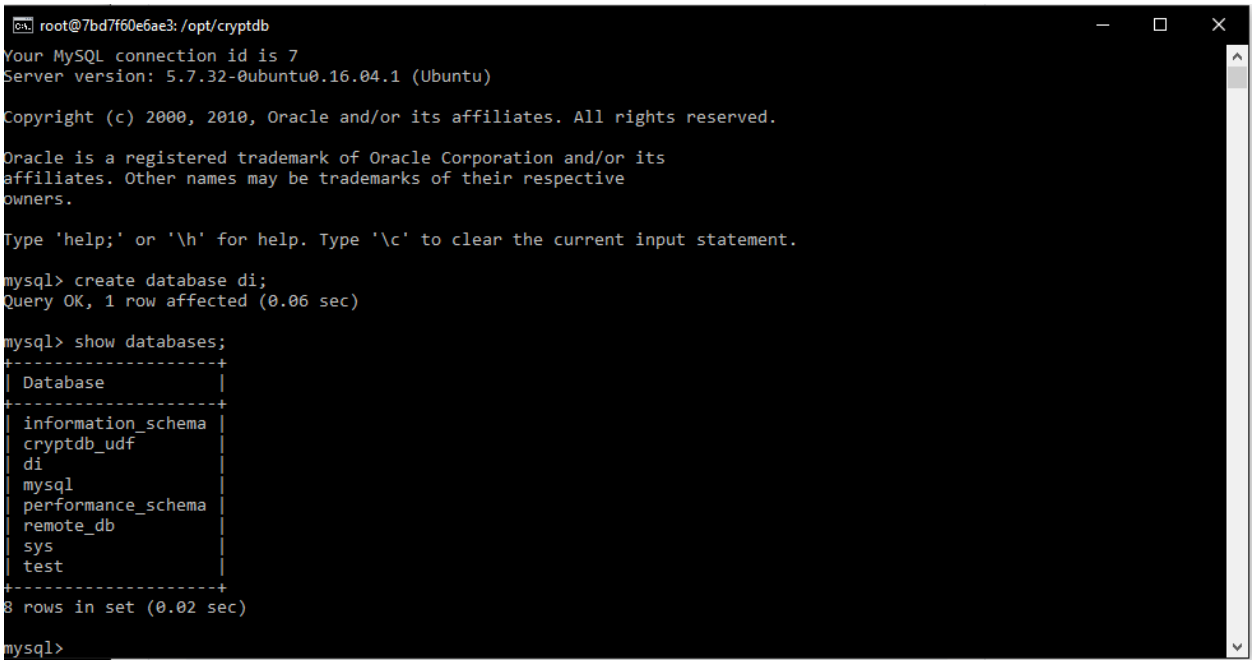


Figure 28 Complete MySQL server databases as shown from the proxy client.

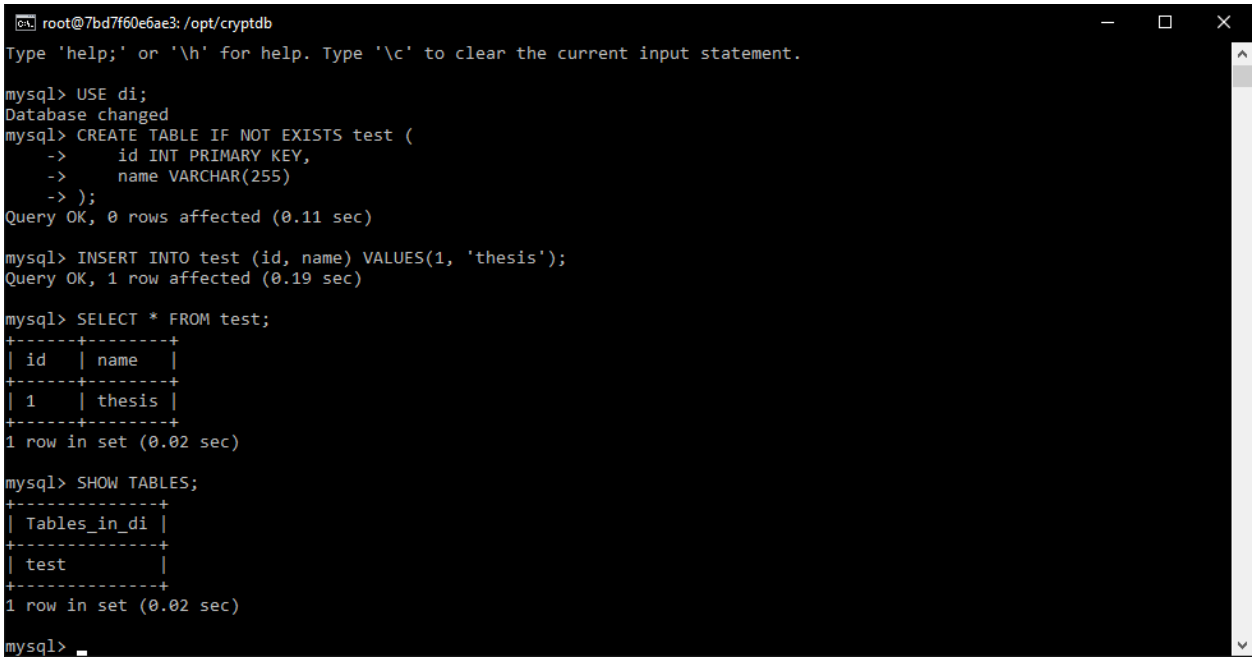


Figure 29 Table Creation and INSERT and SELECT data operations through the proxy client.

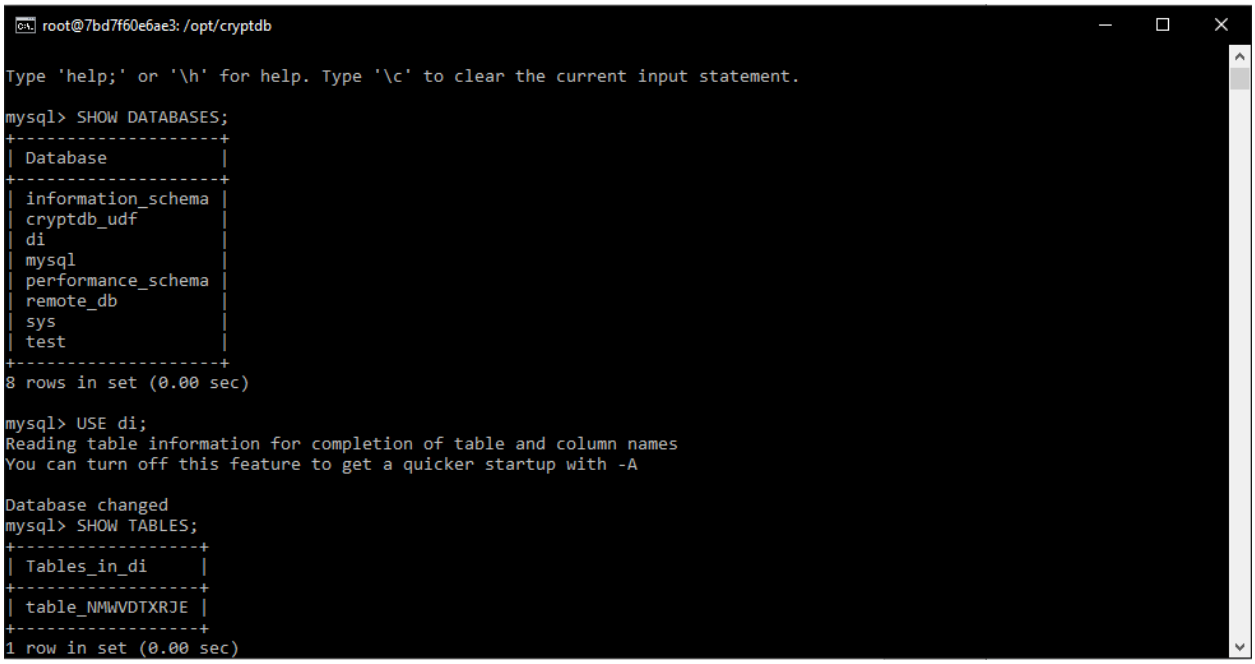


Figure 30 MySQL server connection directly through the official client.

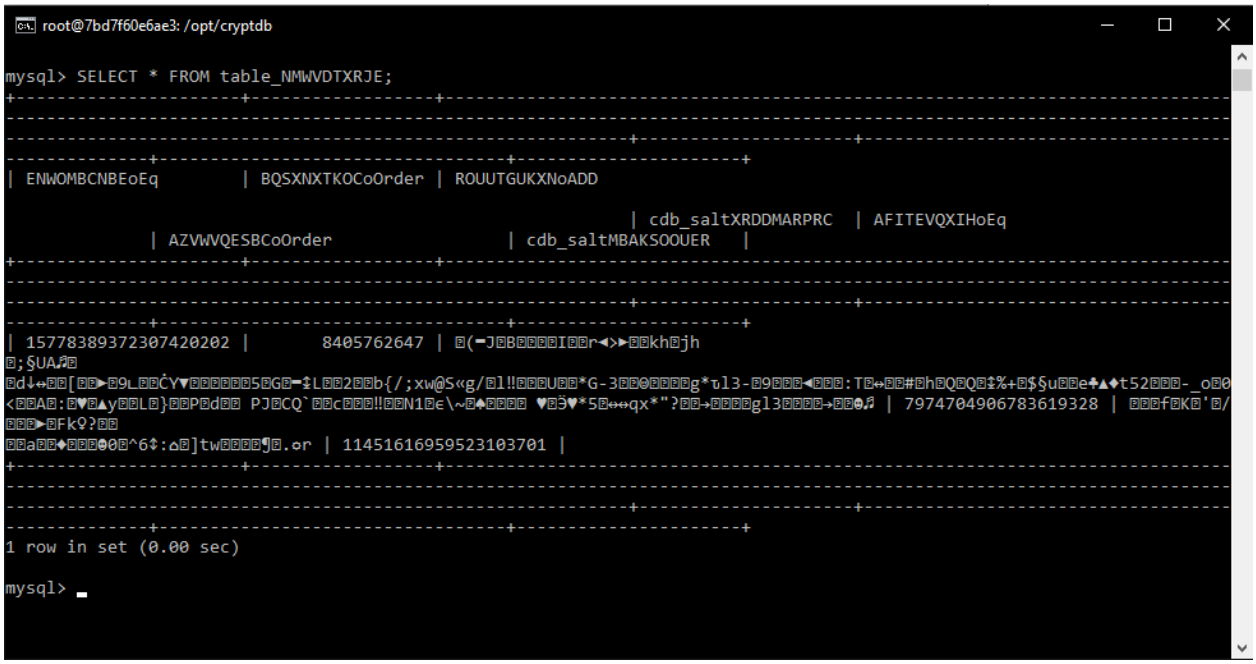


Figure 31 Result of SELECT query on the encrypted table through the official MySQL server client.

4.2 KafeDB OPX Based System

There is a more recent system called KafeDB, which is based on OPX construction and was recently introduced by Shenny Kamara [27] [31]. As stated in Kamara's paper this system is composed of the application, the client and the server. Both the application and the client are assumed to run on a secure environment and the server is assumed to be untrusted. The client is used to encrypt the database and queries and send them to the server to execute them. It is also stated that the key material is stored by the client, so the server never sees the data or queries in plaintext. Moreover, the client is stateless, meaning that it only stores the schema of the plaintext database and the cryptographic keys. There are two important components of the KafeDB system. The first one is the Crypto Engine which is responsible to for encrypting the database and queries and for decrypting the results. This engine implements the OPX construction but as stated future versions could be based on new and improved schemes. The second one is the Emulation Engine. When the database or query is encrypted, it is handed to the emulation engine which is responsible for transforming them into relational tables and SQL queries to be processed by the server. The tables and SQL queries output by the emulation engine are completely different from the tables and SQL queries produced by the actual application. It is stated that in the currently designed emulation engine the tables and SQL queries output by the emulation engine are not the same as the tables and SQL queries produced by the application. Moreover, KafeDB was designed to bulk load new data through a setup module which invokes the crypto

engine to encrypt the data into encrypted structures, and then it uses the emulation engine to reshape them into tables and indexes. It is also mentioned that KafeDB is limited in how much it can optimize queries because of encryption it cannot maintain statistics over the tables. For this matter, KafeDB does most of its query optimization at the client. the encrypted structures and query protocols used by the OPX construction are carefully designed so that the supported operations can be queried in any order. It is stated that this flexibility results in the system being optimization-friendly since it can process query plans that are produced by standard query optimizers. Finally, KafeDB supports split execution. The client splits a given query it into two kinds of subqueries. The conjunctive subqueries, which are supported by the OPX crypto engine, and other subqueries which are not. The conjunctive subqueries are processed by the crypto engine and the rest are executed locally using the results of the conjunctive subqueries.

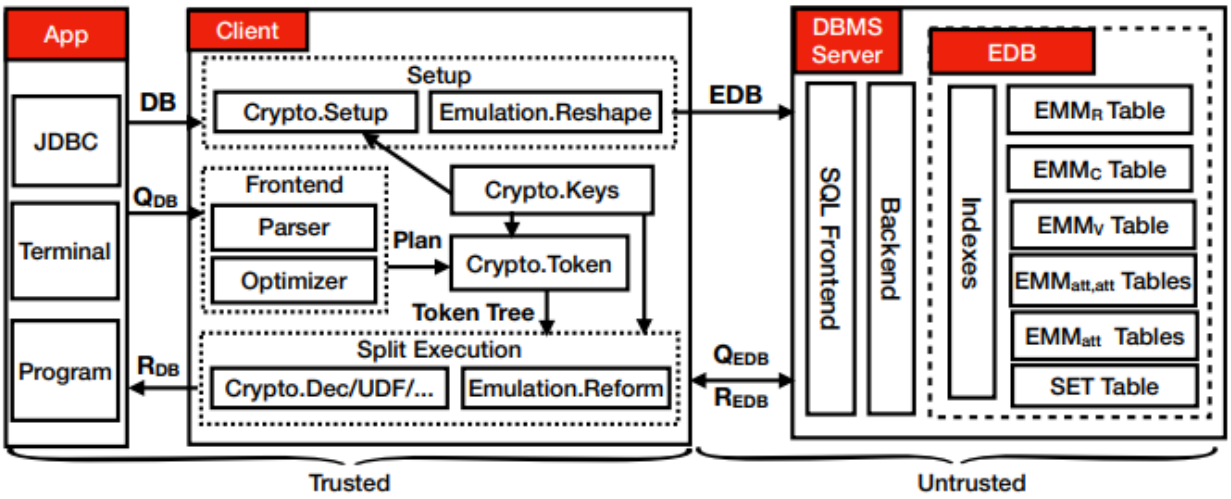


Figure 32 The KafeDB system architecture. [27]

The KafeDB system was evaluated, and its performance was compared against the CryptDB system and PostgreSQL. The main purpose of the evaluation was to demonstrate that one can design an encrypted database system without giving up completely on security, functionality and performance. It is stated that previous systems like CryptDB failed to achieve minimal setup leakage and therefore are vulnerable to multiple practical attacks. For the implementation of KafeDB the server used run PostgreSQL 9.6.2 and the client used Spark SQL 's Catalyst for query parsing and planning, and its executor to facilitate split execution. Also, for the cryptographic primitives, AES in CBC mode with PKCS7 padding was used for symmetric encryption, and HMAC-SHA-256 for pseudo-random functions. The experiments were conducted on Amazon EC2 with instance type t2.2xlarge, which had 8 CPUs, 32GB RAM and 1TB of Elastic Block Store. Also, a high memory capacity ratio was kept to the database

size, which more specifically amounted to 0.5× for KafeDB and 7.2× for plaintext PostgreSQL. Regarding the data generation the TPC-H benchmark was used with a scale factor 1, which leads to about 8.6 million rows and 1GB of data. Moreover, for the comparisons to be more accurate a modified version of CryptDB was used which supported the full TPC-H as well as a machine with slightly different RAM. Since the used code for the modified CryptDB was not open-source, and in order to draw fair comparisons, the reports were only for the query and storage multiplicative overheads incurred by these systems over a plaintext PostgreSQL.

The results shown that KafeDB was about an order of magnitude slower than CryptDB. For KafeDB, excluding three queries that timed out, the median slowdown relative to plaintext was 45.6× with a range of 3.2×-1407.9×. For CryptDB the median was 3.92× with a range of 1.04× to 55.9×. Moreover, all queries performed better with encrypted query optimizations applied. It is stated that with selection pushdown, the speedup varied from 4× to 53×. Also, without many-to-many join factorization, the single join between Customer and Supplier timed out after 24 hours, whereas the factorized joins with additionally Nation took only 12 minutes. With multi-way join flattening, the speedup was around 20×. Furthermore, the KafeDB system incurred an order of magnitude size blowup over plaintext due to both ciphertext expansion and the complexity of the encrypted structures with a multiplicative factor of 13.17×. CryptDB appeared to incur a smaller size blowup of 4.21×. It appeared that KafeDB required about an order of magnitude more time to set up than to load the plaintext into PostgreSQL with a multiplicative factor of 10.37×. Also at scale factor 10, KafeDB showed signs of limited scalability where the overhead for most queries exceeded three orders of magnitude compared to plaintext. Finally, it is stated that the KafeDB system did not undergo any system-level optimization, and the OPX scheme still has much room for improvement beyond the new support for query optimization.

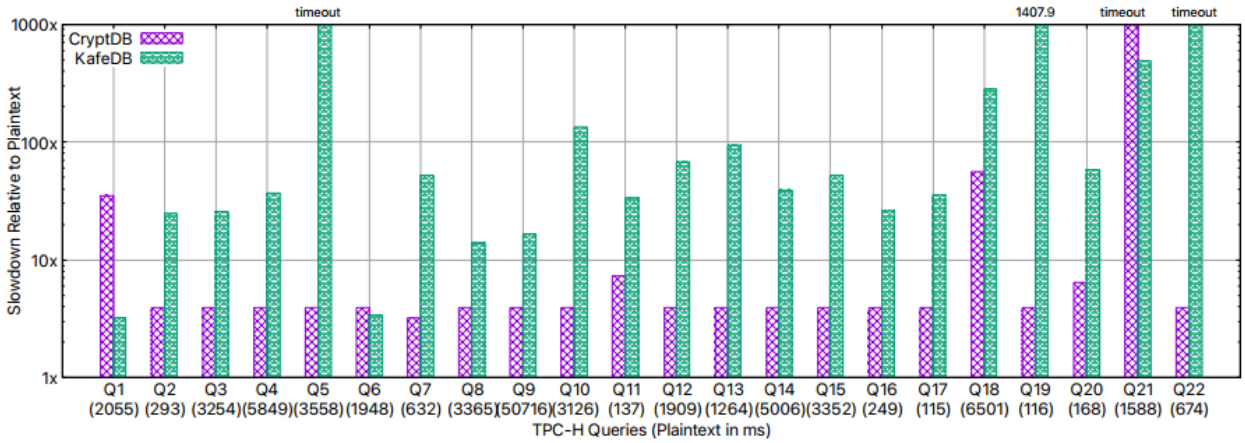


Figure 33 Comparison of KafeDB and CryptDB with TPC-H Benchmark with scale factor 1. [27]

5. CONCLUSION

In this thesis we study the problem of security and privacy over data used on external providers which nowadays with the rapid development of technology has become an important matter. As we discovered there are many encryption schemes available today that already have applications in everyday life. These encryption schemes are essential in order to form complex constructions that can be used on database data encryption between applications and database server transactions. It seems that these encryption schemes can be used efficiently in conjunction in fully pledged systems to tackle the problem. The results that the proposed systems have shown seem a lot promising in both security and performance as they offer more advantages than disadvantages. Moreover, the systems achieve in great extent all important design principles such as minimal leakage, low asymptotic overhead, optimization friendliness, rich query friendliness and legacy friendliness. Although these systems appear to be robust, feature rich, highly configurable and extendable we believe further work needs to be done on the subject. This is because technology progresses rapidly and every system as good and may seem today can become obsolete very fast in the future. Finally, it is important to note that in the future with the arrival of Quantum Computing, many custom systems will need to be revised as regular common encryption schemes will surely become deprecated. Fortunately, a movement has already started with the generation of “quantum-resistant” algorithms that cannot be broken using quantum computers.

REFERENCES

[1] H. Hacigümüs, B. Iyer, Chen Li, et al. “Executing SQL over Encrypted Data in the Database-Service-Provider Model”. In: June 2002, pp. 216–227. doi: 10.1145/564691.564717.

[2] M. Naveed, S. Kamara, C. V. Wright. “Inference Attacks on Property-Preserving Encrypted Databases”. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015).

[3] R. Curtmola, J. Garay, S. Kamara, et al. “Searchable symmetric encryption: Improved definitions and efficient constructions”. In: Journal of Computer Security 19 (Jan. 2011), pp. 895–934. doi: 10 . 1145 /1180405.1180417.

[4] S. Kamara, T. Moataz. “SQL on Structurally-Encrypted Databases: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I”. In: Jan. 2018, pp. 149–180. isbn: 978-3-030-03325-5. doi: 10.1007/978-3-030-03326-2_6.

[5] S. Kamara, T. Moataz, S. Zdonik, Z. Zheguang, et al. An Optimal Relational Database Encryption Scheme. Cryptology ePrint Archive, Report 2020/274. https://eprint.iacr.org/2020/274. 2020.

[6] S. Benson. The Secret Security Wiki — Encryption And Cryptography. Available: https://doubleoctopus.com/security-wiki/encryption-and-cryptography/.

[7] T. Ristenpart. There’s No One Perfect Method For Encryption In The Cloud. Jan. 2017. Available: https://www.darkreading.com/cloud/theres-no-one-perfect-method-for-encryption-in-the-cloud/a/d-id/1327972.

[8] A. Singhal “Symmetric Key Cryptography | Cryptography Techniques” — Gate Vidyalay. Available: https://www.gatevidyalay.com/cryptography-symmetric-key-cryptography/.

[9] A. Singhal “Public Key Cryptography | RSA Algorithm Example” — Gate Vidyalay. Availiable:

[10] J. Oh, D. Yang, K. Chon. “A Selective Encryption Algorithm Based on AES for Medical Information”. In: Healthcare informatics research 16 (Mar. 2010), pp. 22–9. doi: 10.4258/hir.2010.16.1.22.

[11] What is Tokenization vs Encryption - Benefits Uses Cases Explained — McAfee. Available: https://www.mcafee.com/enterprise/en-us/security-awareness/cloud/tokenization-vs-encryption.html.

[12] Deterministic encryption — CryptoWiki. Available: https://cryptography.fandom.com/wiki/Deterministic_encryption.

[13] E. Damiani, S. De Capitani di Vimercati, S. Foresti, et al. “Selective Data Encryption in Outsourced Dynamic Environments”. In: Electronic Notes in Theoretical Computer Science 168 (2007). Proceedings of the Second International Workshop on Views on Designing Complex Architectures (VODCA 2006), pp. 127–142. issn: 1571-0661. doi: https://doi.org/10.1016/j.entcs.2006.11.003. Available: https://www.sciencedirect.com/science/article/pii/S1571066107000321.

[14] R. Popa, C. Redfield, N. Zeldovich, et al. “CryptDB: Protecting confidentiality with encrypted query processing”. In: Jan. 2011, pp. 85–100. doi: 10.1145/2043556.2043566.

[15] A. Boldyreva, N. Chenette, Y. Lee, et al. “Order-Preserving Symmetric Encryption”. In: vol. 5479. Apr. 2009, pp. 224–241. isbn: 978-3-642-01000-2. doi: 10.1007/978-3-642-01001-9_13.

[16] M. Salam, W. Yau, J. Chin, et al. “Implementation of searchable symmetric encryption for privacy-preserving keyword search on cloud storage”. In: Human-centric Computing and Information Sciences 5 (July 2015), p. 19. doi: 10.1186/s13673-015-0039-9.

[17] R. Ada Popa, N. Zeldovich, and H. Balakrishnan. “Guidelines for Using the CryptDB System Securely”. In: IACR Cryptology ePrint Archive 2015 (2015), p. 979. Available: https://eprint.iacr.org/2015/979.

[18] M. Bellare, A. Boldyreva, A. O’Neill. “Determinist and Efficiently Searchable Encryption”. In: vol. 2006. Jan. 2006, p. 18 isbn: 978-3-540-74142-8. doi: 10.1007/978-3-540-74143-5_30.

[19] A. Boldyreva, S. Fehr, A. O’Neill. “On Notions of Security for Deterministic Encryption, and Efficient Constructions without Random Oracles”. In: Advances in Cryptology – CRYPTO 2008. Ed. by David Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 335–359. isbn: 978-3-540-85174-5.

[20] M. Bellare, M. Fischlin, A. O’Neill, T. Ristenpart et al. “Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles”. In: vol. 2008. Aug. 2008, pp. 360–378. isbn: 978-3-540-85173-8. doi: 10.1007/978-3-540-85174-5_20.

[21] M. Chase, S. Kamara. “Structured Encryption and Controlled Disclosure”. In: Dec. 2010, pp. 577–594. isbn: 978-3-642-17372-1. doi: 10.1007/978-3-642-17373-8_33.

[22] C. Gentry. “A Fully Homomorphic Encryption Scheme”. AAI3382729. PhD thesis. Stanford, CA, USA, 2009. isbn: 9781109444506.

[23] S. Kamara, M. Raykova. “Parallel Homomorphic Encryption”. In: Financial Cryptography and Data Security. Ed. by Andrew A. Adams, Michael Brenner, and Matthew Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 213–225. isbn: 978-3-642-41320-9.

[24] F. Armknecht, C. Boyd, C. Carr, et al. A Guide to Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2015/1192. https://eprint.iacr.org/2015/1192. 2015.

[25] R. Adams, J. Fournier, “Sobolev Spaces (Second ed.)”, Academic Press, ISBN 978-0-12-044143-3 2003.
[26] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In EUROCRYPT, pages 563-594, 2015
[27] Z. Zhao, S. Kamara, T. Moataz, S. Zdonik “Encrypted Databases: From Theory to Systems” 11th Annual Conference on Innovative Data Systems Research (CIDR ‘21) Chaminade, USA, 2021.
[28] CryptDB Team – CryptDB implementation Available: https://github.com/CryptDB/cryptdb
[29] Y. Shao - Practical CryptDB source code Available: https://github.com/yiwenshao/Practical-Cryptdb
[30] Agribu - Practical CryptDB Docker source code Available: https://github.com/agribu/Practical-Cryptdb_Docker
[31] The KafeDB Team. KafeDB implementation Available: https://github.com/zheguang/encrypted-spark, 2020.
[32] Google BigQuery Available: https://cloud.google.com/files/BigQueryTechnicalWP.pdf
[33] Always Encrypted Available Database Server Engine: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-ver15
[34] E. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377–387,1970.
[35] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In (STOC’77), 1977.
[36] CryptDB Team – Website Available: https://css.csail.mit.edu/cryptdb/
[37] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In Proceedings of the 21st IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.