Combining Shamir's Secret Sharing Scheme and Symmetric Key Encryption To Achieve Data Privacy in Databases

by

Abdel Maguid Tawakol

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2016

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The Internet has become an essential tool for communication and information access, and with growing demand, new challenges and usage continue to surface. A complimentary tool that came to exist in recent years is Cloud Computing - an economical solution that serves as an alternative to owning and running computing facilities. While cloud computing has many advantages, there are a number of issues that hamper the adoption of cloud computing. Some of the major concerns, can be classified into one of the following groups: traditional security, availability, and third-party data control. The first set of concerns, revolve around security threats that can expose clients private data. The second set of concerns, revolve around the compromise of the operation of the applications in the cloud. Finally, the last set of concerns involve the legal implications of data and applications being held by a third party. Different solutions exist to deal with traditional security, availability, and third-party data control, separately, but one way to handle traditional security, and third-party data control, is through data encryption. The client has to take responsibility for ensuring that the data is setup in such a way, that even if the cloud service provider is compromised, or has a malicious intent, it is not able to get anything from the customers data. Of course, encrypting the data introduces limitations, with varying tradeoffs for different systems. In this work, we use Shamirs Secret Sharing Scheme and a symmetric key cryptographic system (AES) to encrypt data at a field level, such that it can be stored in the cloud without compromising data privacy. Using Shamirs Secret Sharing Scheme to encrypt numeric field values, gives us the ability to perform efficient addition, subtraction, and multiplication on the encrypted numeric field values. We explore two different ways of using Shamir Secret Sharing Scheme and AES, and discuss the advantages and disadvantages of each. We then propose, and complete, a software implementation for the proposed system. The implementation is used in order to compare execution time, memory usage, and bandwidth usage, to the plaintext and MySQL encrypted versions of the database. Analyzing the benchmarks, we can see how the performance varies for different query types when run on tables with different number of records and field types giving the reader an idea about the cost and tradeoffs of the system.

Acknowledgements

I would like to thank professor Gordon Agnew for his insight on the thesis topic and help completing my work. I would also like to thank professor Mahesh Tripunitara on all his help completing the security analysis section. I would also like to thank the committee members for their time reviewing my work.

Dedication

This thesis is dedicated to my parents Hashem and Suheir, and my lovely fiancè Hana

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The Internet has become an essential tool for communication and information access, and with growing demand, new challenges and usage continue to surface. A complimentary tool that came to exist in recent years is Cloud Computing - an economical solution that serves as an alternative to owning and running computing facilities. While cloud computing has many advantages, there are a number of issues that hamper the adoption of cloud computing. Some of the major concerns, can be classified into one of the following groups: traditional security, availability, and third-party data control[32]. The first set of concerns, revolve around security threats that can expose clients' private data. The second set of concerns, revolve around the compromise of the operation of the applications in the cloud. Finally, the last set of concerns involve the legal implications of data and applications being held by a third party[32].

Different solutions exist to deal with traditional security, availability, and third-party data control, separately, but one way to handle traditional security, and third-party data control, is through data encryption. The client has to take responsibility for ensuring that the data is setup in such a way, that even if the cloud service provider (CSP) is compromised, or has a malicious intent, it is not able to get anything from the customer's data. Secondly, mechanisms exist to ensure proper auditing of the data, in order to detect any malicious or erroneous data alterations. Of course, encrypting the data has its limitations, and it is the focus of this work to determine a cost effective implementation that would allow a company/customer to leverage the cloud for their computing needs.

The goal of this work is to find a way to process encrypted data stored in a database, to allow for the utilization of cloud computing services, without compromising the data's privacy. Currently, there is no efficient way to perform computations on encrypted data, which means that in many implementations, the data has to be decrypted first, before any computations can be performed on it, and then re-encrypted. This gave way for homomorphic encryption schemes, which are still a very active area of research, with no single solution for all applications. Existing solutions lack efficiency, and/or are highly complex. One of the prominent work on homomorphic encryption by Gentry, Halevi, and Smart[46], is done under the ring-LWE (Learning with Error [66]) assumption, and has a best case scenario of polylogarithmic overhead. We propose the use of Shamir's Secret Sharing Scheme, in order to encrypt the data values that will undergo any computations,

leveraging the homomorphic properties of the scheme, allowing us to take advantage of storing and processing the data in the cloud, without exposing the clients' private data. Shamir's Secret Sharing Scheme was created to divide a secret in such a way, that it would require the collusion of multiple parties to recover the secret. It can also be used to protect a secret in a hierarchical fashion, where certain individuals carry *higher weight* than others in the secret recovery process. For example, in order to access client data, it requires the president of the company, and two vice presidents, or a vice president and 4 managers, etc... By combining Shamir's Secret Sharing Scheme and a traditional symmetric key cryptographic system (such as AES), we hope to pave the way for a database system that can be stored in the cloud, and accessed from anywhere through the Internet.

In addition to being able to perform computations on encrypted data in the cloud, data redundancy is crucial for many database applications where the stored information is of great value. For example, the database system containing patient's medical records. In many cases, the this data is irreplaceable, and so it is not enough to protect the data from unauthorized access, but there is also a need for backup and redundancy to insure that the data remains safe and accessible at all times.

1.1 Thesis Statement

There exists a database system, that has the following properties:

- Through field level encryption, maintain the confidentiality of the data stored in the database
- Support queries, such as select and criteria based queries, on encrypted data
- Support server-side calculations on data values, without the need for full decryption, by *encrypting* the values using Shamir's Secret Sharing Scheme (details in section 3.2.3)
- Provide database data redundancy through data replication, and threshold scheme data division
- Controlled access to the database through the Internet

In the context of this work, data confidentiality is defined as: making the plaintext version of the data stored in the database available to authorized users only. An authorized

user, is an individual who has access to the private keys used to decrypt the ciphertext stored in the database.

In our work, we propose a complete system that involves three components (details in chapter 4.4): a client-side application, a server-side application, and an encrypted database hosted in the cloud. The implementation leverages AES to encrypt string values, while Shamir's Secret Sharing scheme is used to encrypt numerical values. With the help of an application running on the server side, a database hosted in the cloud, we propose an initial design of how the system would be setup and operate, while maintaining data confidentiality. Shamir's Secret Sharing Scheme, is an example of a privacy homomorphism [67]. The problem is, that it is weak cryptographically because a "chosen plain-text attack" can break it [67]. Therefore, it is not possible to use it by itself, or at least not in its original form.

The security of the system, like any other system with a cryptographic implementation, is potentially vulnerable to implementation errors that could weaken its security - something outside the scope of this thesis. We assume that the implementation used for AES, and implementation of Shamir's Secret Sharing Scheme is complete and reliable.

In this work, we will explore different options for combining AES encryption with Shamir's Secret Sharing scheme as well as the advantages and disadvantages of the various usages. The next step will look at a proposed software implementation for the system, and how the system will be compared to MySQL encrypted database and plaintext database. The next chapter will present a cost analysis in terms of memory usage and execution time for the implementation of the proposed system, on two platforms: laptop and iPhone. We will also discuss potential improvements to the system.

Chapter 2
Background

This section covers a lot of the topics relevant to this work. Topics include: databases, encryption, big data, and cloud computing. Due to the nature of the work, the main focus is on databases and homomorphic encryption.

2.1 Big Data

The targeted application for this work, are databases, and thus, the size of the database, how it is stored, and how it will be used, are all factors that affect the feasibility of the proposal. The first step, would be to define the parameters that affect the performance of the database, as well as its size. The definition of a large database, is changing over time as Hardware and Software continue to evolve and advance [6]. For the purposes of this work, the focus will be on small, and medium sized databases. A table summarizing and comparing small and medium sized databases is shown below:

Small Sized Database	Medium Sized Database
Fits in Memory	Fits in Single Server
$< 10^5$ Records	10^5 - 10^7 Records
$<$ 10 GB of Data	10 - 40 GB of Data
No partitions	Minimal Partitions

Factors that affect performance, include [6]: data volume, throughput, software, and hardware. Data volume is the amount of data as defined by the number of records, tables, terabytes/petabytes, etc...[6]. Throughput defines the database usage, in terms of the number of concurrent users accessing the database, and the number of transactions being carried out by each user. The software includes both the database management system, and the implementation of the database - both of which can affect how quickly data can be retrieved. Finally, the hardware available to host the database and process the queries, will also affect the performance of query processing and data retrieval.

Based on the aforementioned factors, we will define the database to have the following characteristics:

– The DBMS technology used in our model will be MySQL, since it is an established and open-sourced system. Since MySQL is widely used by many large institutions [7], it is assumed that it is an efficient and reliable DBMS that can be used as a testbed for our proposal.

– The database will contain 10^5 - 10^7 records, spread across multiple tables. At least two scenarios with two setups will be tested: records divided equally across all tables, and tables unevenly divided across multiple tables (some tables containing the bulk of the data, and the remaining records in other tables). The storage space will also be < 40 GB.

– The hardware that will be used to store the DB, and execute the queries, meets the minimum hardware requirements as defined by software developers of MySQL [3].

– Since the values are encrypted, the indexing of values encrypted using AES will not be indexed, and the indexing doesn't make sense for the values encrypted using Shamir's Secret Sharing Scheme since the data ordering is removed as well.

– Sharding (breaking a slow large database into a lot of quick little databases), database virtualization, and database optimizations are beyond the scope of this work.

2.2 Symmetric-key Cryptography

Symmetric key algorithms are cryptographic algorithms that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext [64]. Some examples of symmetric key cryptography systems include stream ciphers and block ciphers such as AES. One of the weakness of symmetric key cryptography system is that you can't achieve non-repudiation - the reason being that the secret key is shared between the sender and receiver. That is, if the secret key used for encryption and decryption is the same, it would be impossible to be able to distinguish who used the key to generate the ciphertext - a drawback that doesn't exist in the public key system. Another important drawback is that the secret key used for encryption/decryption has to be exchanged with all authorized parties securely. Finally, with this system, the number of keys that would have to be generated and maintained is high since you have to generate a new key for each pair of individuals communicating[64]. While symmetric key cryptographic systems might have a few drawbacks, it does have an important advantage over public key cryptography system: speed - by a factor of approximately one hundred to one thousand times faster [64].

2.3 Asymmetric-key Cryptography

Public key cryptography, also known as asymmetric key cryptography, refers to a cryptographic algorithm which requires two separate keys: A secret (or private) key for decrypting ciphertext or generating digital signatures, and a public key used for encrypting plaintext or verifying digital signatures[64]. Due to the existence of different keys for encryption and decryption, public key cryptographic systems can be used for applications such as digital signatures and key establishment, and for classical data encryption[64]. Two popular examples of public key system, which will be discussed in the next subsections, are RSA and ElGamal.

Public key cryptographic systems have a number of uses:

- Key Establishment: Protocols that allow for the secure exchange of secrets keys over an insecure channel
- Non-repudiation: Digital signature schemes provide non-repudiation and message integrity through the use of public key cryptographic systems
- Identification: Combining digital signatures and challenge-and-response protocols provide entity identification
- Encryption: Classical data communication encryption algorithms using schemes such as RSA and ElGamal

One of the drawbacks of public key cryptographic systems is ensuring the authenticity of public keys. The current solution to this drawback is the use of certificates. Certificates are issued by what is referred to as a "certificate authority" (CA) - a trusted entity that binds users to a public key after verifying their identity. These certificates can then be placed in the public domain, allowing anyone one to use them for encrypting messages for a particular sender. Of course the compromise of the CA, would allow for the impersonation of entities. The second potential problem with CAs is that they have to maintain high availability in order to be able to perform verification requests at any time. Another drawback of public key cryptographic systems is the intensity of the operations involved with the different schemes in this class. In order to achieve a "secure" system, the operands for the different schemes must have a minimum number of bits to achieve a certain security level, making the computations being performed in each scheme extremely intensive. For example, for RSA, to achieve a 256 bit security level, the public key has to be 15,360 bits -

a very large number! It is estimated the computational complexity of the algorithms grows roughly cube of the bit length [64].

The majority of practical public key cryptographic systems rely on one of the following computational problems:

- Integer Factorization Schemes: These are schemes that rely on the difficulty of factoring large integers (e.g. RSA)
- Discrete Logarithms Schemes: Schemes that rely on the difficulty of discrete logarithm problem in finite fields (e.g. Diffie Hellman Key Exchange)
- Elliptic Curve Schemes: Schemes that rely on the difficulty of determining the discrete logarithm of an elliptic curve element with respect to a publicly known base point (e.g. Elliptic Curve DiffieHellman (ECDH) key agreement scheme)

Public key algorithms are based on mathematical problems which currently have no efficient solution that are inherent in certain integer factorization, discrete logarithm, and elliptic curve relationships. It is computationally easy for a user to generate their own public and private key-pair for use with encryption and decryption. The strength lies in the computational infeasibility of determining a public key from a properly generated private key. Thus the public key may be published without compromising security, whereas the private key must not be revealed to anyone not authorized to read messages or generate digital signatures. Public key algorithms, unlike symmetric key algorithms, do not require a secure initial exchange of one (or more) secret keys between the parties.

2.4 Homomorphic Encryption

Homomorphic encryption is a type of encryption that allows for mathematical operations to be applied to the encrypted values - without having to decrypt it first. This means that computations can be performed on encrypted values, yielding an encrypted result - without having to decrypt intermediate results. Formally, homomorphic encryption for plaintext M, and operation $\bigodot$ (where $\bigodot$ is some operator), is defined as [44]:

$$\begin{aligned} &\forall m_1, m1_2 \in M \\ &E_k(m_1) \bigodot E_k(m_2) = E_k(m_1 \bigodot m_2) \end{aligned} \tag{2.1}$$

A scheme is called *additively homomorphic* if the operator being used is addition, and *multiplicatively homomorphic*, if the operator is multiplication [44]. RSA and ElGamal encryption schemes are examples of multiplicatively homomorphic encryption schemes[44, 65]:

$$\begin{aligned} &\forall m_1, m1_2 \in M \\ &E_{k_e}(m_1 * m_2) = (m_1 * m_2)^e mod N \\ &\qquad\qquad = ({m_1}^e mod N)({m_2}^e mod N) \\ &\qquad\qquad = E_{k_e}(m_1) * E_{k_e}(m_2) \end{aligned} \tag{2.2}$$

The problem with RSA is two fold: it is not protected against chosen plaintext attacks, and it is not additively homomorphic (i.e., doesn't support addition) [44]. On the other hand, a variant of ElGamal encryption scheme has been proposed that, in addition to being multiplicatively homomorphic, supports addition as well [44].

Some of the other existing homomorphic encryption schemes include: Goldwasser-Micali, Benaloh, Naccache-Stern, Paillier, and Paillier variants. Goldwasser-Micali is the first probabilistic public-key, asymmetric key encryption algorithm, which is provably secure, but is inefficient because each bit in the plaintext leads to a large data expansion in the ciphertext [44]. Benaloh is a generalization of Goldwasser-Micali, which improves on the efficiency of Goldwasser-Micali by encrypting data blocks rather than individual bits, but the gain is limited because the decryption process is still heavily influenced by the size of the plaintext. Naccache-Stern, builds on Benaloh's scheme, and attempts to improve the decryption cost.

The next set of homomorphic encryption schemes are based on Paillier encryption scheme: a probabilistic asymmetric and additively homomorphic scheme. Paillier's scheme decreases the data expansion, and through the use of Chinese Remainder Theorem, provides a more efficient decryption than the schemes previously presented, but it does not meet the adaptive chosen-ciphertext attack requirements[44]. Through certain algebraic properties, Cramer and Shoup proposed a general approach to resolving the weakness in Paillier encryption scheme[35]. Damgard and Jurik proposed another variant of Paillier, which lowers the expansion of the ciphertext, but ends up being more costly overall[44].

All of the homomorphic schemes previously mentioned, are based on asymmetric schemes, which are much slower than symmetric schemes.

2.5 Algebraic Homomorphic Systems Requirements

Rivest, Adleman, and Dertouzos, came up with a list of requirements that an algebraic system with homomorphic properties should have [67]:

- The encoding and decoding functions should be easy to compute
- The operations and predicates should be efficiently computable
- The encoded version of the data should not require more space to represent than the amount of space required for the original data
- Knowledge of many encoded values, should not reveal anything about the decoding function or the key (Ciphertext only attack)
- Knowledge of plaintext and its corresponding encoded value, for several values, does not reveal anything about the decoding function or the key (Chosen plaintext attack)
- The operations and predicates in the algebraic system should not be sufficient to yield an efficient computation of the decoding function

The proposed encryption scheme, will attempt to meet as many of the requirements as possible. Currently, the scheme meets the first two requirements but does not meet the third requirement because the secret is broken to three pieces of equal size. A thorough analysis is still required to determine whether the last three requirements are met or not.

2.6 Shamir's Secret Sharing

Shamir's Secret Sharing Scheme allows the sharing of a secret by dividing it into pieces, and giving each participant their own unique part, where some, or all, the parts are needed in order to reconstruct the secret [70]. Requiring all the participants to combine together the secret might be impractical, and therefore sometimes the threshold scheme is used where any k of the parts are sufficient to reconstruct the original secret[70]. The mathematical definition of Shamir's Secret Sharing Scheme:

- Knowledge of any k or more D_i pieces makes D easily computable

- Knowledge of any *k-1* or fewer D_i pieces leaves D completely undetermined (all possible values are equally likely)

The process of reconstructing the secret is performed using polynomial interpolation, where given k points in the 2-dimensional plane (x,y), ..., (x_k, y_k, with distinct x_i, there is one, and only one polynomial $q(x_i) = y_i$ for all i [70]. There exists efficient ways to perform the reconstruction process.

2.6.1 Homomorphic Properties of Shamir's Secret Sharing Scheme

First, we will show how Shamir's Secret Sharing Scheme addition homomorphic properties hold. In this case, the polynomial used to break up the secret is a degree two polynomial. Assume that there are two secret values, C_1 and C_2. Therefore, we will setup the two polynomials to generate the secret pieces as follows:

$$\begin{aligned} a_1x^2 + b_1x + C_1 &= y_1 \\ a_2x^2 + b_2x + C_2 &= y_2 \end{aligned} \tag{2.3}$$

Note that constant values (a and b) for the polynomials that will hold the two secrets, do not have to be the same. That is true, regardless of the degree of the polynomial to be used for breaking up the secret. The reason being that the secret occurs at the y-intercept, and the values of the constants cancel out at x = 0 - regardless of their original values. For the re-construction of the secret, we use Lagrange's Interpolation, using the following formulas:

$$\begin{aligned} L(x) &= \sum_{j=0}^{k} y_j l_j(x) \\ l_j(x) &= \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} \end{aligned} \tag{2.4}$$

The next step is to show the points that would be generated for each polynomial, plus the polynomial which would be reconstructed from the points that are a sum of the secrets C_1 and C_2:

Polynomial y_n	Point 0	Point 1	Point 2
$f_1(x) = ax^2 + bx + C_1$	(x_0,y_0)	(x_1,y_1)	(x_2,y_2)
$f_2(x) = ax^2 + bx + C_2$	(x_0,y_3)	(x_1,y_4)	(x_2,y_5)
$f_3(x) = 2ax^2 + 2bx + (C_1 + C_2)$	$(x_0,(y_0 + y_3))$	$(x_1,(y_1 + y_4))$	$(x_2,(y_2 + y_5))$

Table 2.1: This table shows the points generated for each secret value.

In this case, we are using the same value for constants a and b, but this not necessary. For reconstructing the polynomials $f_1(x)$, $f_2(x)$, or $f_3(x)$, the value $l_j(x)$ will be the same for all of them because it is strictly dependent on the values of x - which are equal for all functions. The only difference in the calculations of reconstructing the values, is the y values used. Now we want to prove that if we add the y values for a particular x, and try to reconstruct the secret, we will get the sum of the two original secrets. When using Lagrange's Interpolation to reconstruct a degree two polynomial, we have to first calculate the values l_0, l_1, and l_2. The next step, is to apply the following equation for reconstructing polynomials $f_1(x)$ and $f_2(x)$:

$$\begin{aligned} f_1(x) &= \sum_{j=0}^{2} y_j * l_j(x) \\ f_1(x) &= y_0 * l_0 + y_1 * l_1 + y_2 * l_2 \\ f_2(x) &= \sum_{j=0,k=3}^{j=2,k=5} y_k * l_j(x) \\ f_2(x) &= y_3 * l_0 + y_4 * l_1 + y_5 * l_2 \end{aligned} \tag{2.5}$$

Now if we apply the equations to reconstruct the function $f_3(x)$, we get the following:

$$
\begin{aligned}
f_3(x) &= \sum_{j=0}^{2} y_j * l_j(x) \\
f_3(x) &= (y_0 + y_3) * l_0 + (y_1 + y_4) * l_1 \\
&\quad + (y_2 + y_5) * l_2 \\
f_3(x) &= (y_0 * l_0 + y_1 * l_1 + y_2 * l_2) \\
&\quad + (y_3 * l_0 + y_4 * l_1 + y_5 * l_2) \\
f_3(x) &= f_1(x) + f_2(x) \\
f_3(x) &= 2ax^2 + 2bx + (C_1 + C_2)
\end{aligned}
\tag{2.6}
$$

Even though the resulting polynomial (from the summation of $f_1(x)$ and $f_2(x)$)), has a different shape, the y-intercept occurs at the sum C_1+C_2. Using a very similar proof to the one shown in equations 2.5 and 2.6, one can show that the same holds for the multiplication property. The only difference for the multiplication, is that it will require five points to re-construct the function because the result of multiplying two polynomials of degree two is a polynomial of degree four.

Shamir's Secret Sharing Scheme, meets all of the requirements propoposed by Riverst, Adleman, and Dertouzous except for one: the encoded version of the data should not require more space to represent than the amount of space required for the original data. Since the data is being split into three parts, each at least the size of the original data, this requirement is not.

2.7 Database Encryption and Security

In general, there are three distinct, but overlapping methods, to achieve data security: physical security, operating system security, and data encryption. To achieve data confidentiality, the data needs to be encrypted, since access control mechanisms can only protect the database if it is being accessed using the traditional mechanisms[71]. It should be noted, that even encryption, by itself, is unlikely to be sufficient for the protection of a database, but it does go a long way to solve many of the problems [36]. There are many challenges to processing encrypted data in general, and databases are no exception. This section will look into details at the work that has been to protect data through encryption

and non-encryption scheme. Each proposed method will be analyzed in detail - highlighting the pros and cons of each. At the end of this section, analysis on some of the early encryption methods explored for this work will be presented.

2.7.1 Database Security Without The Use of Encryption

In this section, we will discuss some of the proposed methodologies for protecting data in a database, without the use of encryption. One of the proposed schemes [17], achieves data privacy by partitioning the database and storing it on multiple servers that can't communicate with each other. The idea is the data on any of the partitions, by itself, can't violate the privacy of the data [17]. The queries are carried out by performing sub-queries on the different database parts, and then putting together the result at the client side [17]. The example presented involves storing a credit card number field securely. This is achieved by generating a random number r, and XOR-ing that value with the credit card number. One database will hold the random numbers, and one database will hold the result of XOR-ing the random number with the credit card number. To recover the credit card number, the client queries both databases, and XORs the the values returned from the databases, to get the credit card number. The advantages of this scheme include: no encrypted connections are required to the various servers hosting the database (because the data at each provider on its on is meaningless, and eavesdropping on that connection will not compromise the privacy of the data), the operation for recovering the data is cheap relative to decryption, and the data can be stored in plaintext form (preserving the operation of the database and the types of queries supported). One of the disadvantages of this scheme is the process of decomposition/partitioning of the database, which is not straightforward, and requires extensive analysis of the attributes of the database to determine how the client defined security and privacy constraints will be met. Another weakness is that the architecture does not support selection on encoded attributes, nor does it support the $GROUPBY$ clause - two key functionalities in databases. Finally, the authors have not performed a real-world case study on their architecture, which could potentially reveal more drawbacks of their architecture.

2.7.2 Database Encryption Advantages and Disadvantages

As with any application where encryption is used, in the context of databases it provides a layer of security to preserve data confidentiality even if the physical data is lost or compromised[36]. There are a lot of concerns involved with encrypting, and executing

queries, on encrypted databases. The first concern is, how will the database be encrypted? Should the encryption be on the record level? Or on the field level? Or should the encryption occur on the database file as a whole? These are the first set of questions that are encountered when database encryption is going to be performed - encryption granularity. In order to answer these question, an individual must examine each method, in light of the application its being used in, to determine the best fit. The first challenge introduced with most database encryptions is the ciphertext searching/substitution vulnerability. This issue occurs because of the high frequency of identical plaintext values being encrypted under the same key. While this issue occurs in any encryption application, it is especially problematic in databases because the possibility of repeated plaintext is high.

The next common challenge introduced with most database encryption is the fact that the encryption is not order preserving. This is problematic for the operation of the database, because it relies on indexes to speed up query execution, which in turn requires ordered data to be pre-computed[72]. While encryption functions preserve equality, allowing for hashing to be used for indexing, frequency information is revealed[72].

Given all these challenges and concerns, it is clear that sacrifices, or compromises, have to be made in order to achieve a particular level of security. The type of compromises will be determined by the level of security defined by the user, and will vary from one application to the next.

2.7.3 Database Encryption Techniques

There are different ways to achieve database encryption: encrypting the database as a whole, encrypting records (under a single key), encrypting records (use separate field keys), and encrypting fields (under single or multiple keys). We will explore each encryption method in the following subsections.

Complete Database Encryption

For this encryption method, a single key is used for encrypting the database file, or for encrypting the physical drive that is hosting the database[71]. While this method maintains the confidentiality of the data, it does not support discretionary access control[71]. This method also does not support any database functionality (queries) nor does it support any mathematical computations (addition, multiplication, average, etc...), without the decryption of the database first. Therefore, this type of encryption might be ideal only for archiving databases, and will not be considered or analyzed any further in this work.

Record Based Encryption

For this encryption method, it can be performed using single key for all the records, or multiple keys - one for each record. For the case of a single key use, each record is encrypted such that the values for the different attributes of that record are combined into one large encrypted tuple. This method of encryption is used as a database archival method - requiring large amounts of decryption for any processing[72]. This of course is a major drawback, making it an undesirable option for our application.

One method that involves record based encryption, involves indexing the encrypted attributes/fields that are involved in search and join predicates [48]. The idea is, the encrypted version of the table will contain an encrypted version of the record, plus an index field for every attribute/field involved in a search or join predicate. The mapping function used for the indexing process is order preserving, and is used with a partitioning function to build the encrypted version of the database. With the help of a query translator and meta data about the attributes in the database stored on the client side, a server-side representation of the query is created and sent to the server to be executed. The encrypted tuple result is then decrypted and returned to the user. Finally, an algebraic framework for query splitting is proposed, such that the cost of the query execution (comprised of I/O and CPU cost of evaluating the query at the server, the network transmission cost, and the I/O and CPU cost at the client) is minimized. The advantage of the proposed system is that the data is indeed encrypted, and can be stored at an untrusted site, allowing for the majority of the query execution to occur at the untrusted server. However, there are a few disadvantages. The first disadvantage is that the *binning* of values reveals data distribution information [53]. The second disadvantage is the complexity involved with the implementation, which requires partition functions, mapping functions, identification functions, as well as the maintenance of the meta data to allow for the execution of queries. Finally, it is not clear what the overhead of this setup is, compared to a database system that contains an unencrypted version of the same database under test. The experiments presented show the change in execution time as the number of buckets used for indexing is increased. But the question in this case is, how does this execution time compare to running the same query on an unencrypted version of the same database? This is something that was not discussed in this work.

The second type of record based encryption is performed with separate field keys. A very specialized cryptographic system has been suggested, that is based on the Chinese Remainder Theorem[37, 36]. The advantages of the proposed system is that it can detect any subtle modifications to the ciphertext - whether they have been done maliciously, or occurred due system errors[36]. This is achieved because the presented system is basically

a block cipher over an entire record, with block ciphers error propagation property. The second advantage of this method is that prevents the substitution of field values, since the record is encrypted as a whole. Another advantage, is that the system is not susceptible to searching/substitution vulnerability since the possibility of having repeated ciphertext is low - given that the possibility of two records being identical is fairly low. Finally, a very important advantage is that it allows for partial block decipherment (unlike block cipher cryptographic systems), which enables easy access to certain fields for a set of records, without the need for decrypting the whole record[36]. The disadvantages of this system include expensive write and update operations, because if the value of one of the fields is changed, the entire record has be to re-encrypted[36]. The second major disadvantage is that an update to any record, requires knowledge of all the field values for that record. Therefore, it is not enough to have the encryption/decryption keys, but also the other values in that record have to be known by the user performing the update, or have to be retrieved prior to the update. This is a major disadvantage because it is very plausible that the person performing a record update, does not have knowledge of the values of the other fields - whether it is because he does not have access to all the fields in those record (does not have security access), or because they simple don't have the values memorized. Another disadvantage is, it is not possible to project out fields without performing a computation over complete records, which requires knowing the field keys[38].

The last example of a record level encryption to be explored, is the one by [53]. In this work, the authors propose adding a different counter-based CTR for each record. The counter generates a random number, which is encrypted, and XOR-ed with an attribute. The next attribute in the record is encrypted by XOR-ing the plaintext with the encrypted version of the next sequence in the counter. In other words, the following scheme is operations are performed to perform the encryption:

CTR	$Attribute1$	$Attribute2$	$Attribute3$
Counter C	$\oplus E(C)$	$\oplus E(C+1)$	$\oplus E(C+2)$

Table 2.2: Encryption scheme proposed by [53].

The advantages of the proposed scheme, is that it meets a pre-defined set of security requirements (as discussed in their work), while still supporting select, project, and join queries. The disadvantage of this scheme is that in its current form, it only supports read-only queries. Although they claim that it can be extended to support write queries, they did not discuss how this would be done. The second disadvantage is that it requires tamper-proof hardware module to perform secure execution. Therefore, this scheme will require certain hardware setup wherever it is deployed - adding to the cost and complexity

of deployment. Finally, in order to meet the defined security requirements, dummy tuples are sometimes sent to the client - increasing both the amount of data communicated, and the amount of data the client has to filter through to get the relevant data. Finally, the scheme does not support any mathematical operations on the encrypted fields - the field values have to be decrypted before they can be involved in any computations.

Field Based Encryption

For this encryption method, it can be performed using single key or multiple keys (one for each field). For the single key field encryption method, it can be one key that is used to encrypt every field value in a database. The single key field encryption method is vulnerable to ciphertext searching/substitution, if there are a lot of repeated plaintext values[76]. The second type of field based encryption is using separate keys for each field of each record. In order to avoid having to store a large number of keys, the keys can be generated as a one-way function of a record id, field id, and a single secret key[76, 38]. The multiple key method mitigates the weaknesses of ciphertext searching/substitution, and has a lot of advantages, which will be discussed in more details.

Advantages of field based encryption include:

- Support projection, and selection, in encrypted form [38]
- Allows for the decryption of select fields in a record independently, without having to decrypt all the fields in the record [38]
- Compatibility with legacy applications [71]
- If the encryption keys are based on a function, then the amount of info that needs to be stored for encryption/decryption is limited

Disadvantages of field based encryption include:

- To retrieve records based on a field criteria, the field of every record has to be checked. In other words, requires full table scan [38]
- Field level encryption causes expansion of short fields - for example for using DES it would expand short fields to 64 bits [38]

One work suggested for field level encryption in databases, is through the use of multiple representations of data values for the same field (attribute) - formally known as homophonic cipher [76]. In other words, multiple representations of a ciphertext are used, for the same plaintext value, in order to prevent frequency analysis. The author of this work suggests applying the homophonic cipher at the character level, where the assumption is that each character has a 6-bit representation and that an 8-bit homophonic encoding is used, which will require a $r/3$ dummy characters to be certain that a homophonic encoding exists with a flat distribution[76]. In order for this scheme to allow for relational joins, exact flat distribution will not be possible, and information to dynamic attacks is leaked[76]. Another major weakness for the proposed system is that when a query for finding a record (or group of records), based on the value of some field is being executed, the entity initiating the query has to generate the different homophonic representations for each character - at most 256 8-bit numbers. For our application, this has a two fold drawback: the amount of processing required to generate the different combination for a plaintext value, and the communication cost of transmitting those values to the CSP. If the query is suppose to match multiple field values, then the effect is compounded further - making it computationally heavy. This scheme supports average calculations, but extra characters have to be used to identify dummy records, and extra computations have to be performed to remove the dummy records from the calculation.

The work by Denning suggests a different approach to field encryption: using different keys to encrypt each field in order to mitigate the weaknesses of ciphertext searching/substitution. The work focuses on different methods of key generation, such that the probability of getting repeated ciphertext is low - an extension to the work by Flynn and Campasano which proposes the use of different keys for each record [43]. Also to reduce the issue with maintaining a large number of encryption/decryption keys, the key is generated based on a fixed relation, such that it can be easily generated during decryption. Denning proposes five possibilities for the key generator function, where $\oplus$ is the exclusive-or operator [38]:

1. $\mathrm{K}_{ij} = \mathrm{E}_{K_j}(\mathrm{R}_i)$, where $\mathrm{K}_j = \mathrm{E}_K(\mathrm{F}_j)$
2. $\mathrm{K}_{ij} = \mathrm{E}_{K_i}(\mathrm{F}_j)$, where $\mathrm{K}_i = \mathrm{E}_K(\mathrm{R}_i)$
3. $\mathrm{K}_{ij} = \mathrm{R}_i \oplus \mathrm{K}_j$, where $\mathrm{K}_j = \mathrm{E}_K(\mathrm{F}_j)$
4. $\mathrm{K}_{ij} = \mathrm{K}_i \oplus \mathrm{F}_j$, where $\mathrm{K}_i = \mathrm{E}_K(\mathrm{R}_i)$
5. $\mathrm{K}_{ij} = \mathrm{E}_K\ (\mathrm{R}_i \oplus \mathrm{F}_j)$

Note that R_i is the identifier for record i, F_j is the identifier for field j, and K_{ij} is the key for cell i, j. The disadvantage of the first method is that two encryptions are needed to compute each element key in a record - tripling the effort required to encrypt or decrypt an individual record [38]. The advantage of the first method is, for multiple record accesses to a particular field j, the element keys in field j can be obtained with one additional encryption each, once K_j is computed. The second method is similar to the first method, but the order of the encryption is switched, allowing faster access to all fields within a record, but slower access to multiple records [38]. The third method replaces one of the encryptions with an XOR. The problem with that approach is that if key K_{ij} is compromised, then the field key K_j is compromised as well, and therefore every element key in field j, is easily computed (assuming that record identifiers are known) [38]. The fourth method is similar to the second, where a record key is first generated, but instead of a second encryption an XOR operation is performed [38]. With this method, the time to encrypt or decrypt an entire record is competitive with record based encryption, but multiple record accesses to a single field still requires an encryption (of the record key) to obtain the element keys in the field [38]. The fifth method performs a single encryption and an XOR operation, giving it the advantage of never requiring double encryption to compute an element key [38]. This method, does though, always require encryption, making it slower than method 3 for multiple record accesses to a single field, and slower than method 4 for single record encryption and decryption. Another disadvantage is $K_{ij} = K_{pq}$ will occur whenever $R_i \oplus F_j = R_p \oplus F_q$ [38]. The common problem with the previously mentioned solutions is that if a query was to be executed on values encrypted using this scheme, you would have to supply a value that can be used to compute the decryption key (or as proposed by this work a trusted interface would be holding this key)- thus giving the server the ability to recover the plaintext if the secret key is compromised. Otherwise, it is not possible to perform a query, since the person generating the query would have to generate the different possible ciphertexts to compare to the field values being searched. There are also no details about how the query execution would be carried out exactly, or what type of database queries are supported other than *select* and *projection* queries.

Another field level encryption based work suggested, similar in nature to [38], suggest XOR-ing the plaintext value with a number based on the database coordinates, before encrypting the result using a symmetric-key cryptographic scheme [41]. The work also proposes a secure indexing scheme, to maintain the database query execution efficiency [41]. The operation for encrypting a cell involves the following operation: $E_k(V_{trc} \oplus \mu(t, r, c))$, where V_{trc} is the plaintext value located in table t, row r and column c, μ is a function that generates a number based on the database coordinate of the field being encrypted, and E_k is the encryption function under private key k. Similar to the work of Denning,

the advantage of this method is that the structure of the database is maintained, you just encrypt the values in the various fields. The second advantage is that an indexing scheme is also proposed, to allow for indexing of the encrypted fields. Finally, since different keys are being used to encrypt each field value, the proposed scheme is not prone to ciphertext searching/substitution weaknesses. It is not clear what the performance hit is for the proposed scheme, relative to an unencrypted database. The author claims that the only overhead of the proposed scheme is, the overhead associated with the encryption and decryption operations [41]. In this case, the encryption operation requires the generation/retrieval of database coordinate information whenever a field is being encrypted, and the decryption operation requires the retrieval of the database coordinate information - operations that have not been discussed, and could potentially affect performance of query execution. Another disadvantage is that the proposed system does not support any arithmetic operation, such as addition, multiplication, average, etc...

Another field level encryption based work suggested the use of tamper free controller, mandatory access control, and a two level subject [69]. In this work, it was proposed to leave unclassified database values to be stored in their plaintext form in order to improve query execution speed, and to perform field level encryption (DES) on private and classified database values. The key generation procedure differs, depending on whether classified data or private data is being encrypted. For private data, the encryption key generation procedure for encrypting cell i, j, is through the use of a random key generation function g, which is being feed a seed $((A_j \oplus ID) \oplus T)$ and the master controller key K, the private key is generated and used to encrypt the field. For classified data, the encryption key generation procedure for encrypting cell i, j, is through the use of a random key generation function g, which is being feed a seed attribute j A_j and master controller key K. The generated keys are securely stored at the master controller, for retrieval by the mandatory access control of the database, to perform the encryption and decryption of the query request/responses. One of the advantages of this system is that it does not alter the structure of the database, encrypting certain fields as required, and leaving the rest in their plaintext form as needed. The second advantage is the key generation function is chosen such that the probability of getting repeated keys is low, and it is also computationally infeasible to determine one element key from other element keys [69]. The proposed scheme also allows for the detection against modification of sensitive data and their classification label, because the plaintext is replicated many times (to fill the encryption block), it can provide authenticity as well as secrecy [69]. Finally, the proposed scheme addresses confidentiality, access control, integrity, and authentication and non-repudiation. One of the disadvantages of the proposed scheme is that not all the fields are encrypted. While this might be acceptable in some applications, it is probably not a wide range of applications.

The second disadvantage is that queries involving a statistical calculation over a range of data in the database, such as sums, averages, and counts, can not be performed directly [69]. This means that if any of these types of queries is to be carried out, all the records involved have to be retrieved, decrypted, before the calculation can be performed. Finally, there is no study about the cost of the proposed system. The authors claim that the cost is *very minimal*, but there is no empirical data to support this claim.

Another implementation that involves field level encryption is the work done by [31]. In this case the authors propose the use of both public key and private key cryptographic systems to encrypt the fields of the database. They achieve access control mechanisms, and data privacy, by encrypting the plaintext field value using symmetric key encryption and a randomly generated private key, and then encrypting the private key using the user's public key. If the user wants to share that field value with other users, then the private key is encrypted using the other authorized users' public key, allowing them to access the database values. The advantage of this implementation, is that it retains the structure of the database. The second advantage is that it achieves both data privacy, and access control on the various fields. Some of the disadvantages of this scheme include the high number of keys that have to be stored and maintained. It is not clear what the performance impact is on storage and retrieval process of these keys. The second disadvantage affecting the performance, is the use of the public key cryptographic system on top of the symmetric key cryptographic system. There are no benchmarks on the proposed scheme, and its effect on the operation of the database. Finally, this scheme does not support any mathematical computations, such as addition, multiplication, and statistical calculations.

2.8 Processing Encrypted Data

The area of processing encrypted data, is still being explored, and still faces a number of challenges. The work on encrypted data processing will pave the way for more companies and individuals, that don't want to give up their data confidentiality, to consider utilizing the cloud for their computing needs. One of the leading problems with processing encrypted data is that very often, in order to perform arithmetic operations, or even any type of general processing (such as sorting), on the data, one has to convert the data back to its non-encrypted origin before performing the required operations [20]. This is the most basic solution for this problem - if you have something encrypted, and you want to process it, then decrypt it and perform all the processing required and then re-encrypt the data again. This of course, is not an efficient solution, and might not be possible in some scenarios. For example, if a user has an encrypted database, if they have to decrypt all the tables

involved in a query, that might take a very long time to decrypt, process the query, and then re-encrypt the data.

While different methods have been developed to allow for the processing of encrypted data, in the context of different applications, it should be noted that some operations (comparisons and search) do not require any special setup [20]. Even though performing a search and comparison on encrypted data is possible without any special setup, the performance is severely affected in database query execution [72]. The reason being that encrypted databases does not allow for data indexing, forcing the database to scan all the records to find a match.

One of the proposed methods of being able to perform keyword-based searches of a large text file, is by creating a concordance of the file [76]. The first step is to break up the file into blocks, and as each block is being encrypted, a concordance is constructed, storing the words in the file, and the block that contains that work. The concordance is stored at a central site, with the encrypted files. When a keyword is to be searched, the encrypted version of the keyword is sent to the site, where the concordance is searched for that keyword and all the blocks containing that keyword are returned. The user then simply decrypts the blocks to retrieve the plaintext. This method is the basis for for an implementation of large database, managed by the STAIRS full-text retrieval system [29]. The advantages of this scheme include: the data is encrypted at the central site and is thus protected against static attacks, there is relatively little overhead in processing (if searching as previously described is all that is done), allows for multi-level security classification through the use of different keys for different files, and the database along with the concordance can be created incrementally - allowing for the growth of the database [29]. The first disadvantage of this method is: if the word being searched for occurs in a large number of blocks, then that will require a large number of retrieval and decryption - incurring a heavy communication and processing cost for the user. The second disadvantage is that this scheme does not support other major database functionality, such as joins, projections, removals, and updates.

Another proposed algorithm for processing encrypted data is done by adding the keyword (key) and the plaintext word [20]. The operation of addition can be a "modulo 2 addition" or any other method used by the system, provided it uses a word size in a modulo L that will prevent loss of data due to overflow [20]. In this algorithm, the first addition of the key to the plaintext amount (P_1) gives the first ciphertext balance (C_1). Afterwards, if additional values want to be added to the ciphertext balance, then you just perform addition on C_1. When a decryption is required, then one can simply subtract the key from the ciphertext to recover the plaintext. In other words, the following operations are performed:

$$\begin{aligned} C_1 &= K + P_1 \\ C_2 &= C_1 + P_2 \\ C_3 &= C_2 + P_3 \end{aligned} \tag{2.7}$$

Now, in order to *decrypt* the sum of P_1, P_2, and P_3, we just have to subtract the key:

$$\begin{aligned} C_3 &= C_2 + P_3 \\ C_3 &= (C_1 + P_2) + P_3 \\ C_3 &= ((K + P_1) + P_2) + P_3 \\ C_3 - K &= [((K + P_1) + P_2) + P_3] - K \\ &= P_1 + P_2 + P_3 \end{aligned} \tag{2.8}$$

Some of the drawbacks of the proposed algorithm are [20]:

- A one-time only breaking of the key enables the decryption of all the data at any time
- The periodic updates that are done in plaintext may increasingly provide information to a potential enemy
- If the balance is ever discovered in an intermediary state, it is possible to trace all the updates
- In order to perform a correct decryption, it is necessary to distinguish between data entering the system as plaintext and data entered as ciphertext

A small variation to the algorithm described above, that was proposed by the same authors, is to add the key to each ciphertext before adding it to another ciphertext. For that solution, they need to keep track of the number of ciphertexts added together, so that you can *subtract* a multiple of the key to get back the sum of the ciphertexts. This still suffers from similar problems as the previous method.

Chapter 3

Proposed System Architecture & Data Encryption Method

3.1 Proposed System

The proposed system has two components: a mobile device, and a cloud service provider (CSP) hosting the database. A diagram depicting the architecture of the system is shown below:

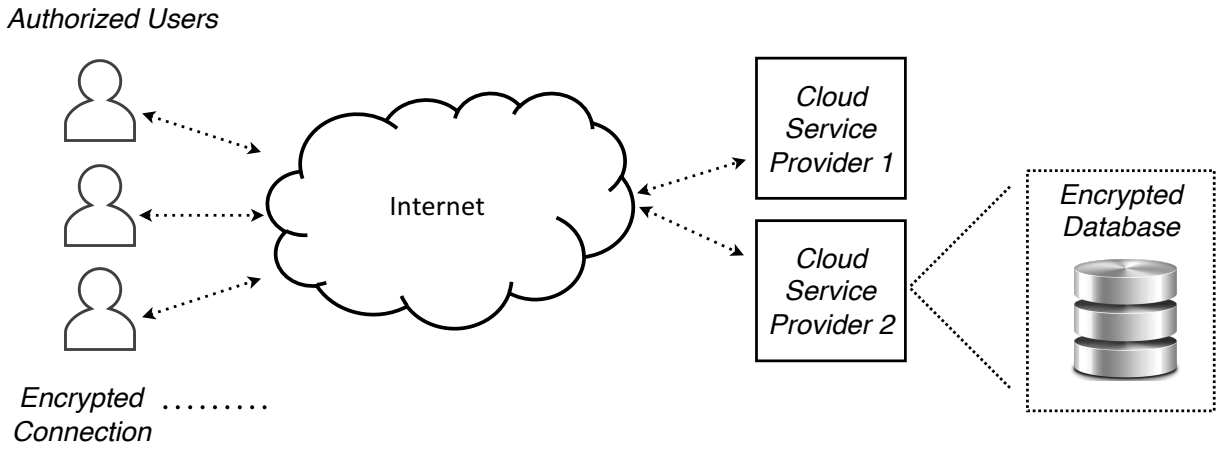

Figure 3.1: An overview of the proposed system architecture.

The proposed system allows multiple authorized users, who have access to the private keys and database metadata, to issue queries to the encrypted database hosted in the cloud. The CSP is accessed by establishing an encrypted connection to protect against eavesdropping, while the data in the database is encrypted to protect the privacy of the data in the database.

An overview of the components of the system, and the operation of the system are shown in the diagram below:

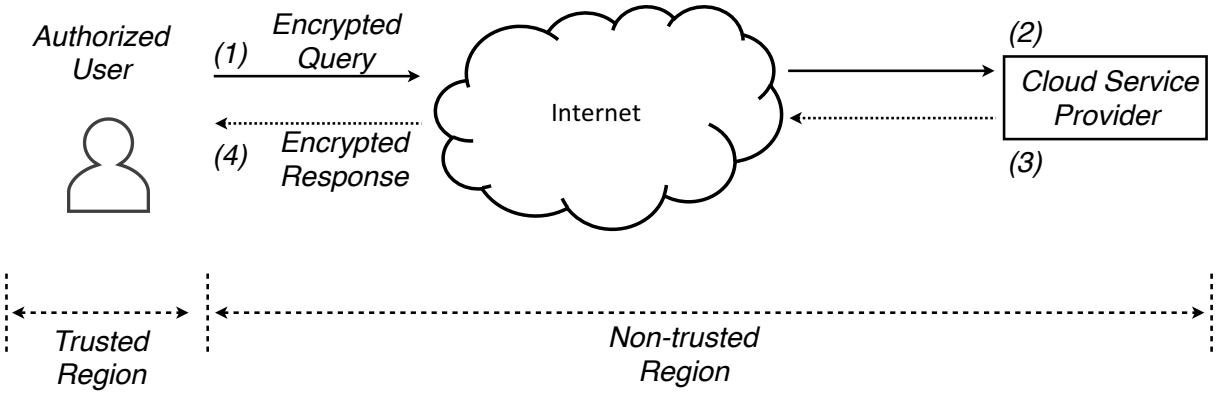

Figure 3.2: An overview of the system operation, and the interaction between the various components of the system.

Specific details of how the various components of the system work, including implementation details, are included in section 4.4. Here we discuss the general usage/operation of the system. An application running on the device of an authorized user has two main purposes: first, the ability to convert a plaintext query to its encrypted form, and second, decrypt the encrypted result and display it to the user. The first step (1), is to use the database metadata (table names and field names) to convert the plaintext query to its encrypted form, and send the query over an encrypted connection to the CSP. The database server hosted on the CSP receives the query (2), and executes it. Next (3), the database server sends back the encrypted records to the client that issued the query. Finally (4), the application running on the authorized device, uses the encryption keys and database meta-data (field types), to decrypt the records appropriately, before returning the plaintext results to the user. Any extra processing required for queries that are not natively supported by the server, are also performed by the application.

3.1.1 Security Objectives

The security objectives for the proposed system, can be split into two main components: data security, and system security. First we will look at the data security component, were we the privacy of the data in the database should also be protected at rest and against chosen plaintext attacks (CPA) from probabilistic polynomial-time (PPT) adversaries. The

data in the database can be split into two groups: values encrypted using AES in counter mode, and data encrypted using Shamir's Secret Sharing Scheme. We did not alter the way AES is being used in counter mode - merely introduced a nonce generation component. We also did not alter the way Shamir is being used, with the exception of keeping the x values of the polynomial secret, the values encrypted using Shamir are unconditionally secure. We define the experiment for the numeric encrypted values for Shamir $\prod$ = (Gen, Enc, Dec), and any adversary A that is a probabilistic polynomial-time adversary:

Definition 3.1.1. Experiment $\text{SHAMIR}_{A,\prod}$:

1. The adversary A outputs a pair of messages m_0, $m_1 \in\subset M$ - corresponding to numeric field values in the database
2. A random bit $b \leftarrow$ 0,1 is chosen, and then a ciphertext $c \leftarrow \text{Enc}_k(\text{m}_b)$ is used to produce an encrypted query q, execute it on the server, and returns all the matching records to the adversary A. We call c the challenge ciphertext
3. The adversary A is given the result consisting of a secret piece value for $c_{x,b}$ for field f, which corresponds to plaintext numeric values m_b
4. The output of the experiment is defined to be 1 if b′ = b, or and 0 otherwise. (In case $\text{PrivK}^{cpa}_{A,\prod}$ (n) = 1, we say that A succeeded.)

We define the experiment for the values encrypted using AES in counter mode $\prod$ = (Gen, Enc, Dec), and any adversary A that is a probabilistic polynomial-time adversary, and any value n for the security parameter as follows:

Definition 3.1.2. Experiment $\text{AES}_{A,\prod}$:

1. The adversary A is given input 1^n and oracle access to $\text{Enc}_k()$, and outputs a pair of messages m_0, $m_1 \in\subset M$ - corresponding to string field values in the database that are of the same length
2. A random bit $b \leftarrow$ 0,1 is chosen, and then a ciphertext $c \leftarrow \text{Enc}_k(\text{m}_b)$ is used to produce an encrypted query q, execute it on the server, and returns all the matching records to the adversary A. We call c the challenge ciphertext
3. The adversary A continues to have oracle access to $\text{Enc}_k()$, and outputs a bit b′.
4. The output of the experiment is defined to be 1 if b′ = b, and 0 otherwise. (In case $\text{PrivK}^{cpa}_{A,\prod}$ (n) = 1, we say that A succeeded.)

The system security objectives for the proposed system, are concerned with the operation of the system, where the adversary shouldn't be able to:

- Determine the database being accessed, the number of queries being performed by a user, the number of records returned for the various queries, and the number of users accessing the database
- Deny service to authorized users
- Maliciously change values in a database

Through the use of an encrypted connection between the device and the CSP, the security objectives for the operation of the system can be achieved against adversary eavesdropping but it, it isn't possible to do the same for the CSP. Since the CSP is hosting the database and processing the queries, it is not possible to protect against the first point. There is also a need to "trust" the CSP will be able to maintain the availability of the service, and protect against unauthorized access to the database such that a person can't maliciously change the encrypted data. The security objectives for the data and system security preclude the following:

- Protecting the data against malicious changes, or incorrect changes due to errors, from the CSP or authorized users (data authentication)
- Number of fields associated with a specific table can not be hidden from the CSP
- Type of query being carried out (select, update, delete), can not be hidden from the CSP
- Field types for a specific table (integer, string), can not be hidden from CSP
- Frequency of access to fields and/or tables, can not be hidden from the CSP

As the system stands in the proposal, if the data stored in the database is changed maliciously, it becomes unrecoverable. For the case of string values, it would have to be first determined that the retrieved value is incorrect, and it would have to be retrieved from another CSP. As for numeric values, it would have to be determined that the recovered value is incorrect, perhaps because the re-construction leads to value that doesn't make sense, and it would have to try to reconstruct using combinations of secret pieces from different CSP. Therefore, if two of the three secret pieces are incorrect, then the numeric value is

unrecoverable. The users's device contains all the secret keys, and database metadata, and so of course, it has to be protected appropriately to avoid exposure of the data. The data in the database has to be protected against malicious changes to the data, as well to maintain data availability for authorized users. The privacy of the data in the database should also be protected at rest.

3.2 Proposed Encryption Method

Data encryption in the database is performed at a field level, where strings and any numeric values not involved in any computations are encrypted using a symmetric key encryption scheme, and numeric values are encrypted using Shamir's Secret Sharing Scheme. In this work, we propose a nonce generation method for using AES in counter mode, and a way of combining the symmetric key encryption system with the Shamir's Secret Sharing Scheme, in order to maintain the privacy of data in database.

3.2.1 Encrypting Strings

Any values not involved in computations are encrypted using a symmetric key encryption scheme such as Advanced Encryption Standard (AES). In order to ensure that the same plaintext does not produce equal ciphertext - a weakness known as ciphertext searching/-substitution vulnerability, two methods were explored: using AES in cipher block chaining mode (CBC) or electronic code book (ECB) mode with a different key for each cell, and using AES in counter (CTR) mode. A formal definition of the usage of the encryption scheme:

1. Let $\mathbf{M}_{i,j}$ denote a plaintext value in row **i** and column **j**
2. Let **IV** be 256 bit string value, generated once during system setup, using a cryptographically secure Pseudo-Random Number Generator (PRNG)
3. Let primary key (**PKey**) be a string value representing the record number (example: 5), field key (**FKey**) be a string value representing the field number/position in the table (example: 3), and table name (**tname**) be a string value representing the table name
4. Let $\mathbf{K}_{i,j}$ be 256 bit string value, generated by concatenating **PKey**, **FKey**, **tname**, and **IV** as follows: $\mathbf{K}_{i,j} = \mathbf{PKey}_i + \mathbf{FKey}_j + \mathbf{TName} + \mathbf{IV}$ with a minimum of 128 bits used from **IV**

5. Let $\mathbf{E}_{K_{i,j}}$ denote the encryption operation using AES under key $\mathbf{K}_{i,j}$

6. Let $\mathbf{C}_{i,j}$ denote a ciphertext value for plaintext value $\mathbf{M}_{i,j}$ generated as follows: $C_{i,j} = \mathbf{E}_{K_{i,j}}(\mathbf{M}_{i,j})$

7. Plaintext recovery is done by first generating $\mathbf{K}_{i,j}$ as previously described, and then doing the following: $\mathbf{M}_{i,j} = \mathbf{D}_{K_{i,j}}(\mathbf{C}_{i,j})$

The key generation process proposed, ensures that the ciphertext is unique across tables, through the table name, unique across rows through different primary keys, and unique across columns, through the field key. The key generation method assumes that the concatenation of *TName*, *PKey*, and *FKey* is less than or equal to 128 bits long, while the rest of the bits of the key will come from the randomly generated *IV*. This will ensure that even if an attacker has aprior knowledge of the table name and primary key, then they will still need to guess at least a 128 bit key comprised of the *IV*, plus the size of the field key *FKey*. Since this is a symmetric key encryption system, the same key is used on the encrypted value to recover the plaintext value.

The alternative was to use AES in counter mode, and generate different nonce values for each cell. A formal definition of the usage of the encryption scheme:

1. Let **IV** be 256 bit string value, generated once during system setup, using a cryptographically secure Pseudo-Random Number Generator (PRNG)

2. Let **K** be 256 bit string value, generated once during system setup, using a cryptographically secure Pseudo-Random Number Generator (PRNG)

3. Let primary key (**PKey**) be a string value representing the record number (example: 5), field key (**FKey**) be a string value representing the field number/position in the table (example: 3), and table name (**TName**) be a string value representing the table name with a minimum of 128 bits used from **IV**

4. Let $\mathbf{S}_{i,j}$ represent a 256 bit string value, which is a concatenation of enough bits from the **IV**, **FKey**, **PKey**, and **TName**

5. Let $\mathbf{E}_K$ denote the encryption operation using AES under key **K**

6. Let $\mathbf{Z}_{i,j}$ denote a string value generated as follows: $\mathbf{Z}_{i,j} = \mathbf{E}_K(\mathbf{S}_{i,j})$

7. Let $\mathbf{M}_{i,j}$ denote a plaintext value in row **i** and column **j**

8. Let $\mathbf{C}_{i,j}$ denote a ciphertext value for plaintext value $\mathbf{M}_{i,j}$ generated as follows: $C_{i,j} = \mathbf{Z}_{i,j} \oplus \mathbf{M}_{i,j}$

9. Plaintext recovery is done by first generating $\mathbf{Z}_{i,j}$ as previously described, and then doing the following: $\mathbf{M}_{i,j} = \mathbf{Z}_{i,j} \oplus \mathbf{C}_{i,j}$

The proposed method for generating the nonce, was inspired from Dennings work on determining unique keys to encrypt each cell [38]. In the proposed system, the IV, Key, and field key, remain part of the secret. The majority of the nonce, will be compromised of the initialization vector, which will ensure that even if an attacker has aprior knowledge of the table name and primary key, then they will still need to guess at least a 128 bits of the IV, plus the size of the field key *FKey*. The experiments were performed using AES in CBC mode, but the results should not vary highly since the AES in counter mode only involves an extra step of performing an XOR operation. The advantages of the nonce/key generation process for the string values is as follows:

- Each cell in the table is encrypted using a unique nonce/key, and thus even if there are repeated values within a record or field, the corresponding ciphertext value will be different
- There are only three pieces of information that need to be stored to perform encryption/decryption: *IV*, *TName*, field key *FKey*, and *K* if using AES in counter mode

The disadvantages of the nonce/key generation process for the string values is as follows:

- If *IV* value is determined, then all the non-numeric values can be exposed because the remaining portion of the nonce/key would be easy to determine
- As the number of fields in the database increase, so does the number of *FKey*. This will require more storage, and processing time during the encryption/decryption process to retrieve the appropriate key - if the keys are spread across multiple tables

3.2.2 Encrypting Numeric Values (Option 1)

Two variations were explored for implementing Shamir's Secret Sharing Scheme, using a linear polynomial. Each variation, has its own advantages and disadvantages, but it is believed that option 1 is less *secure*. In this sub section, we will discuss the first method

explored for encrypting numeric values using Shamir's Secret Sharing Scheme, and the vulnerabilities that make it less desirable than option 2. For cell (*i,j*), containing a plaintext numeric value (M) that will be used in mathematical computations, the encrypted value $C_{i,j}$ would be computed as follows:

$$
\begin{aligned}
E_{KS_n}(M) &= (ax + S)\ mod\ p_c \\
C1_{i,j} &= E_{KS_1}(M) \\
C2_{i,j} &= E_{KS_2}(M) \\
C3_{i,j} &= E_{KS_3}(M)
\end{aligned}
$$

Each generated secret piece (*C1,C2,C3*), will be stored at a different CSP such that at any two of the three secret values are required to re-assemble the secret. For the purposes of this scheme, the x values and prime value p used to generate the secret pieces, will be part of the secret as well. To recover the secret, the user must retrieve two of the three secret pieces, and interpolate using the corresponding x values. The coefficient for the polynomial used to generate the secret pieces will depend on a value generated once at the start from a range of values, using a cryptographically secure pseudorandom value generator:

$$a = (RandValue)\ mod\ p_c \tag{3.1}$$

For a given field in a table, the same coefficient value a will be used to encrypt all the values in that field. The $RandValue$ is generated from a uniform distribution over the integer range one to p, where p is a prime number that is larger than all the secret values being encrypted. The advantages of this method include:

- Irrespective of the number of tables, fields, and records, the two pieces of information that are needed to perform the encryption are the coefficient value a and x, while for decryption you will only need the x values
- The encryption is order-preserving. In other words, secret_piece1 for secret1, will be less than secret_piece2 for secret2, if secret1 is less than secret2. This is also true for all the other secret pieces (secret_piece2 and secret_piece3). This would allow MySQL to natively perform the MIN, MAX, <, and > on the encrypted data
- Supports addition, subtraction, and multiplication on the encrypted values

- If the same x values are used for evaluating the polynomials in different fields, then addition, subtraction, and multiplication is supported across different fields in the table as well. For example, the value from field1 can be added to the value from field2, in the same table. The same is true for fields across different tables

Some of the disadvantages of this method include:

- If the plaintext value of any of the secret values is determined, then all the other values are easily recovered, because the relation between the secret pieces is linear. Therefore, secret_piece1 - secret_piece2 = secret1 - secret2, and therefore, if you have secret 1 and its corresponding secret_piece1, you can determine secret2 by subtracting the difference between the secret pieces. This is also true for determining any of the other secret values

- Equal secret values will produce the same secret piece for a given field, introducing the weakness of frequency analysis on the values for a given field

- The largest value to be encrypted has to be known ahead of time to choose the prime value correctly.

- The largest value to be encrypted has to be less than the largest prime less than 2,147,483,647 (maximum value for signed 32 bit integer), or less than the largest prime less than 9,223,372,036,854,775,807 (maximum value for a signed 64 bit integer)

The first disadvantage mentioned above is very dangerous, because determining one secret value, exposes all the other values. The second disadvantage, not only allows for frequency analysis, but makes it very difficult to re-encrypt the values and keep them secret again, because through frequency analysis of the old data, it can be mapped to the new. For the reasons perviously mentioned, the next option was explored.

3.2.3 Encrypting Numeric Values (Option 2)

The second proposed option for encrypting numeric values, used for the purposes of this project, uses a linear polynomial to encrypt the value. The main difference is that a different coefficient a is used to encrypt each individual value. All the other aspects of the encryption method, remains the same. Therefore, the encryption process will be require one extra step for each numeric value: generation of the coefficient a. A formal definition of the usage of the Shamir scheme:

1. Let secret $\boldsymbol{S}$ be an integer value in finite field $\boldsymbol{F}$ of size $\boldsymbol{p}$ such that $\boldsymbol{S} < \boldsymbol{p}$
2. Let $\boldsymbol{a}$ be a randomly generated integer, chosen from a uniform distribution over integers in the range from zero to $\boldsymbol{p}$
3. Build degree one polynomial $\boldsymbol{f(x) = (ax + S)\ mod\ p}$
4. Let $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$ be integer values used to generate secret pieces $\mathbf{C1}_{i,j}$, $\mathbf{C2}_{i,j}$, and $\mathbf{C3}_{i,j}$ respectively, for secret value $\boldsymbol{S}$ in row i, column j
5. The *encryption* process involves generating the secret pieces $\mathbf{C1}_{i,j}$, $\mathbf{C2}_{i,j}$, and $\mathbf{C3}_{i,j}$ by evaluating the polynomial from step 3
6. To recover the secret (*decryption*), perform interpolation using any two of the three secret values $\mathbf{C}_{li,j}$, with their respective x_l values. Any of the following combinations works: $\mathbf{C1}_{i,j}$, $\boldsymbol{x}_1$ and $\mathbf{C2}_{i,j}$, $\boldsymbol{x}_2$, or $\mathbf{C1}_{i,j}$, $\boldsymbol{x}_1$ and $\mathbf{C3}_{i,j}$, $\boldsymbol{x}_3$, or $\mathbf{C2}_{i,j}$, $\boldsymbol{x}_2$ and $\mathbf{C3}_{i,j}$, $\boldsymbol{x}_3$
7. If the secret being recovered is the multiplication of two secret pieces, then all three secret pieces have to be used because the multiplication produces a polynomial of degree two

Some of the advantages of this method include:

- Irrespective of the number of tables, fields, and records, the two pieces of information that are needed to perform the encryption are the *RandValue* and x, and for decryption (secret re-assembly) the$x$ value
- Supports addition, subtraction, and multiplication on the encrypted values
- If the same x values are used for evaluating the polynomials in different fields, then addition, subtraction, and multiplication is supported across different fields in the table as well. For example, the value from field1 can be added to the value from field2, in the same table. The same is true for fields across different tables
- The linearity between the secret pieces, which existed in the previously proposed method, doesn't exist anymore - removing the associated vulnerabilities

Disadvantages of this method includes:

- The largest value to be encrypted has to be known ahead of time to choose the prime value correctly. Alternatively the largest possible prime has to be selected, from the start
- The largest value to be encrypted has to be less than the largest prime less than 2,147,483,647 (maximum value for signed 32 bit integer), or less than the largest prime less than 9,223,372,036,854,775,807 (maximum value for a signed 64 bit integer)
- MySQL does not natively support *MIN/MAX* and comparison operators natively by MySQL

Even though you lose the advantage of supporting *MIN/MAX* and comparison operators natively by MySQL, but removes two serious vulnerabilities: If any of the secret values is determined, then all the other values are easily recovered, and frequency analysis for equal secret values producing the same secret pieces for a given field.

3.2.4 Encryption Analysis

Advanced Encryption Standard, also known as Rijndael, was chosen because of its high speed and low RAM requirements - allowing it to perform well on a wide variety of hardware[73]. The other advantage to using AES, is there are no known practical attacks that would anyone to recover the plaintext from ciphertext encrypted using AES without the knowledge of the secret key. The use of 128, 192 and 256 key lengths of the AES algorithm, are considered to be sufficient to protect classified information up to the SECRET level, while the TOP SECRET level requires either 192 or 256 bit key lengths [1]. The key being used in the proposed system is 256 bits long. There are no requirements for the key itself, other than its length. The nonce, as described above, would be compromised of a random component, the table name, the primary key of the recover and the field key.

For unconditionally secure secret sharing scheme, the requirements are that each share of the secret must be at least as large as the secret itself, and random bits are required in the formulation of the shares. For Shamir's Secret Sharing Scheme to be unconditionally secure, the requirement is the coefficient of the polynomial are randomly chosen from 0 to p (where p is a large prime larger than the secret), and that each share of the secret must be at least as large as the secret. These requirements are met in the proposed implementation. By definition, if the attacker gains access to one set of the data, they will not be able to learn any new information about the numeric values. In addition to that, since the x-values are part of the secret, then the attacker will not be able to recover the secret even if they

gain access to two secret pieces corresponding to the same secret. This can be easily shown as follows:

Proof. Let S_1 be the first secret, y_1 be the first secret piece recovered, and y_2 be the second secret piece recovered. If values y_1 and y_2 are known, using the equation of a linear polynomial, we have the following two equations:

$$y_1 = ax_1 + S_1 \tag{3.2}$$

$$y_2 = ax_2 + S_1 \tag{3.3}$$

If we substitute equation 3.2 into 3.3 (or subtract 3.2 from 3.3), we get:

$$y_1 - y_2 = a(x_1 - x_2) \tag{3.4}$$

This leaves us with three unknowns. Using the secret pieces for other secrets to try and recover x_1 or x_2, will introduce another unknown coefficient a for each secret - thus making it impossible to recover any of the x values or secret values.

□

For the proposed system, only authorized users should have access to the software which contains the database information and secret keys. It is also assumed that the authorized users are not malicious, or have any malicious intent. Therefore, non-authorized users would not be able to mount a chosen plaintext attack, because they can't provide plaintext and see the corresponding ciphertext, nor can the attacker mount a chosen-ciphertext attack because it would not be able to chose ciphertexts and see its plaintext. will not have access to both the ciphertext and matching plaintext, thus a known-plaintext attack should not be possible either. The only type of attack that needs to be considered is the ciphertext-only attack - since the data is stored in the cloud. For numeric values encrypted using Shamir's Secret Sharing scheme, they are resistant to known ciphertext attacks, because as we have shown in the previous proof 3.2.4, even if the attacker has access to all the secret pieces for a given secret, they still can't learn anything about the secret value used to generate them.

3.2.5 Advantages And Disadvantages of Proposed System

Some of the advantages of the proposed system includes:

- For the nonce generation process for string values, irrespective of the number of tables, fields, and records, only four pieces of information are required to perform the encryption and decryption: the randomly generated value, the table name, primary key, and field key
- For the encryption process for non-numeric values, the only two pieces of information are required: the $\boldsymbol{x}$ values, and the prime $\boldsymbol{P}$ used during the generation process
- Allows for easy record updates, and projection, because the values are encrypted independent of each other
- Each cell will have a unique nonce for encryption, and so there will be no repeated ciphertext for the same plaintext
- Numeric encryption method supports addition, subtraction, and multiplication on the encrypted values
- If any of the values are corrupted at any of the CSP, it can be updated from other CSPs if it was a string value, or it can be regenerated if it was a numeric value (redundancy)

Some of the disadvantages of the proposed system includes:

- Shamir's Secret Sharing Scheme is not space efficient, since each value is broken into three values
- Field values encrypted using AES double in size because of padding and conversion to hexadecimal form. The increase in data has a negative effect on the amount of bandwidth for retrieving the data, and the amount of storage required at the CSP
- Does not natively support GROUP BY, COUNT, and Order BY operations on fields. It also does not natively support partial string matching - aka the value being searched has to be an exact match
- If the data in two of the three databases is compromised or lost, then the numeric values are not recoverable - since you need at least two of the three secret pieces to recover the original secret. Therefore there has to be guarantees by the CSP that the data will not be lost, or intentionally corrupted

- The numeric calculations are bound by two things: the definition of BIGINT in MySQL, and the prime number used to generate the secret pieces. The sum, multiplication, and average calculations have to be less than the maximum value that can be stored by BIGINT (8 bytes, maximum value 9223372036854775807), otherwise the query will fail[12]. In order to be able to decrypt the value correctly, the value has to also be less than prime number chosen to generate the secret pieces
- It is a bit slower than using AES in CBC mode (or counter mode) in the traditional way, because the encryption module has to be re-initialized every time

Chapter 4
System Comparisons

The three systems being compared are: plaintext MySQL database, encrypted MySQL database, and the proposed system. The encrypted MySQL database, is encrypted using MySQL's built in encryption system. Details of the supported query types, and implementation techniques for queries not supported natively are discussed in this section as well.

4.1 Database Details

In this section, we will focus on how the data used in the database was generated, and the data it contains for carrying out experiments. In order to test various functions of the proposed system, the database was setup with tables containing varying field types and varying number of records. The idea is to see how various query execution times are affected by the different field types, and the number of records. The field values were generated using a random data generator website (Mockaroo)[11], and a Python script combining different string values. A copy of the script used is included in the appendix section.

The database used in the experiments consists of five tables: item price, employee sales, position, location, and employee. A summary of the details of each table is shown below:

Table Name	Number of Fields	Field Types	Number of Records
itemPrice	item_id, item_description, item_price	int, string, float	24,312
empSales	sale_id, emp_id, sale_date item_id, num_units_sold, sale_total	int, int, string int, int double	100,000
position	pos_id, pos_name	int, string	4,955
location	loc_id, loc_city, loc_state_province loc_country, loc_address	int, string, string string, string	6,000
employee	emp_id, emp_fname, emp_lname pos_id, loc_id, emp_salary emp_sin, emp_hire_date	int, string, string int, int, double int, date	10,000 -> 1,280,400

Table 4.1: A summary of the tables used in the database for testing the proposed system. Note that there are eight versions of the employee, with the number of records starting at ten thousand, and doubling until we reached one million and two hundred and eighty thousand records.

The data used in the database, includes some assumptions: string values less than 32 bytes, field names and table names are less than 16 bytes, and numeric values are less than the chosen prime value 2147483647. The prime value chosen, took into account the values in the tables, to properly support addition and multiplication between different fields. For numeric values, they are stored using *long signed int* type to avoid any integer overflow issues. The data is used is believed to be realistic, with real location names used, and common first and last names used as well.

4.1.1 Components

To initiate a proper comparison, the software implementation for the plaintext database and the MySQL encrypted database, is very similar in operation to the proposed system. All the implementations, access a database hosted on the same CSP. The software for the laptops/desktops is written in C, while the software for mobile device, running iOS, is written using a combination of C and Objective C. Details regarding the devices and database used in the experiments, are shown in chapter 5. All the implementations makes use of MySQL's C API.

4.2 Plaintext MySQL Database

This is a typical database implementation, where all types of queries supported by MySQL can be carried out. For this case, static strings of the different query types have been generated and stored in local variables. Once a query executes on the server, the result is printed to the console. This represents the best case scenario, with no penalty associated with data retrieval or decryption. A diagram highlighting how the software for accessing the plaintext MySQL database is implemented, is shown below:

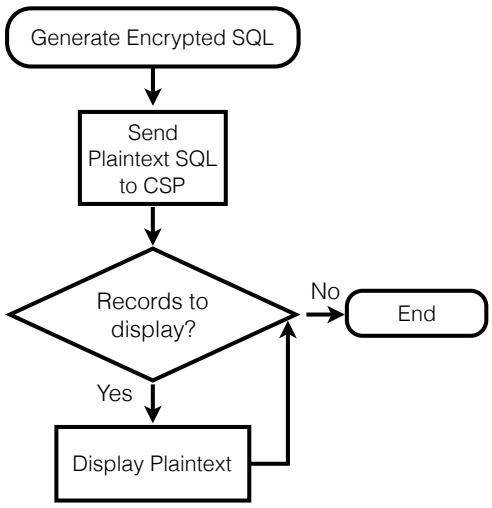

Figure 4.1: A description of how the software for the plaintext MySQL database is implemented. This type of query is used as a base benchmark for comparative purposes.

4.3 MySQL Encrypted DB

This is a software implementation which accesses the database encrypted using the built-in encryption functionality of MySQL. MySQL supports a number of encryption and compression functions, including the official Advanced Encryption Standard (AES) [13]. The functions of interest are: AES_ENCRYPT (for encrypting a value using AES) and DES_ENCRYPT(for decrypting a value using AES), used in AES-256 encryption mode. In order to compare this system to the proposed system the encryption mode for AES is first set to cipher block chaining mode (CBC). The functions perviously mentioned, are

used by supplying three parameters: the value to be encrypted/decrypted, the private key, and the initialization vector. Padding is automatically added by the function to the string value being encrypted, to make it a multiple of 128 bits [13]. Note that the one of the differences between this system, and the one proposed, is that in this case a single key and initialization vector to encrypt the field values are supplied - as opposed different nonce generated for each cell in the table.

4.3.1 Encrypting Existing Database

This subsection discusses how the database is converted into its encrypted form. Three methods were examined:

1. Encrypt one record at a time, and insert it into the encrypted database
2. Encrypt one record at a time, and perform bulk insertion into the encrypted database
3. Encrypt one record at a time, and generate a file formatted in a certain way containing all the encrypted values for the records, and then perform a *LOAD DATA* [14]

Unfortunately, the third method did not work, because the LOAD DATA command returned a parsing error message when a file containing field values with *AES_ENCRYPT* keyword, encryption key, and initialization vector was used. The first method turned out to be the fastest method, with the majority of the time in the second method spent concatenating a large string to execute on the server. In order to achieve the best results using the first method, the database was converted to its encrypted form on the local machine, and then exported to the database system online. The conversion process of the database involves reading records from the plaintext database, generating the encrypted query, and executing that query on the encrypted database. A summary of the steps for converting a plaintext table into its encrypted form, is shown below:

1. Retrieve encryption keys for the all the fields in the table being converted
2. Encrypt field and table names
3. Read a record from the plaintext database
4. Using the appropriate field value and encryption key, generate the MySQL *insert* statement

5. Execute the query on the server hosting the encrypted database

6. Repeat steps 3 to 5 while there are plaintext records to process

A sample MySQL *insert* query, used in the conversion of the plaintext database, is shown below:

```
INSERT INTO `1c7a4f0622bb0d253329c58be2c32e78`
VALUES('%s', AES_ENCRYPT("%s",
'ZXCVBNM,./ASDFGHJKLposition11312',
'ZXCVBNM,./ASDFGHJKLQWERTYUIOP123')))
```

Note that the table being inserted into above, contains only two fields: *position id* and *position name*. The first value, *position id*, which represents the primary key of the table, is not encrypted and remains in its plaintext form. For the field position name, the field value is encrypted using the *AES_ENCRYPT* function which takes three values: the field value for that particular record, the private key, and the initialization vector, respectively. Note that the same key and initialization vector have to be used for all *insert* queries, otherwise the *select* queries will not work.

4.3.2 Supported Queries

In this subsection, we discuss how some of the queries that are not natively supported by MySQL encrypted database can be performed using software - for comparative purposes with the proposed system. Some of the queries that are not natively supported by MySQL encrypted database include: SUM, AVG, MIN/MAX, greater than, and less than. In order to make these queries work, all the values have to be retrieved from the server, and processed by the client. For example, the MIN/MAX query on a particular field would work as follows:

1. Perform a *select* query, with projection on the numeric field to be used to extract the minimum/maximum value - example:

```
SELECT AES_DECRYPT(`bfd56004fc2d85d551a911d4b0514de6`,
'ZXCVBNM,./ASDFGHJKitemPrice11116',
'ZXCVBNM,./ASDFGHJKLQWERTYUIOP123')
FROM `CSP1`.`81ca31489b952789500d7eb9f2c864ae`
```

2. Using a local variable, keep track of the minimum/maximum value encountered so far as you process all the records in the table

3. Once all the records have been processed, return to the user the minimum/maximum value stored in the local variable

For the SUM and AVG query, the same query is executed, but a different calculation is performed before returning the value to the user. For performing the greater than or less than type of query, an *if* statement in the code determines whether the value from the record being processed should be displayed or not.

4.3.3 System Operation

The way this system works is almost identical to the way a plaintext MySQL system works, with the main difference being the need to supply the decryption keys for the various fields in the table. For the purposes of this project, a single key is used to encrypt each field encrypted using AES-256 in CBC mode. To execute a *select* query, the following steps are performed:

1. Retrieve the decryption keys for all the fields involved in the query

2. Encrypt the table and field names

3. Generate the query string by combining the appropriate encrypted field and their respective keys

4. Execute the generated query string

5. Display the decrypted results returned from the server to the user

A sample query for performing a select all on the position table is shown below:

```
SELECT `695b090ef7ee3bf4d54ef59270aa54fd`,
AES_DECRYPT(`fce6c9f0b20c58f058c13cf4637e7a81`,
 'ZXCVBNM,./ASDFGHJKLposition11312',
 'ZXCVBNM,./ASDFGHJKLQWERTYUIOP123')
FROM `CSP1`.`1c7a4f0622bb0d253329c58be2c32e78`
```

A sample query for performing a select all on the position table with a *where* clause:

```
SELECT `695b090ef7ee3bf4d54ef59270aa54fd`,
AES_DECRYPT(`fce6c9f0b20c58f058c13cf4637e7a81`,
'ZXCVBNM,./ASDFGHJKLposition11312',
'ZXCVBNM,./ASDFGHJKLQWERTYUIOP123')
FROM `CSP1`.`1c7a4f0622bb0d253329c58be2c32e78`
WHERE `fce6c9f0b20c58f058c13cf4637e7a81` =
AES_ENCRYPT(`Occupational Therapist`,
'ZXCVBNM,./ASDFGHJKLposition11312',
'ZXCVBNM,./ASDFGHJKLQWERTYUIOP123')
```

As it was shown in the previous examples, all the query processing is completed by MySQL database server, and the decrypted result is returned to the user for display. To ensure there is protection against eavesdropping on the connection with the CSP, a secure connection is established using SSL.

4.3.4 Advantages and Disadvantages

In this subsection, we examine some of the advantages and disadvantages of using the encrypted version of MySQL database. Some of the advantages of MySQL encrypted database includes:

- The encryption/decryption operation is performed on the server. This can be viewed as an advantage, because it reduces the computational load on the device performing the queries
- The cost of converting a database from its plaintext form, to its encrypted form, can be done with less computational resources (as shown in the experiments section)
- If the data is stolen from the server, it is encrypted, and thus protected

Some of the disadvantages of MySQL encrypted databases includes:

- The encryption/decryption operation is performed on the server side, and thus the values are visible to the server in their plaintext form, and thus the server has to be trusted
- The secret key has to be supplied to the server to perform any query

- Does not provide native support for sum, average, min, and max for encrypted numeric values - they would have to be done locally on the device
- The comparison for integer values does not work - only the equal to works

4.3.5 Threat Analysis

In this subsection we examine a potential architecture, where a database encrypted using MySQL is stored in the cloud, and accessed by various devices through the Internet. Areas of interest include any potential points of attack and any information that could be exposed in the system. The architecture of the system consists of three main parts: the users device, the Internet, and the CSP.

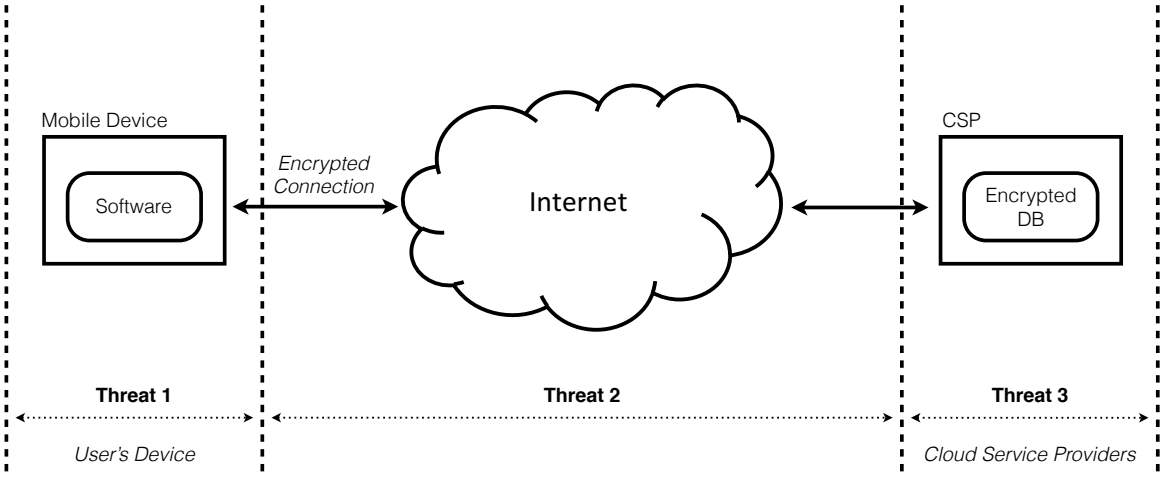

Figure 4.2: Threat model for MySQL encrypted database

The users device is critical because it contains all the encryption keys for the system, as well details about the database, including table names and field names. Therefore, the user's device must be protected to avoid having the keys being leaked and exposing the data. The implications of exposing the secret keys of the database is not just the recovery of the plaintext, but also the cost of having to re-encrypt the entire database and the downtime involved until that is complete. The second group of threats involves the communication between the device and the CSP. Information that could potentially be learned from snooping on the connection link between the device and CSP include:

- The private keys used to encrypt the data

- The table names and field names (if there is a projection on any field)
- The type of query being carried out
- The query results are in their plaintext form, so everything related to the query becomes known

To protect the information discussed from snooping, an encrypted connection has to be established between the device and the CSP. This will ensure that a snooping entity can't learn anything about the data being accessed or the database. The next issue of concern is the availability of the CSP. While the data is encrypted, a compromised server would reveal information about the database when it is processing queries. Finally, preventing authorized access through distributed denial of service attack is an issue of concern, but it is assumed that as part of the service agreement, the CSP should be able to shield those types of attacks - such as Amazons AWS cloud security [25].

The last area of threat involves the CSP. In order for this scheme to work, the server has to be trusted. This of course might not be possible, depending on the client and the application. But in this case, as with the plaintext version, the CSP has the ability to access the data every time it processes a query. This scheme is good for protecting the data during storage, but otherwise it does not provide privacy if the database is hosted at an untrusted site.

4.4 Proposed System

In this subsection, the proposed system is discussed by looking at the work done during the implementation process, including the encryption library used for AES, how the Shamir Sharing Scheme is implemented, the encryption of the database, and how some of the queries are supported.

4.4.1 AES Encryption

The first open source library explored for this project, is MCrypt. MCrypt is a replacement for the Unix *crypt* command, with extensions including AES algorithm [10]. The source code for AES 128 bit, and 256 bit are 1.14 and 1.15 respectively. The source code was

retrieved from the last libmcrypt source available online (v2.5.8). According to the documentation, the code is *written for clarity, rather than speed*, and thus the initial results were very slow in terms of execution time.

The source code was slightly modified, in order to speed up the encryption algorithm initialization process. No modifications have been done on the algorithm itself, with all the modifications being related to the process of loading the algorithm module. The *mcrypt_generic_init* function is called to setup the encryption key and initialization vector, but performs a few steps that can be skipped:

- Retrieve key size for the encryption scheme used
- Retrieve all the key sizes that are supported by the encryption scheme, if more than one exists
- Check the various supported key sizes against the key size being passed by the user to make sure there a compatible size has been passed

The steps mentioned above can be skipped because the assumption is that the correct key size is going to be passed every time. The last set of changes involves the retrieval of the handler for initializing the encryption scheme, and for loading the handler for setting the encryption key. The functions involved are *init_mcrypt* and *mcrypt_set_key*, where a call is made to *mcrypt_dlsym* to search for and retrieve the appropriate handler. The change involves immediate retrieval of the handler, instead of searching for it - saving a considerable amount of time every time the encryption module is initialized to encrypt and decrypt values. Finally, the library has also been compiled with the highest optimization level provided by XCode configuration file, and c/c++ compiler (v4.2.1). Unfortunately, while there was a significant improvement in processing time (up to 20x) with the changes previously mentioned, the results were still not satisfying - and thus a second option (CCCrypt) yielding a 2x improvement over modified version of MCrypt was explored. Note that MCrypt has not been updated since 2012, and thus is not ideal for use since it is not being maintained regularly anymore.

CCCrypt is an interface for a library developed by Apple, providing access to a number of symmetric encryption algorithms [9]. The usage of this interface is similar to that of MCrypt, with general operation involving: initialization of the encryptor with raw key data and options (CCCryptorCreate()), performing the actual encryption (CCCryptorUpdate()), and release of the encryptor (CCCryptorRelease()) [9]. Details of how to use the different methods to perform the encryption, is provided in details in the manual page for

the interface[9]. The encryptor also provides a padding function, but it was not used for the purposes of this project because a padding function was developed to perform the padding when the MCrypt library option was being explored - and thus used for consistence's sake. The exact same implementation was used for both the iPhone and the laptop, with tests over thousands of runs, yielding execution time for CCCrypt AES on the laptop on average 2.2 μs and 5.8 μs on the iPhone.

The encryption of a string involves four steps: encryption nonce generation, string padding, encryption, and converting the encrypted value to a hex string. The nonce generation process uses the table name, primary key, and field key to determine how much to copy from the random string, and then concatenate the rest of the key information. The next step is to add padding to the string before sending it, with the key, to a function which will perform the encryption, and then convert the encrypted string into hexadecimal form. The reason for the last step, is that the encrypted string can contain special characters that would be interpreted incorrectly by MySQL, causing errors when executing the generated MySQL query. The final hex formatted string is sent back to the calling function to be included in the MySQL query. To decrypt a value, four steps are involved as well: nonce generation, converting the hex string to a byte array, decryption, and removing the padding. The first step is to generate the nonce value - the same way it was generated during the encryption process. The second step is to call the decryption function, passing the decryption key. The function will first convert the string to its byte form, perform the decryption, and finally remove the padding before returning the plaintext to the user.

Execution time, measured over a million runs of the AES implementation under a key of size 256 bits and message size of 16 bytes used, yielded on average 5135.4 ns on the iPhone and 2312.05 ns on the laptop, for the encryption and decryption operations.

4.4.2 Shamir Secret Sharing Encryption

Numeric values are encrypted using Shamir's Secret Sharing Scheme. There are two different functions that perform the encryption: one for integer values, and one for double values. The assumption in this project is that all double values, have at most two decimal places, and so the difference between the integer version and the double version of the encryption method, is that for the double version the value is multiplied by a hundred, and the value is converted to an integer - otherwise, everything else is the same. The process of encrypting an integer value involves three steps: converting the string value to an integer, generating a coefficient for the polynomial, and finally generating the secret pieces. The generation of the secret pieces involves one multiplication, one addition, and three modulo

operations. Code for the conversion function is included in the appendix section. The exact same implementation was used for both the iPhone and the laptop, with tests over thousands of runs, yielding execution time for Shamir on the laptop on average 0.211 μs and 0.358 μs on the iPhone.

4.4.3 Encrypting Existing Database

To covert an existing database into its encrypted form, three different ways were examined:

1. Encrypt one record at a time, and insert it into the encrypted database
2. Encrypt one record at a time, and perform bulk insertion into the encrypted database
3. Encrypt one record at a time, and generate a file formatted in a certain way containing all the encrypted values for the records, and then perform a *LOAD DATA* [14]

It was determined that the last method is the fastest one. Details of the cost of converting the database, are discussed in the next chapter.

The overall process for converting the database is summarized in the diagram below:

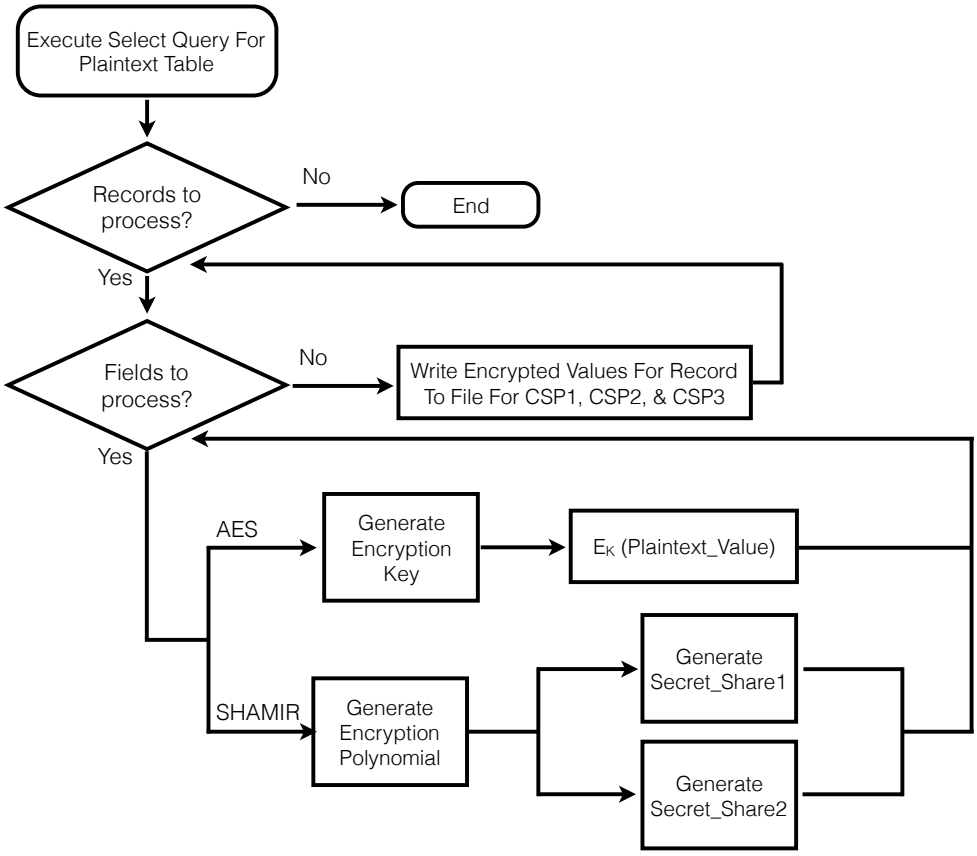

Figure 4.3: A description of how the plaintext database is encrypted in the proposed system

Once the encrypted data is generated, a *LOAD DATA* command is run in MySQL while supplying the file name and the table name containing the encrypted *insert* queries.

4.4.4 Supported Queries

Queries that involve average, sum, addition of two values, or multiplication of two values, can be done natively on MySQL server, with the device just decrypting the returned result. All the other queries will require extra processing from the device initiating the query - details in the next subsection.

4.4.5 System Operation

In this section, we will look at some of the queries that have been tested using the proposed system. It is only a subset of the queries that can be performed, and it is believed that all the queries can be performed using a combination of the queries shown, or with different software methods. The proposed system, depends on metadata that describes the various tables in the database, the fields, and the encryption method used to encrypt each field. The metadata is used to determine the encryption type required to process each type of query. In its simplest (and fastest) form, classes or structs, contain hardcoded information about the table and fields, such that there is very low retrieval time. While this method is very fast, it can be more difficult to maintain, and thus more prone to software bugs. But this is something that is outside the scope of this work since different data structures exist that can be used for storage and retrieval, with varying tradeoffs. In this subsection, we will discuss how the various types of queries are performed in the proposed system. It should also be noted, that it was determined that there is no noticeable advantage to retrieving all the records at once using *mysql_use_result*, before processing it, as oppose to using *mysql_store_result*. This is something that might make a difference if the decryption process is slow, or the bandwidth available for retrieving the records is low, to the point where it can cause an overall slow down in the query processing.

There are a few variations for select queries which include: select all, select with projection on certain fields, select with a where clause, select AVG/SUM/MIN/MAX, and select with a join. The implementation for each type of query, varies slightly, with some queries being natively supported by MySQL DB, and others requiring further processing on the device. As a proof of concept we will discuss in subsection 4.4.5: implementation and operation of some of the select query types, challenges associated with each type, and space and time complexity of each type. We will also perform the same type of analysis/discussion for update and delete queries (4.4.5).

Select Query

For a *select all* query, and *select with projection*, they generally operate the same way. A diagram depicting the general steps involved with processing an encrypted select query for a table containing a mix of numeric and string fields is shown below:

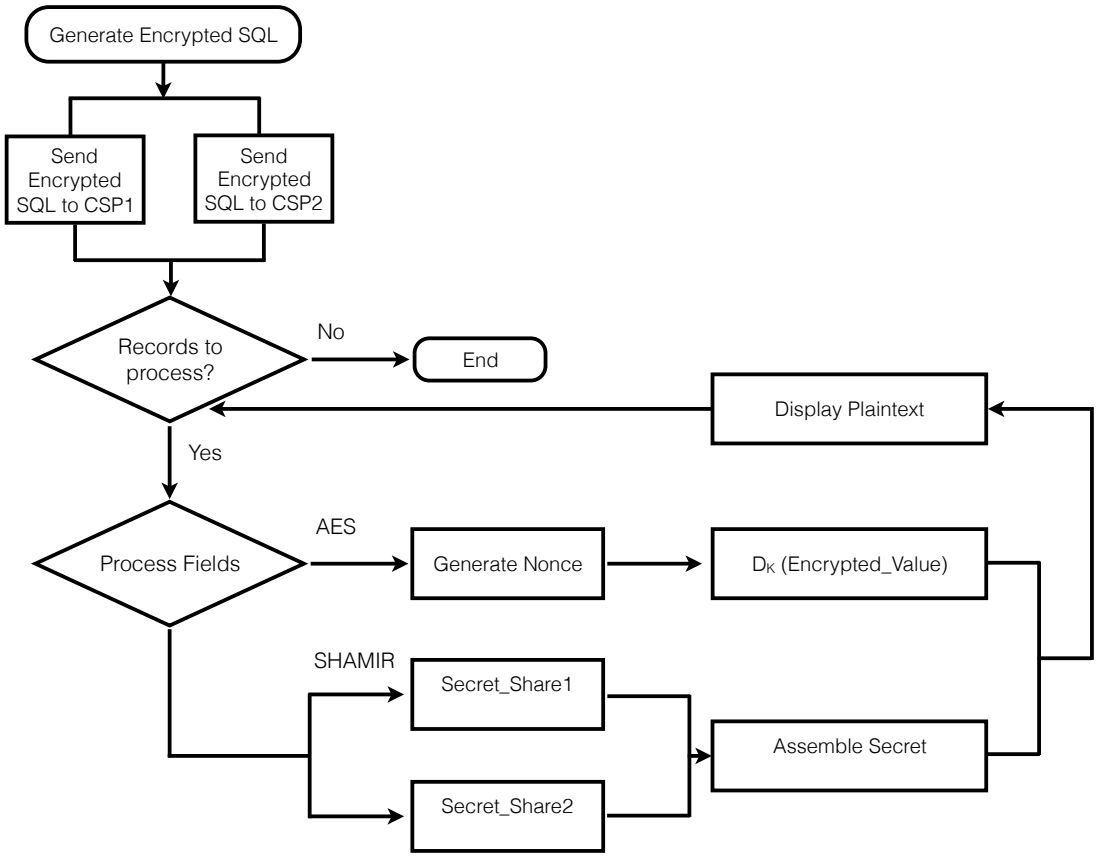

Figure 4.4: A description of the steps to complete a select query in the proposed system for a table containing fields encrypted using AES and Shamir's Secret Sharing Scheme.

The first step in completing a select query, is to convert the plaintext query into its encrypted form. This is done by encrypting the table name, and any field names (for queries with projection), using the table and field encryption key. The query would retain the same form as its plaintext form, with the exception of the field names and table name being encrypted. A sample query for the employee table is shown below:

SELECT * **FROM** `CSP1`.`90cd43ee2ba785dc9b5ffb820ad9571d`

Once the encrypted query is generated, the encrypted query is sent to the appropriate CSP for execution and the returned results are processed independently. In order to decrypt the result, two pieces of information are required: the type of encryption used for that particular field, and the decryption key. The system should be designed such that this information is either hardcoded, or retrieved once, to reduce any overhead with processing the records. For the purposes of this thesis, the information is hardcoded. Since this table contains fields encrypted using both AES and Shamir, the second query is generated such that only the fields encrypted using Shamir are retrieved from the second CSP. This will remove any unnecessary bandwidth usage. For example, the second query for the employee table in our case would be:

```
SELECT '3573d7ecd3e286a5981a824d2e7ca48a',
 'bafc8e08a78cfa167f5a5fb25bb68b08',
 'a1a74d21f31d77750b356f8b54e0abc5',
 '58e6712f8e84e99c91c4d15cab52fdde'
FROM 'CSP2'.'9ab6f8b0f05fa0544205702bb53b8e1d'
```

Note that the employee table contains eight fields, and only four are retrieved from the second CSP - a saving of approximately 256 bytes/record (4 * 64 bytes/field). The final step is to decrypt the values, and display it to the user.

The next variant of select queries, is a *select* query with a *where* clause. A diagram depicting the operation of this variant is shown below:

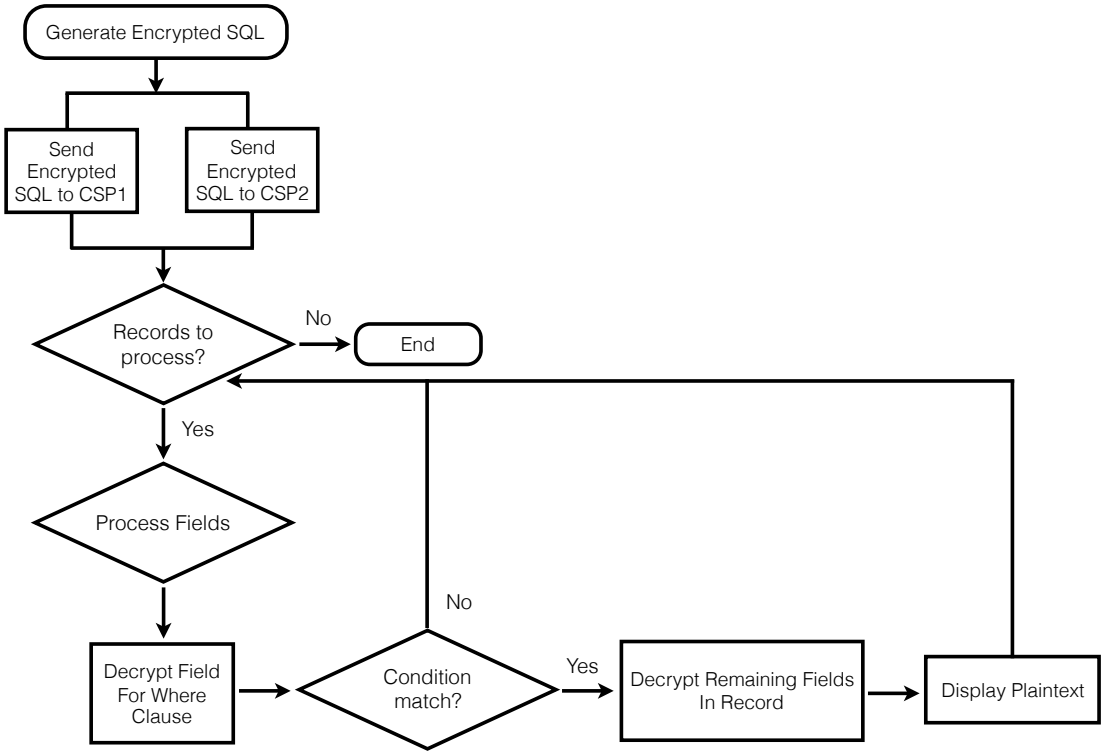

Figure 4.5: A description of the steps to complete a select query, with a where clause in the proposed system

In this case a select all query is performed, and the *where* clause, is processed locally on the device. This is done by first decrypting the field value being matched on, and if there is a match, then the rest of the fields in the record are decrypted appropriately, and finally displayed to the user. This will save time on one of the expensive operations in this process - decryption of AES encrypted field values. The *select min(field_name)* and *select max(field_name)*, work in a similar fashion where a different encrypted query is generated projecting on the fields of interest, and all the records are processed with a temporary variable maintaining the minimum or maximum value.

The next variant of select queries, natively supported by MySQL, are: *select AVG(field_name)* and *select SUM(field_name)*. These types of queries require contacting two CSPs, each returning a single result, just like how it would work on the plaintext version of the database. A diagram depicting the operation of these queries, is shown below:

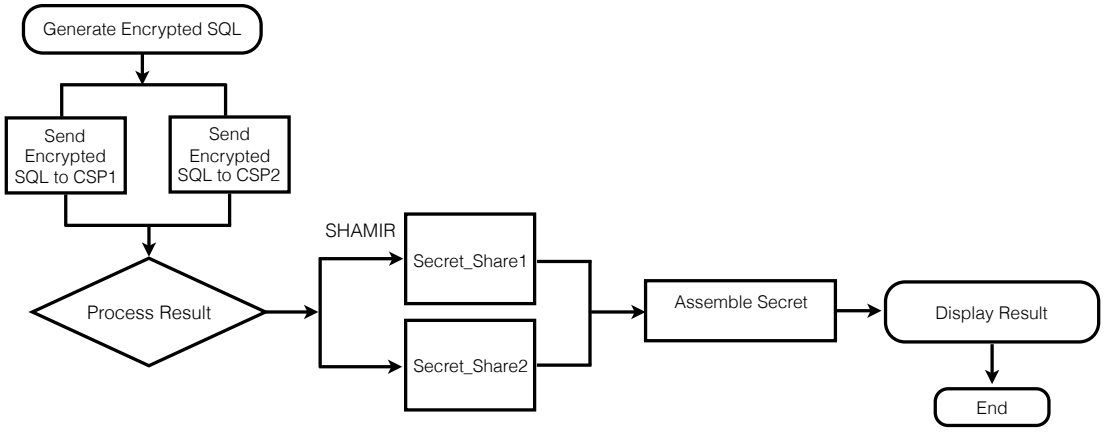

Figure 4.6: A description of the steps to complete a *select average* or *select sum* query on a numeric field in the proposed system.

Note that the addition of two field values, works in a similar fashion, with the only difference being the number of results returned and decrypted.

The final variant of the select queries we will examine, is *right join*. A *right join* is performed on two steps: the fields being joined on are first retrieved (from the referenced table), decrypted, and stored on the device, and then while retrieving the data of the other table, the actual join is performed based on the foreign key in the child table. As an example for the proposed system, a right join query between the location table and employee table was implemented. The query in its plaintext form would be:

```
SELECT location.loc_city, location.loc_state_province,
employee.emp_fname, employee.emp_lname
FROM location
RIGHT JOIN employee
ON location.loc_id = employee.emp_loc
```

The previously described query is carried out in the proposed system as follows:

1. Perform a select query on location table with projection on loc_id, loc_city, and loc_state_province.
2. Decrypt the results from step 1, and store the results in an array of length equal to the number of records returned from the location table with the loc_id being as the index in the array
3. Perform a select query on employee table with projection on emp_fname, emp_lname, and emp_loc
4. Process each record from the employee query. Decrypt all the values of the record returned
5. Use the decrypted value from emp_loc to access the location information stored in the array (if it exists)
6. Display all the field values combined for the record (city, state/province, first name, and last name)
7. Repeat steps 4 and 5, until you have processed all the records in the employee table

This type of query, is not expensive in terms of decryption operations because you are not performing any extra decryptions of fields to complete the query. The cost in this case, is in terms of extra memory usage, to store the foreign key related information of the location table - the larger the number of fields and records, the higher the memory usage. The other potential cost, versus a plaintext database query, is the extra bandwidth used to transfer the field values - encrypted data has to be blocks of 128 bits, and thus, are likely to be padded.

Update and Delete Queries

There are a few variations in update queries, but they are generally the same, where the values of one, or more, fields within a record are updated. The primary key of a record can not be updated, because it is part of the encryption key. Thus, if for any reason the primary key needs to be changed, the record would have to be removed, and re-encrypted and inserted under a new primary key. In most cases, the primary key is auto generated, and this should not be an issue. Update queries are performed over two steps: identify

the primary key of the records that need to updated after going through all the records, and then generate and execute the update queries. A diagram depicting the operation of update queries is shown below:

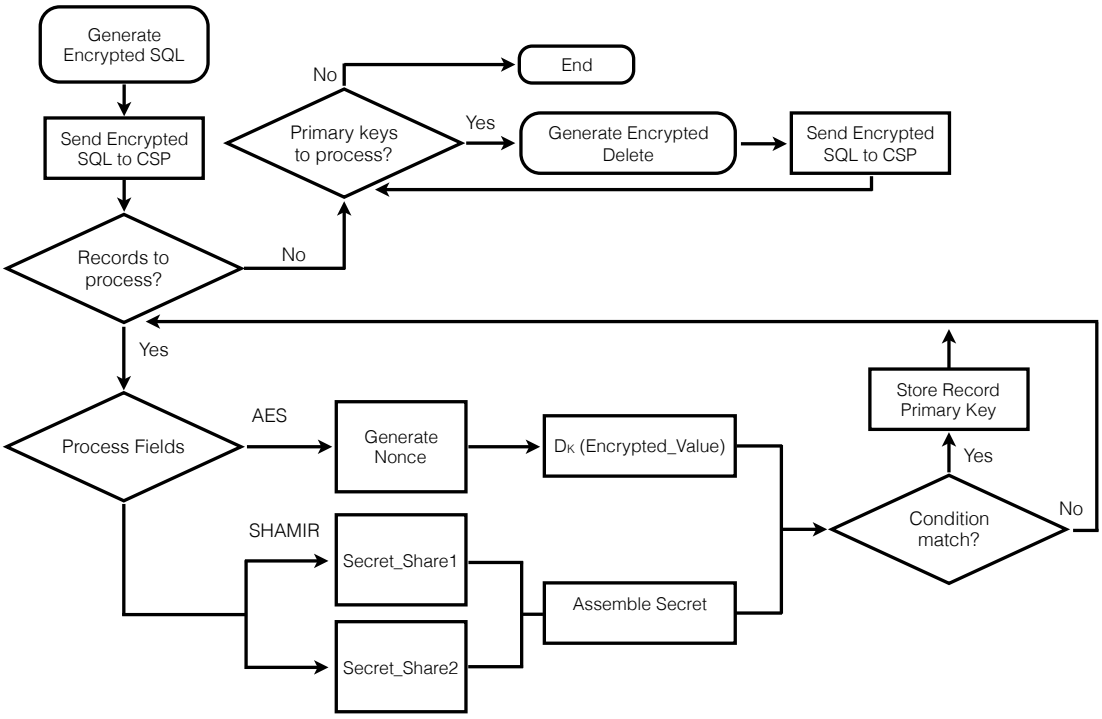

Figure 4.7: A description of the steps to complete a update query in the proposed system

Matching on any value in the record, will require a full table search, and then a single update query. Therefore, the cost of updating records, is a function of the size of the table, and would be expensive in terms of data retrieval and processing for large tables.

For delete queries, a record is deleted based on matching on one, or more, values in the record, and thus it has a very similar implementation to how an update query is done. Delete queries would work exactly the same as update queries, with the exception generating an encrypted delete query with primary key of the records that need to be deleted after they have been identified.

Summary of Query Cost

In this subsection we have a summary of the cost of some of the queries explored, for the proposed system. The cost can be split into two main components: time complexity associated with processing queries, and the bandwidth cost for the data that has to be transferred. There is of course an extra cost associated with some of the queries, such as min/max and join queries, because they are encrypted using the proposed system. A summary of the cost to the device generating the query:

Query Type	**Processing**	**Bandwidth**
SELECT * FROM table1	$m_1(a*c_1+s*c_2)$	$m_1(a+s)$
SELECT * FROM table1 WHERE (field1) = val	$m_1*c_1+l((a-1)*c_1+s*c_2)$ | $m_1*c_2+l((a)*c_1+(s-1)*c_2)$	$m_1(a+s)$
SELECT SUM/AVG(field1) FROM table1	*1*	*1*
SELECT (field1) + (field2) FROM table1	$m_1(c_2)$	$2m_1s$
SELECT (field1) * (field2) FROM table1	$m_1(c_2)$	$2m_1s$
SELECT MIN/MAX(field1) FROM table1	$m_1(c_2)$	$2m_1s$
UPDATE table1 SET (field1) = val WHERE (field1) = val	$2l(a*c_1)$ | $2l(s*c_2)$	$m_1(a)$ | $m_1(s)$
SELECT (field1) FROM table1 RIGHT JOIN table2	$m_1(a*c_1+s*c_2)$ + $m_2(a*c_1+s*c_2)$	$(m_1+m_2)(a+s)$

Table 4.2: A summary of the cost of executing each type of query as a function of: number of string fields (s), number of numeric fields (a), number of records in a given table (m), and number of records matched (l) on in where statement. The cost functions, take into account that the cost of decrypting AES encrypted fields c_1, and the cost of reassembling Shamir's Secret Scheme c_2, are different, with the cost of decrypting AES assumed to be much higher. Here, the cost c_1 and c_2, represent the execution time required to perform the decryption - which will differ, depending on the hardware running it.

For the select queries with a *where* statement, the processing cost is lower, if l is small, because the decryption of the record only occurs if there is a match. The cost of performing a select with a *where* statement on a numeric field, is slightly lower because the majority of the decryption is done on numeric values. The drawback for select query with a *where* statement, is the bandwidth cost - which is equal to that of a *select all*. The *SUM* and *AVG* queries, are performed on the server, and the device only needs to decrypt the result. For the sum and multiplication of fields, the are considered cheap on processing, because they involve decrypting numeric values, but are expensive on bandwidth, because it involves retrieving two parts to of the secret to recover the secret.s comes from The cost of doing queries involving mathematical operations (sum, average, addition, and multiplication) is considered to be very cheap because it involves the decryption of a single numeric value. Since no matching can be done on encrypted data by MySQL server, performing the *select min* or *select max* on numeric fields, requires retrieving the whole dataset rest of the queries list above involves getting the whole data set, and thus, is considered expensive in terms of bandwidth and processing time. The *update* query for a single field value, has a similar processing cost to a select with a *where* clause because you are matching on a field to find the records that require updates. Regarding the bandwidth cost, it is slightly lower for update queries because only the fields being matched on are retrieved - not the whole record. Note that the cost, is two times the number of records being matched on, because you will have to decrypt the new values and send them back to the server.

4.4.6 Advantages and Disadvantages

In this subsection we discuss some of the advantages and disadvantages of the proposed system. These advantages include supported features and operational advantages. A summary of some of the advantages of the system are summarized below:

- Availability: The data is available to any authorized user, at any time - assuming that the service providers are reliable
- Confidentiality: The data is encrypted, and can be safely stored in the cloud without revealing any information to the CSP, or anyone who might gain unauthorized access to the data
- Data redundancy: Data is spread across multiple CSPs allowing the recovery of the data even if one of the CSPs loses the data

- Efficient Computations Offloaded: The homomorphic properties of Shamirs Secret Sharing Scheme allows for some of the data processing to be performed in the cloud. This gives the user the ability to offload mathematically intense computations to the cloud
- Transparency to the user. The database retains its structure, and operates in the same fashion as a plaintext database. The queries also retain their structure, but use encrypted: field names, tables names, and values instead of their plaintext counterparts. This means that legacy systems can adopt the proposed system, allowing for applications that access the database to continue doing so in the same fashion - with a layer in between to perform the encryption/decryption of the queries
- The encryption schemes used are well established, with a known security record
- Cost saving of having a DBA, and an IT department, to manage the infrastructure setup locally

A summary of some of the disadvantages and challenges associated with the proposed system include:

- Initially, there is a cost to generate and upload the database to CSP. For a polynomial degree one, at least three points have to be generated, in order to support multiplication and division. The number of points that have to be generated for numeric values is: (degree of the highest polynomial * 2) + 1. The cost of uploading the database to the CSP is $3x$, if the cost of uploading an unencrypted database to a single CSP is x
- In order to re-assemble a secret result, the secret pieces from two CSPs (assuming a degree one polynomial was used) have to be retrieved, and for numeric results from multiplication the secret pieces from three CSPs have to retrieved
- Keeping all the databases synchronized from multiple users accessing the database at the same can be pose a challenge. This can be an issue because users can be attempting to update fields that are being accessed by other users
- Safe guards have to be in place to ensure that an adversary would not be able to intentionally corrupt the data because if two of the three secret pieces for numeric values becomes corrupted, then that value is not recoverable
- If a string encrypted value becomes corrupted in one (or more) CSP, there has to be a mechanism in place for recoverability

4.4.7 Threat Analysis

In this subsection we look at the various components of the proposed system, the potential points of attack, and the information that could be exposed by the usage of the system. The architecture of the system consists of three main parts: the user's device, the Internet, and the CSP. A diagram depicting the overall architecture of the system is shown below:

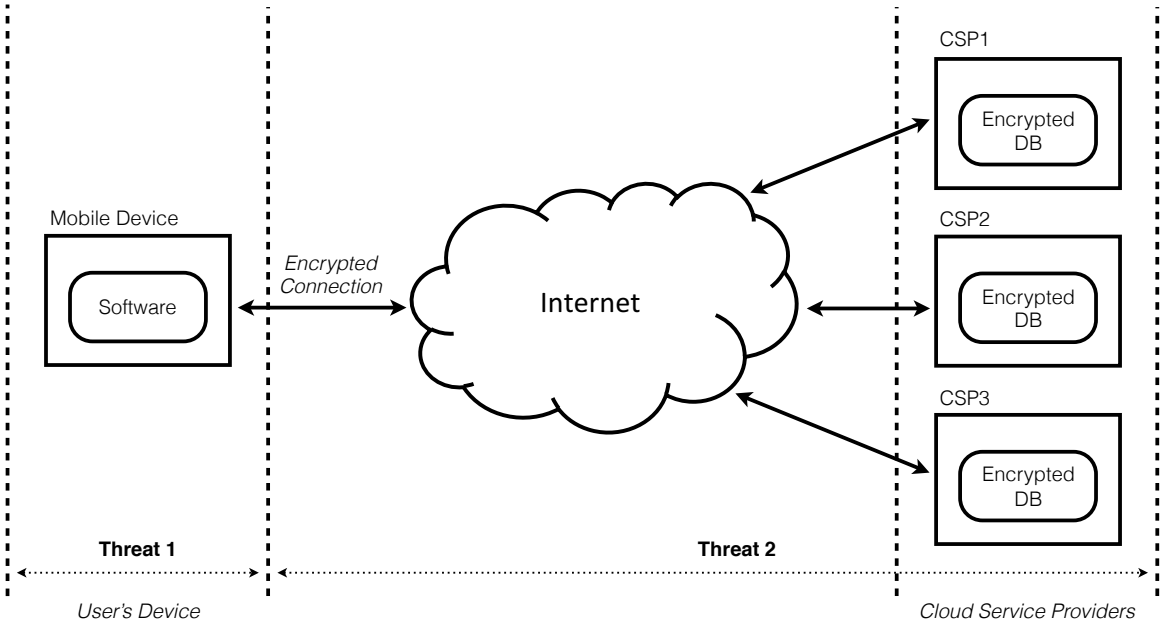

Figure 4.8: A description of the architecture of the proposed system, with the different components that could potentially come under attack

The threats to the proposed system fall into one of two groups: threats to the mobile device, and threats to the CSP. The user's device is critical because it contains all the encryption keys for the system, as well as all the details about the database. Therefore, safeguards must be applied to the user's device to ensure that this information is not leaked or compromised in any way. The implications of exposing this information is not just the loss of private data in the cloud, but also the need to re-encrypt the database from scratch, and the downtime associated with waiting until that is done. The second group of threats target the Internet, and the CSP. Information that could potentially be learned from snooping on the connection link between the device and CSP include:

- The number of records returned from the query, and potentially the number of records for a table if a select all is being performed
- The number of fields associated with a specific table
- The type of query being carried out
- Field types for a specific table
- If the connection is being monitored for a long time, the frequency of any particular table or field being accessed

To protect the information discussed in the previous list, an encrypted connection has to be established between the device and the CSP. This will ensure that a snooping entity can't learn anything about the user's activity with the server, or the data being accessed. The next issue of concern is the availability of the CSP. While the data is encrypted, and so any unauthorized access to the data, will only reveal the information previous listed. Another item of concern is malicious changes to the data stored in the database, which could occur if the CSP is compromised. Finally, preventing authorized access through distributed denial of service attack is an issue of concern, but it is assumed, that as part of the service agreement, the CSP should be able to shield those types of attacks - such as Amazons AWS cloud security [25].

Chapter 5
Experiments, Results, and Analysis

In this chapter, we will start off with looking at the devices to run the experiments on. Next, we will look at the queries to be executed to benchmark the proposed system, and compare it to the MySQL encrypted DB and plaintext DB. Next, we will look at the experiments carried out, as well as the results from the experiments. Finally, we will provide some analysis of the data collected from the experiments.

5.1 Devices

The experiments are performed on two devices, laptop and mobile device. The platform used for the database is MySQL, and it was hosted on Amazon's relational database service (RDS) free tier class. A summary of the the database hosted in the cloud is shown below:

DB Engine:	MySQL
License Model:	general-public-license
DB Engine Version:	5.6.23
DB Instance Class:	1 vCPU[1], 1GiB[2] Network Performance: low
Multi-AZ Deployment:[3]	No
Storage Type:	General Purpose (SSD) baseline of 3 IOPS/GB[4] and ability to burt to 3,000 IOPS
Allocated Storage:	5 GB

Table 5.1: A summary of the DB instance used on Amazon's relational database service (RDS) for hosting the database on. Two instances were used to simulate two CSPs to run the experiments on.

Since the database was hosted on the free tier version of the database, it is considered to be low performance - low network performance, low virtual CPU frequency, and small amount of allocated memory. A summary of the laptop computer specifications used in the experiments to generate the queries and process the results is shown below:

Model:	MacBook Pro Retina Late 2013
CPU:	2.6 GHz Intel Core i5 3MB shared L3 cache
Random Access Memory:	16GB 1600 MHz DDR3
Graphics:	Intel Iris 1536GB
OS:	OS X Yosemite 10.10.5
Storage Type:	Solid State Drive 500GB
WLAN:	802.11ac IEEE 802.11a/b/g/n compatible

Table 5.2: A summary of the laptop used for encrypting the database and benchmarking the proposed system

The laptop being used is over two years old, and it is expected that current computers are more powerful and could potentially yield better results. A summary of the cell phone specifications, used in the experiments to generate the queries and process the results, are shown below:

Model:	iPhone 5s
CPU:	Dual-core 1.3 GHz Cyclone (ARM v8-based)
Random Access Memory:	1 GB RAM DDR3
GPU:	PowerVR G6430 (quad-core graphics)
OS:	iOS 9.02
Storage Type:	Flash 16GB
WLAN:	Wi-Fi 802.11 a/b/g/n

Table 5.3: A summary of the mobile device used for benchmarking the proposed system

[1]Virtual Central Processing Unit

[2]1 gibibyte = 2^30 bytes = 1073741824 bytes

[3]Multi-AZ Deployment is provisioning a failover database in a different time zone

[4]Input/Output Operations per second

The cell phone used for the experiments, is over two years old as well, and Apple has significantly improved the computing power of the recent iPhone. The current generation iPhone has double the RAM of the iPhone used in the experiments, and has a faster CPU running at 1.84GHz.

5.2 Performance Metrics

In this section, we define the performance metrics to compare the proposed system versus the MySQL encrypted database and plaintext database. One of the well known benchmarking processes for databases, is TPC. The purpose of TPC-C benchmarking is to provide relevant, and objective performance data to industry users. The objective of the TPC benchmarkC (TPC-C), is to create a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments [45]. The performance metric reported by TPC-C is a business throughput measuring the number of orders processed per minute, where each transaction is subject to a response time constraint [45].

There are five transaction types included:

- **New-Order transaction**: Represents a mid-weight, read-write transaction with a high frequency of execution and stringent response time.
- **Payment transaction**: Represents a light-weight, read-write transaction with a high frequency of execution and stringent response time. Represents 43% of the total transactions
- **Order-Status transaction**: Represent a mid-weight read only database transaction with a low frequency of execution and response time requirement to satisfy on-line users. Represents 4% of the total transactions
- **Delivery transaction**: Consists of processing a batch of 10 new (not yet delivered) orders. Has a low frequency of execution and must complete within a relaxed response time requirement. Represents 4% of the total transactions
- **Stock-Level transaction**: Represents a heavy read-only database transaction with a low frequency of execution, a relaxed response time requirement, and a relaxed consistency requirement. Represents 4% of the total transactions

The queries executed by the benchmarking system include a mix of:

- Select with a projection and a where clause, matching on string of numeric field values
- Update numeric and string field values, based on a value retrieved from a previous select query
- Insert a complete record based on a value retrieved from a previous select query
- Count query on a certain field

The price per performance is calculated by determining the total price of the system, and dividing that by the reported throughput. The experiments performed include running a different number of concurrent users, and for different run times, on the same number of records. Details of the automatically generated database, extracted from MySQL Workbench tool V6.2.3, include:

Table Name	**Number of Fields**	**Number of Records**	**Table Size (estimate)**
Customer	21	30,000	21.1 MiB
District	11	10	16 KiB
History	8	320,000	36.6 MiB
Item	5	100,000	9.5 MiB
New Orders	3	20,000	1.5 MiB
Order Line	10	3,000,000	324.5 MiB
Warehouse	9	1	16 KiB
Orders	8	300,000	22.0MiB
Stock	17	100,000	35.1 MiB

Table 5.4: A summary of the TPC-C generated database

The fields in the various tables, contain different field types and sizes including: variable string length of different size, numeric field values, and date/time. The parameters used to run the TPC-C benchmark are summarized below:

Parameter Name	Description	Value	Value Used
Number of Warehouses	Size of the database based on the number of warehouses specified during database generation	1 - 20	1
Number of Concurrent Connections	Number of concurrent users in the benchmark	2 - 20	2 - 20
Ramp Up Time (seconds)	Time before starting the benchmark	1 - 9999	10
Measure Time (seconds)	The amount of time the benchmark runs for	1 - 9999	1300

Table 5.5: A summary of the parameters used to run TPC-C

The benchmarking system focuses on determining the number of transaction at which the backend database technology can complete, running on a particular hardware set. The first set of graphs shows the average TpmC values, for different number of concurrent users, running on the plaintext and encrypted versions of the database:

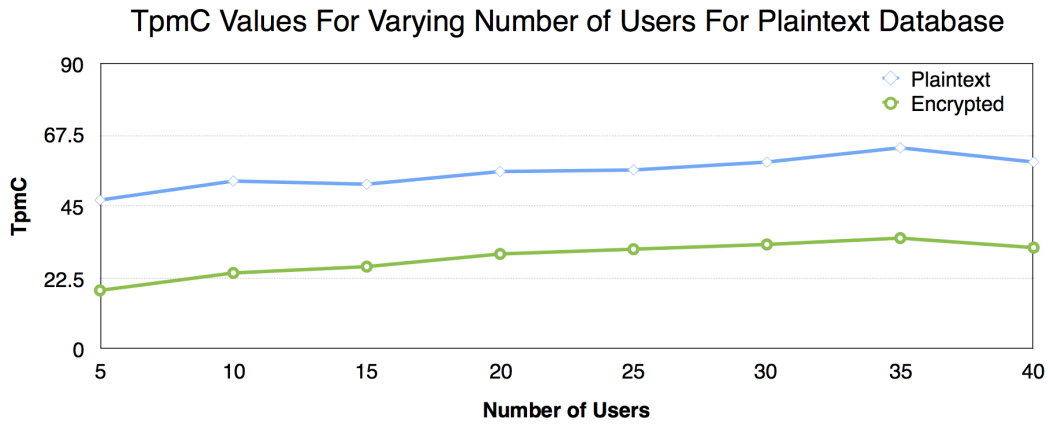

Figure 5.1: Average TpmC values over 1300 seconds for different number of users on plaintext and encnrypted version of the database.

While the TpmC values for the plaintext version of the database are twice that of the encrypted version of the database, the trend is approximately the same for the same number of users. This confirms that the delay in cost, is directly proportional to the number of string values that need to be decrypted by the device, and is fixed irrespective of the number of concurrent users accessing the database. Of course this is still highly dependant on the maximum number of concurrent users that backend can handle.

Next we look at the TpmC values for five and ten users running on both the plaintext and encrypted version of the database:

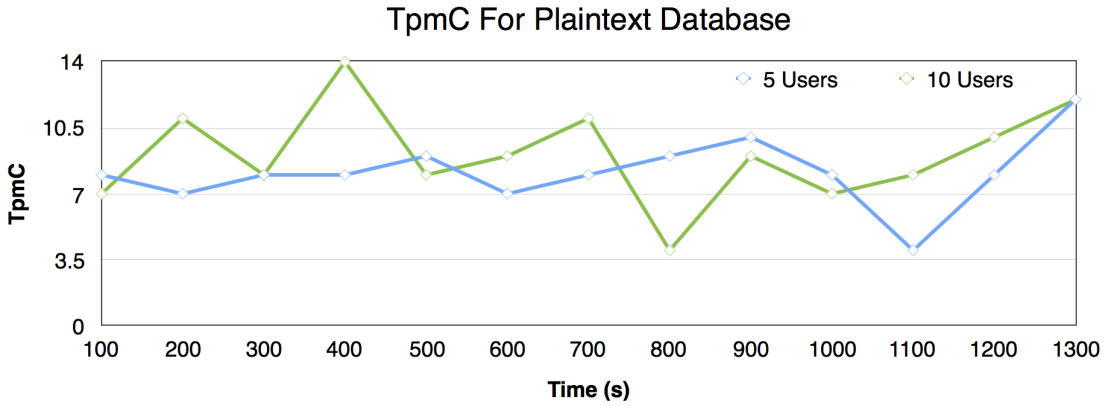

Figure 5.2: TpmC values for five and ten users running on the plaintext version of the database.

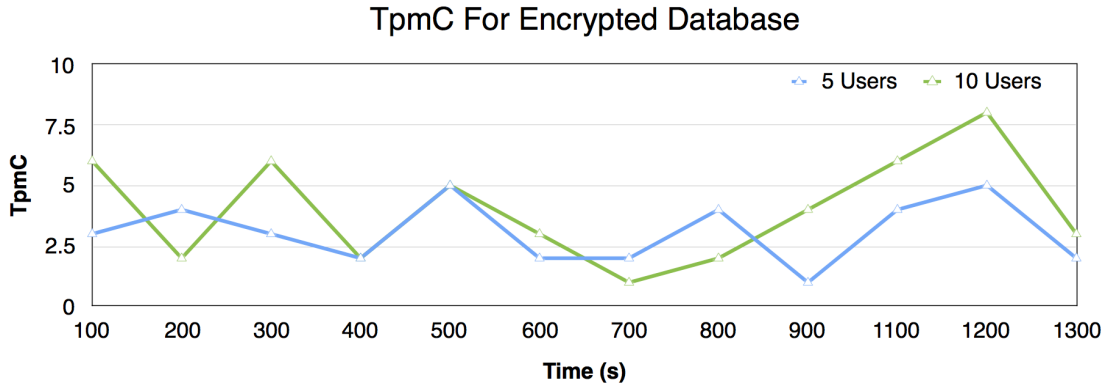

Figure 5.3: TpmC values for five and ten users running on the encrypted version of the database.

Here we see very similar trends between the plaintext and encrypted version of the database, with TpmC values varying from 4 to 14 for the plaintext database, and 1 to 8 for the encrypted version of the database. It is expected that the relative TpmC numbers would remain the same, since the delay due to the decryption on average should remain constant.

A different benchmarking process, which was loosely based on the TPC-C on-line transaction processing benchmark, was devised. For the purposes of this project, different types of queries are executed back to back, hitting different rows in tables of different sizes. To simulate a heavy load, two different machines connected on a separate network, were accessing the database while simulating different number of users (ranging from five to sixty) simultaneously accessing the database at the same time. It was determined that the maximum number of concurrent users supported by the server is approximately sixty for the database instance class used on Amazon. It was also determined that at the maximum number of concurrent users, there was no detectable difference in terms of execution time on the device performing the experiment - and thus, the idea was abandoned. Unlike the TPC-C system, timing of the queries was not strictly based on a model because the main idea was to simulate a worst case scenario - the perceived delay at peak usage. In the experiments, each function representing a query type, was run in different order, a different number of times for a total of one thousand times. For example: run first query ten times, followed by the second query ten times, followed by the first query again five times, and so on. The average is then calculated from the traces (see .6 for sample trace) generated by Instruments.

The purpose of the experiments in section 5.3, is to collect empirical data that can be extrapolated on, to determine the cost of the proposed system in terms of execution time, bandwidth, and memory usage (5.3). The performance measurements are based on our implementation, and are not necessarily the most efficient in terms of execution time and memory usage - but that is an engineering problem that can be solved depending on the requirements set by the client. For the purposes of this work, since the database is being treated as a black box, the performance comparison will be based on the following metrics: query execution and processing time, memory usage, network bandwidth, and storage. Query execution time, is the total execution time including the time it takes to generate the query, send the query to the server, and process (decrypt) the results returned by the server (subsection 5.3.3). Of course, for the plaintext version of the database, the time it takes to process the query will be zero, but the purpose of this comparison is the total time it takes to display the data in its plaintext form for both cases. The second measurement, is how much memory is used to process the query - a potential limitation in mobile devices (subsection 5.3.3). The next major factor, network bandwidth for retrieving the data in

both cases, is an approximation of the size of the records returned. It is expected that more data would have to be transmitted in the proposed system, due to the fact that any information encrypted using AES is at least twice as large, and Shamir's Secret Scheme will require retrieving information from at least two CSPs. Finally, the last measurement of interest is the amount of storage used by the database (subsection 5.3.2).

5.3 Experiments and Results

The purpose of the experiments is to examine the change in cost in terms of execution time, bandwidth, and memory usage, for: varying number of records for the same number of numeric and string fields, and for varying number of fields for the same number of records. While the experiments do not cover every possible combination of fields and records, the database used gives a general trend that can be extrapolated upon, for what the cost of various databases might be - given the mixture of string fields and numeric fields. In order to establish a performance baseline, all the queries will be executed on the plaintext database, the MySQL encrypted database, and on the proposed system. The experiments involve four basic steps: generating the query, executing the query on the CSP, retrieving the results, and print the records to the user on the console. For the proposed system, the extra step required is decrypting the encrypted fields correctly, and getting the secret pieces from other CSPs , if necessary. It was determined, that the maximum number of concurrent users supported on this cloud computing VM is sixty one, and the server can handle the maximum number of users performing a join query simultaneously, with no measurable effect on the performance for the device used for benchmarking. In this section we will look at: the queries to be executed (5.3.1), the cost of converting the database to its encrypted form (5.3.2), the experiments carried out, and analyze the results of the experiment (5.3.3).

5.3.1 Queries

In this subsection, we look at the queries that are going to be executed on the database. The queries are grouped by table, and the summary includes the number of records returned, as well as the record numbers returned. Some of the values for the queries were chosen to match records grouped together, while others were chosen such the matching records are spread out over the table. A summary of the queries carried out on the various tables is shown below:

Query	**# of Records Returned**	**Record Numbers**
empSales all	100,000	1 - 100,000
empSales where saleDate	11	1, 14228, 24938, ... 99592
empSales where empID	101	1, 2025, 2584, ... 97843
empSales where saleTotal	2	22040, 39938
empSales sum(saletotal)	1	N/A
empSales avg(saletotal)	1	N/A
empSales sum(unitsSold)	1	N/A
empSales avg(unitsSold)	1	N/A
location all	6,000	1 - 6,000
location where city	545	3, 8, 17, ... 5995
location where state	293	32, 46, 47, ... 5958
location where country	1,187	1, 2, 6, ... 5993
position all	4,955	1 - 4,955
position where posName	39	129, 292, 342, ..., 4942
position where posName2	42	17, 80, 133,... , 4929
itemPrice all	24,312	1 - 24,312
itemPrice where price	1	12,034
itemPrice where descr.	3	16014, 17004, 24306
itemPrice where descr. 2	5	98, 10927, 14028, 14995, 15173
select sum(itemPrice)	1	N/A
select avg(itemPrice)	1	N/A

Table 5.6: Information about the queries being executed for the **employee sales**, **location**, **position**, and **item price** tables

Query	**# of Records Returned**	**Record Numbers**
select all	10,000	1 - 10,000
select all where fname	20	1 - 20
select all where fname	20	5,000 - 5,020
select all where fname	20	9,981 - 10,000
select all where hireDate	834	14,15,38,...9999
select f1 + f2	10,000	1 - 10,000
select f1 + f2 + f3	10,000	1 - 10,000
select f1 + f2 + f3 + f4	10,000	1 - 10,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 5.7: Information about the queries being executed for the **employee** table with ten thousand records

The information for the rest of the employee tables with different number of records, are available in the appendix section. The number of records, doubles for the employee table, until it reaches approximately one million and and two hundred thousand records. The number of records returned, also doubles for all, but the last two queries.

5.3.2 Database Conversion Costs

The cost of converting the database can be split into four parts: processing time, memory, disk usage, and bandwidth. The cost of processing involves execution time required by the software running on the laptop, to convert the plaintext database to its encrypted form, as described in the previous chapter 4.3.1,4.4.3. The amount of memory usage, includes the amount of RAM required by the software to perform the conversion. Next, is the amount of storage required to store the database on disk until it is uploaded to the CSP. Finally the cost of the bandwidth to upload the database to the CSP - which is directly related to the size of the database. A table summarizing memory usage and execution time for the proposed system and MySQL encrypted database to convert the database is shown below:

Table Name	Memory Usage (MB) Proposed System	Memory Usage (MB) MySQL Encrypted	Execution Time (ms) Proposed System	Execution Time (ms) MySQL Encrypted
itemPrice	2.504	2.524	179.12	258.29
position	0.700	0.74	38.81	43.66
location	0.872	1.04	130.84	60.41
empSales	0.105	0.292	836.33	981.76
employee 1	1.2	1.492	173.86	140.93
employee 2	1.2	1.492	356.31	295.51
employee 3	1.2	1.492	708.58	486.92
employee 4	1.2	1.492	1,415.05	1,004.29
employee 5	1.2	1.492	2,856.60	1,949.74
employee 6	1.2	1.492	5,590.20	3,678.56
employee 7	1.2	1.492	10,977.30	8,884.26
employee 8	1.2	1.492	21,594.12	13,924.62

Table 5.8: A summary of the processing cost of converting the database to its encrypted form, using the developed implementation for the proposed system and MySQL encrypted database. The values are averaged over a thousand runs on the laptop.

The memory cost of converting the database is more or less the same for the proposed system and MySQL encrypted DB, so there is no advantage to using one over the other. The software written for converting the database, in both cases, is single threaded, but the

key difference is, the encryption of the records in the proposed system is performed on the laptop, while for the MySQL system its performed on the CSP, which is multi-threaded and makes use of multiple CPUs[15]. Therefore, there is a clear difference in execution time between the proposed system and the proposed system. This distinction is clear in the employee tables, where as the number of records doubles, the execution time doubles for the proposed system, but the increase is not linear for MySQL server. For simplicity purposes and to explore the worst case scenario for the proposed system, a single threaded program was developed. If the database being converted is large, it would make sense to develop a multi-thread program to decrease the execution time. The next aspect of interest for the cost, is the amount of disk storage space, which affects both the amount of disk space required to host the database, as well as the bandwidth used to upload it to the server. A table summarizing disk usage is shown below:

Table Name	Plaintext Average Row Length (bytes)	Proposed Encrypted Average Row Length (bytes)	MySQL Encrypted Average Row Length (bytes)	Plaintext Table Size (MiB) [5]	Proposed Encrypted Table Size (MiB) [5]	MySQL Encrypted Table Size (MiB) [5]
itemPrice	26	88	62	1.36	2.7	2.1
empSales	28	95	104	4.6	11.1	12.7
position	28	80	43	250.68 x 10^{-3}	530.1 x 10^{-3}	353.1 x 10^{-3}
location	56	276	98	389.22 x 10^{-3}	1.7	752.1 x 10^{-3}
employee	46 - 55	244 - 244	145 - 154	780.90 x 10^{-3} - 96.2	3.1 - 305.4	1.7 - 205.6

Table 5.9: A storage comparison between the plaintext tables, and their respective encrypted forms. The data was collected from MySQL Workbench table information page - sample in appendix 1.

In the proposed system, tables that contain only string values, double in size - as shown in the position table. This is expected since the string value is padded to reach 32 bytes

[5] 1 MiB = 2^{20} bytes = 1024 kibibytes = 1048576 bytes

before being encrypted and converted to a hex array - which doubles the size to 64 bytes. A table like the employee table, having four numeric fields and three string fields, grows on average a little over six times in size - as oppose to sixteen times, if they were all string fields. So while the numeric values are split into three pieces, the individual pieces do not use up as much space as the AES encrypted string. There is a difference between the MySQL encrypted values, and the proposed system values in terms of the expected storage size. For the tables containing only string values, such as location and position table, the storage space should be exactly the same, but this is not the case. The position table in the proposed system is approximately 1.5x larger than the MySQL encrypted version, even though the same encryption type is used, and its suppose to produce 128 bit blocks. It seems that in the MYSQL encrypted database, the data is padded to the closest multiple of 128 bits blocks for short strings, as oppose to what was done in the proposed system where all the values are automatically padded to 256 bits blocks - and thus saving on space.

5.3.3 Query Cost & Analysis

In this subsection, we will look at the experiments for carrying out the different types of queries on the laptop and mobile device. We will look at two aspects: memory usage, and execution time. A summary of the memory usage for processing the queries for each table for the laptop and iPhone are shown below:

Query for table:	**Plaintext Memory Usage (MB)**	**MySQL Memory Usage (MB)**	**Proposed Memory Usage (MB)**
position	≈ 0.280	$\approx 0.380 - 0.390$	≈ 0.420
item price	≈ 0.280	$\approx 0.400 - 0.410$	$\approx 0.435 - 0.460$
location	≈ 0.280	≈ 0.420	≈ 0.500
employee sale	≈ 0.280	$\approx 0.405 - 0.450$	$\approx 0.440 - 0.500$
employee	≈ 0.280	$\approx 0.415 - 0.485$	$\approx 0.350 - 1.930$

Table 5.10: A table summarizing the memory usage on the iPhone for queries executed on the plaintext database, MySQL encrypted database, and the proposed system.

Query for table:	**Plaintext Memory Usage (MB)**	**MySQL Memory Usage (MB)**	**Proposed Memory Usage (MB)**
position	≈ 0.200	≈ 0.310	≈ 0.320
item price	≈ 0.200	$\approx 0.320 - 0.336$	$\approx 0.330 - 0.340$
location	≈ 0.200	≈ 0.330	≈ 0.350
employee sale	≈ 0.200	$\approx 0.304 - 0.332$	$\approx 0.340 - 0.380$
employee	≈ 0.200	$\approx 0.320 - 0.396$	$\approx 0.330 - 1.910$

Table 5.11: A table summarizing the memory usage on the laptop for queries executed on the plaintext database, MySQL encrypted database, and the proposed system.

Since a single record is retrieved at a time, the memory usage is directly proportional to the amount of memory required to store a single record in memory - and thus the variation in memory usage is not that high between the various query types. Note that the memory requirements for MySQL encrypted database, is lower, because the records are already decrypted - and thus in terms of storage is very close to that of the plaintext. If there are more fields in each record, the memory usage would be higher. It was determined that the memory usage increases linearly based on the field size. In other words, for every added string field the memory usage increases by approximately 64 bytes, and for every integer field the memory usage increases by approximately 32 bytes.

Now we examine the execution time on the device for the different types of queries, and compare it to the MySQL encrypted database and plaintext database. For the position table, containing a single field encrypted using AES, a graph comparing the execution time for the proposed system versus the plaintext and MySQL encrypted database is shown below:

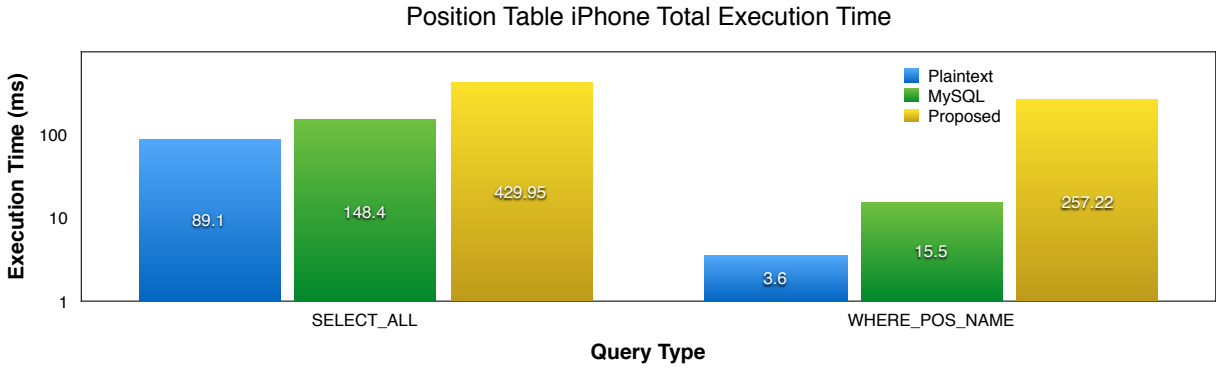

Figure 5.4: Execution time for queries executed on position table on the iPhone

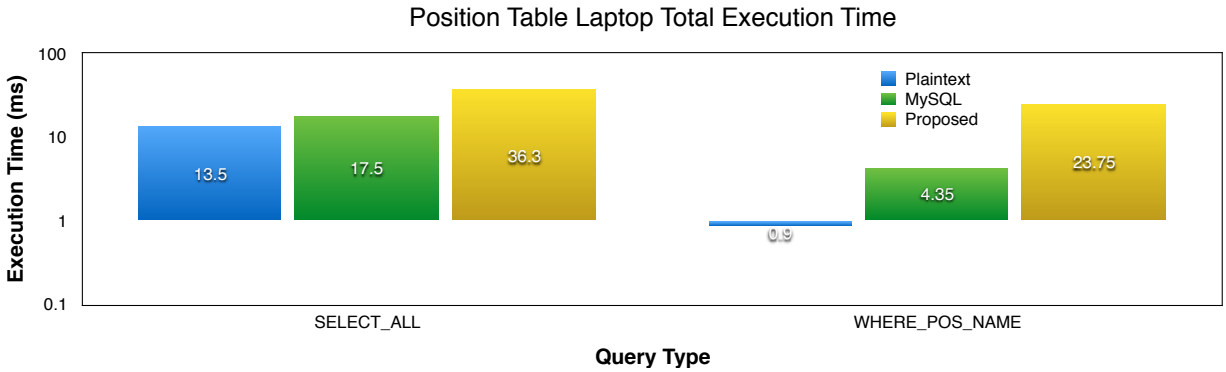

Figure 5.5: Execution time for queries executed on position table on the laptop

We can see from the data gathered from the position table, that the cost of performing a *select all query* for the proposed system, is approximately $4.8x$ for the iPhone, and $2.7x$ for the laptop, compared to the plaintext database. If we compare the proposed system to the MySQL encrypted database, we find there is a $2.9x$ and $2x$ fold increase for the iPhone and laptop respectively. Overall, the execution time is lower for the laptop - which can be expected, because the laptop has higher computational power. This indicates that as the computational power continues to improve, the gap between the proposed system

and plaintext version will shrink further. The cost of performing a *select all* with a *where* clause for the proposed system, is approximately $71x$ for the iPhone, and $26x$ for the laptop, compared to the plaintext database. Comparing the cost of the proposed system to the MySQL encrypted database, we find there is a $16.6x$ and $5.5x$ fold increase for the iPhone and laptop respectively. While the execution time is lower for the *select* query with a *where* clause in the proposed system, it is still considered to be high relative to the other systems. It is also expected that the execution time for the *select* query with a *where* clause will increase, if the number of matching records increases. The *select* query with a *where* clause endures a much higher penalty, mainly because the proposed system has to retrieve all the records, and decrypt them locally to find the one that matches.

Now we look at a breakdown of the execution time for the position table, to determine where the bulk of the processing occurs. In the table below, we have a summary breakdown of the components that contribute to the extra processing for the proposed system:

Position Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all_where _pos_name	258.17	180.12	17.5	0	88.06	1.0439
select_all	430.104	176.6	21.09	0	152.21	49.8

Table 5.12: A breakdown of the position table execution time, including total execution time, for the proposed system on the iPhone.

Position Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all_where _pos_name	23.75	11.85	1.9	0	7.7	0.25
select_all	36.3	12.6	1.9	0	10.02	9.3

Table 5.13: A breakdown of the position table execution time, including total execution time, for the proposed system on the laptop.

For the proposed system, the bulk of the time spent performing the extra processing, is due to the decryption of the field values, and key generation process. On the laptop, this

amounts to approximately 58% of the total execution time for the *select all* with a *where* clause, and 40% for the *select all* query. On the other hand, for the iPhone, approximately 76% of the total execution time for the *select all* with a *where* clause, and 46% of the time on *select all* query.

For the item price table, containing a mixture of string and numeric values, a graph comparing the execution time for the proposed system versus the plaintext and MySQL encrypted database is shown below:

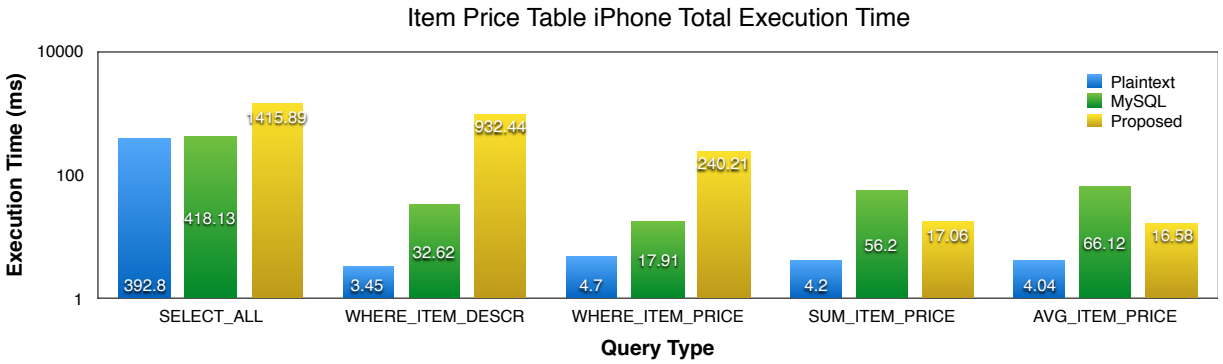

Figure 5.6: Execution time for queries executed on item price table on the iPhone

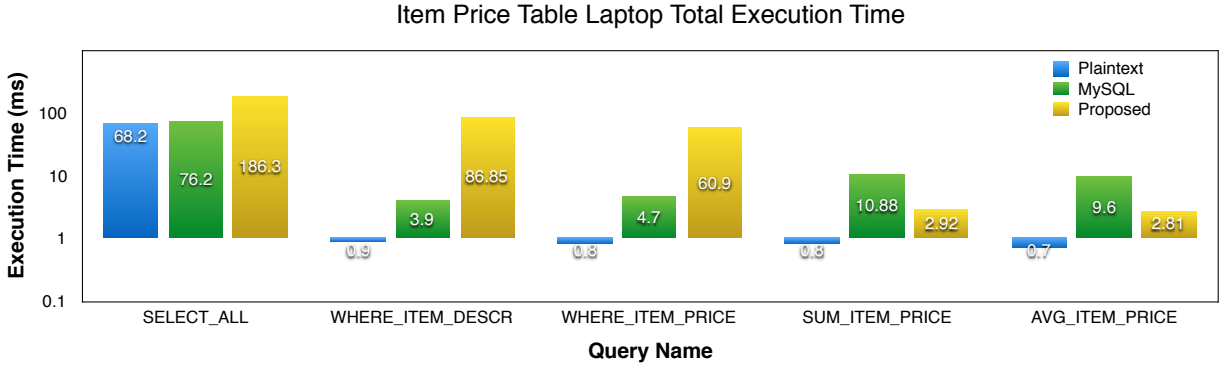

Figure 5.7: Execution time for queries executed on item price table on the laptop

The item price table, increases the number of records four times from the position table, and combines different field types. Again we see a similar trend to what we have seen in the position table, where there is a big difference in the execution time between the *select all* query, and the *select* with a *where* clause string values. The important thing to note

here, is that the gap in execution time between *select all* query, and the *select* with a *where* clause, has increased significantly, as the number of records has increased. In this case, the *select* with a *where* clause has a $270x$ and $96x$ fold increase for the iPhone and laptop respectively. This is a significant jump in the processing time. Comparing the *select* query done on a **string** value, versus on a **numeric** value, for the proposed system, we see that the execution time is $3.9x$ faster for the **numeric** value on the iPhone, and $1.4x$ faster for the laptop. Finally, the *sum* and *average* query execution times are approximately $3x$ and $5x$, for the iPhone and laptop, respectively, than the MySQL encrypted DB - a significant difference. Now we look at a break down of the execution time for the item price table in the proposed system:

itemPrice Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	1416.72	654.3	82.9	4.06	305.14	262.701
select_all_where _itemDescr	933.11	626.15	55.19	0.771×10^{-4}	183.5	0.914
select_all_where _itemPrice	240.3	1.09	0.1934	2.2	182.16	0.839
select_sum _itemPrice	17.06	0	0	0.713×10^{-5}	14.94	0.18
select_avg _itemPrice	16.58	0	0	0.913×10^{-5}	14.46	0.11

Table 5.14: A breakdown of the item price table execution time for the proposed system on the iPhone.

itemPrice Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	186.3	66.8	15.1	3.2	32.03	57.4
select_all_where _itemDescr	86.85	43.65	8.35	0.816×10^{-6}	29.4	0.05
select_all_where _itemPrice	60.9	8.264×10^{-6}	0.2264×10^{-6}	2.2	47.8	0.11
select_sum _itemPrice	2.92	0	0	0.198×10^{-6}	2.49	0.02
select_avg _itemPrice	2.81	0	0	0.198×10^{-6}	2.36	0.05

Table 5.15: A breakdown of the item price table execution time for the proposed system on the laptop.

For the proposed system, some of the important information that we can extract from the breakdown of the execution time includes:

- For the laptop, the amount of time spent on encryption related tasks include: 46% for *select all*, 60% for a select with a **string** *where* clause, and 4% select with a **numeric** *where* clause
- For the iPhone, the amount of time spent on encryption related tasks include: 52% for *select all*, 73% for a select with a **string** *where* clause, and 1.45% select with a **numeric** *where* clause
- For the laptop, less than 1% of the time is spent on encryption related tasks for the *sum* and *average* queries
- For the iPhone, less than 1% of the time is spent on encryption related tasks for the *sum* and *average* queries

Next, we look at the location table graphs comparing the execution time for the proposed system versus the plaintext and MySQL encrypted database:

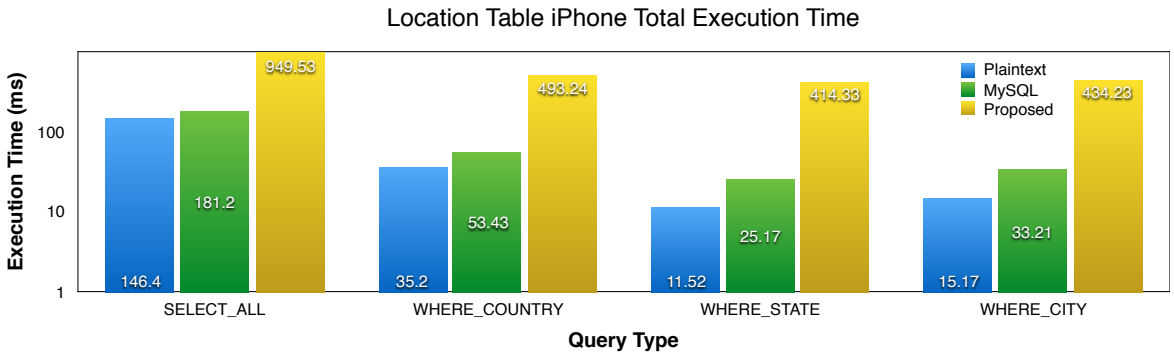

Figure 5.8: Execution time for queries executed on location table on the iPhone

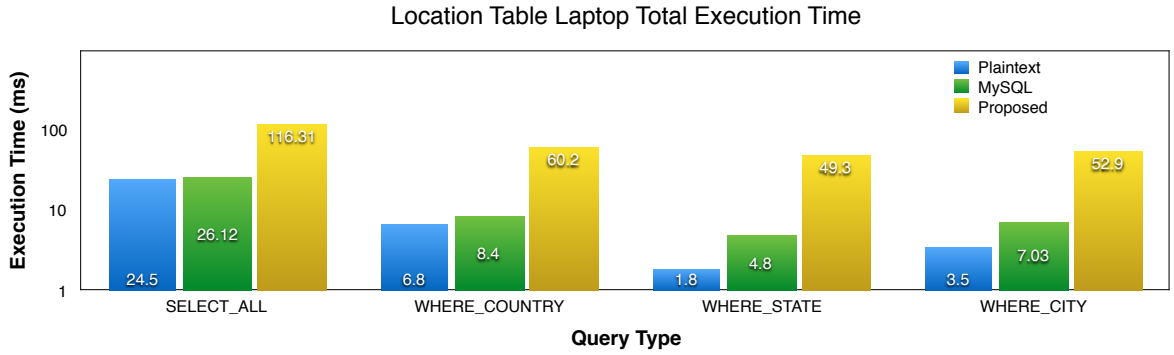

Figure 5.9: Execution time for queries executed on location table on the laptop

One of the important trends to extract here, is that extra cost for processing for *select* queries with a *where* clause, decreases as the number of records returned by the query increases. For example, the cost of performing a *select* matching on *state*, is approximately $37x$ relative to the plaintext database on the iPhone, and $27x$ for the laptop. On the other hand, for the *select* query matching on *city*, which returns approximately twice as many records, the cost is approximately $28x$ relative to the plaintext database on the iPhone, and $15x$ for the laptop. As the number of records returned doubles again for matching on country, the cost is approximately $14x$ relative to the plaintext database on the iPhone, and $8.8x$ for the laptop.

Now we look at a break down of the execution time for the location table in the proposed system:

Location Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	950.04	559.34	60.022	0	217.44	56.130
where_country	494.77	297.9	30.556	0	121.6	11.05
where_state	415.75	221.0	25.441	0	138.332	3.005
where_city	435.0	238.861	25.021	0	130.966	11.22

Table 5.16: A breakdown of the location table execution time for the proposed system on the iPhone.

Location Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	116.31	61.6	10.8	0	20.4	14.9
where_country	60.2	25.1	4.091	0	25.3	2.7
_where_state	49.3	16.2	2.8	0	25.6	1.4
where_city	52.9	18.9	3.021	0	26.2	1.4

Table 5.17: A breakdown of the location table execution time for the proposed system on the laptop.

For the proposed system, some of the important information that we can extract from the breakdown of the execution time includes:

- For the laptop, the amount of time spent on encryption related tasks include: 61% for *select all*, 48% for a select matching on *country*, 41% for a select matching on *city*, and 39% for a select matching on *state*
- For the iPhone, the amount of time spent on encryption related tasks include: 65% for *select all*, 66% for a select matching on *country*, 61% for a select matching on *city*, and 59% for a select matching on *state*

A trend to consider here, is the effect of increasing the number of string fields on the execution time. If we compare the location and position tables, we find that it takes approximately $0.158ms$ to process a single record, while it takes $0.0858ms$ for the position table. While the number of string fields increased four times, the time it takes to process a record only increased $1.8x$. The decryption occurs serially in both cases, and thus it is possible that as the number of fields increases, the increase in processing time becomes linear (or worse).

The next table we will look at, is the employee sales table, which contains three numeric values and two string values. A graph comparing the execution time for the proposed system versus the plaintext and MySQL encrypted database is shown below:

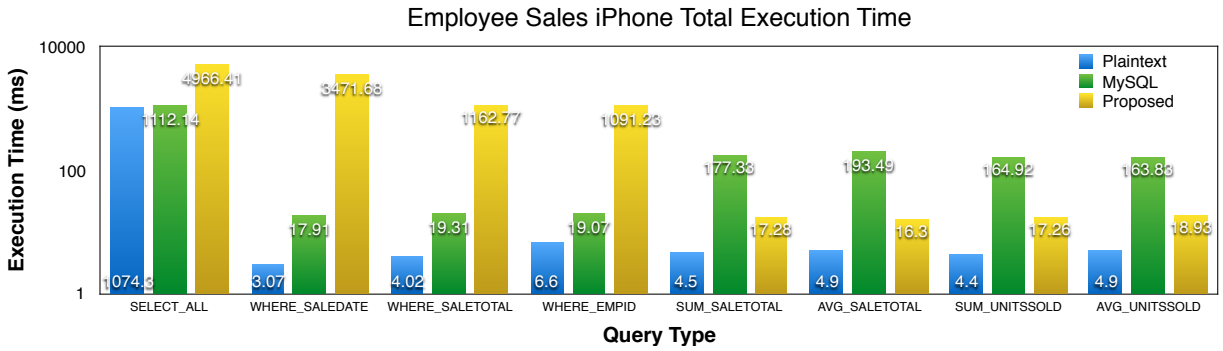

Figure 5.10: Execution time for queries executed on employee sales table on the iPhone

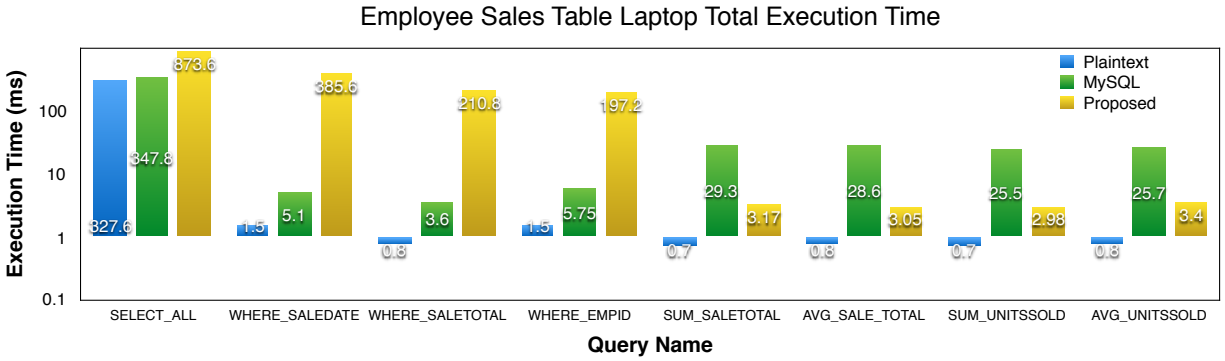

Figure 5.11: Execution time for queries executed on employee sales table on the laptop

A summary of some of the trends found in the employee sales table include:

- A four fold increase in the number of the number of records, from the item price table, has no effect on the execution time for the sum and average calculation in the proposed system
- A four fold increase in the number of the number of records, from the item price table, has approximately a three fold increase on the sum and average calculation in MySQL encrypted system
- For the proposed system, the cost of performing a select query with a where clause on a *string* value, is approximately $3x$ that of doing a select query matching on a *numeric* value for the iPhone, and $2x$ for the laptop

EmpSales Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	4966.373	2171.011	265.199	79.668	1079	1030.91
where_saleDate	3472.757	2022.67	233.102	0.000320	1017.883	1.03
where_saleTotal	1162.274	0.0861	0.00691	23.009	1012.01	1.01
where_empID	1091.1	4	1.883	15.033	973.6	6.7
sum_saleTotal	17.28	0	0	0.903×10^{-5}	14.79	0.11
avg_saleTotal	16.3	0	0	0.910×10^{-5}	14.09	0.15
sum_numUnitsSold	17.26	0	0	0.904×10^{-5}	14.84	0.12
avg_numUnitsSold	18.93	0	0	0.907×10^{-5}	16.38	0.14

Table 5.18: A breakdown of the employee sales execution time for the proposed system on the iPhone.

EmpSales Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	873.6	275.31	60.061	36.7	166.8	292.7
where_saleDate	385.6	175.95	28.9	18.2	160.9	0.2
where_saleTotal	210.8	361.2 x 10^{-6}	120 x 10^{-6}	9.5	163.4	0.21
where_empID	197.2	624.9 x 10^{-6}	27.1 x 10^{-6}	8.9	163.5	0.1
sum_saleTotal	3.17	0	0	0.198 x 10^{-6}	2.76	0.01
avg_saleTotal	3.05	0	0	0.208 x 10^{-6}	2.66	0.03
sum_numUnitsSold	2.98	0	0	0.210 x 10^{-6}	2.59	0.01
avg_numUnitsSold	3.4	0	0	0.201 x 10^{-6}	2.96	0.07

Table 5.19: A breakdown of the employee sales execution time for the proposed system on laptop.

Some of the information we can extract from the employee sales execution time break down include:

- For the laptop, the amount of time spent on encryption related tasks include: 42.6% for select all, 57.8% for a select with a string where clause, and 4.5% select with a numeric where clause
- For the iPhone, the amount of time spent on encryption related tasks include: 51% for select all, 65% for a select with a string where clause, and 1.46% select with a numeric clause
- For the laptop and iPhone, less than 1% of the time is spent on encryption related tasks for the sum and average queries, and the majority of the time is spent on getting the data from the server

Here we see that the average and sum calculations are fairly efficient, where most of the execution time is spent on retrieving the data, rather than on any decryption related operations. On the other hand, queries with a where clause, continue to get more expensive as the number of records increases, since most of the work is being done on the device, rather than on the MySQL server.

The next set of experiments, focus exclusively on the employee table, where the number of records start at ten thousand, and goes up to one million and two hundred and fifty

thousand records. The employee table contains three string values, and four numeric values. First, we will present the total execution time comparison for the various queries carried out on the employee table containing ten thousand records:

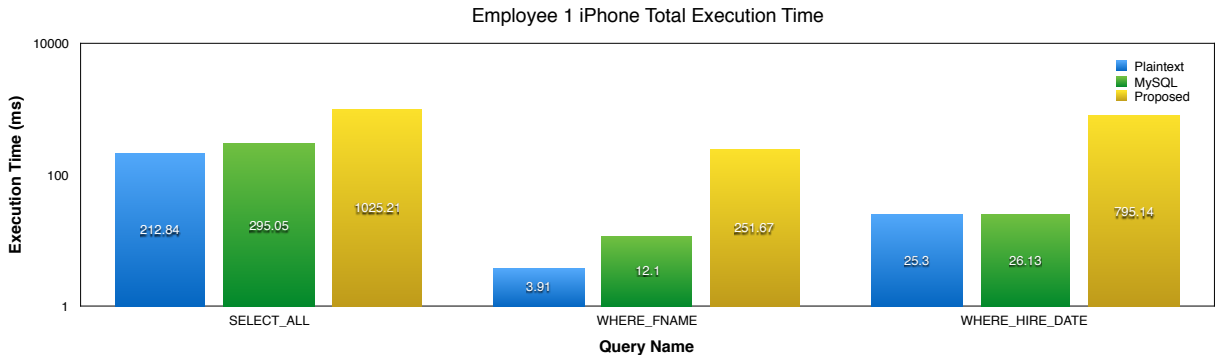

Figure 5.12: Execution time for the first set of queries executed on employee table containing ten thousand records on the iPhone

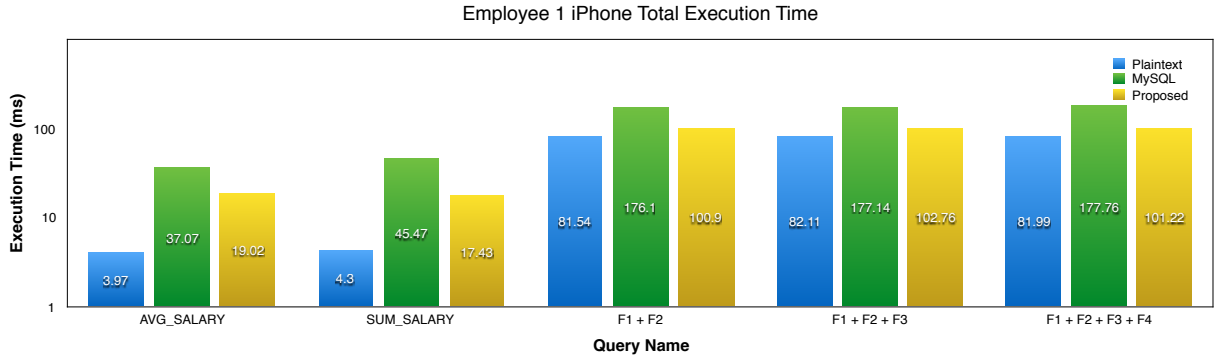

Figure 5.13: Execution time for the second set of queries executed on employee table containing ten thousand records on the iPhone

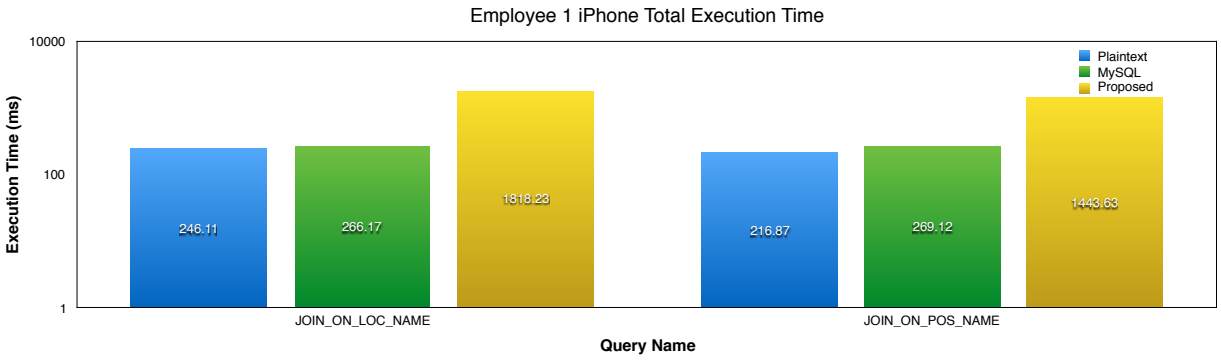

Figure 5.14: Execution time for the third set of queries executed on employee table containing ten thousand records on the iPhone

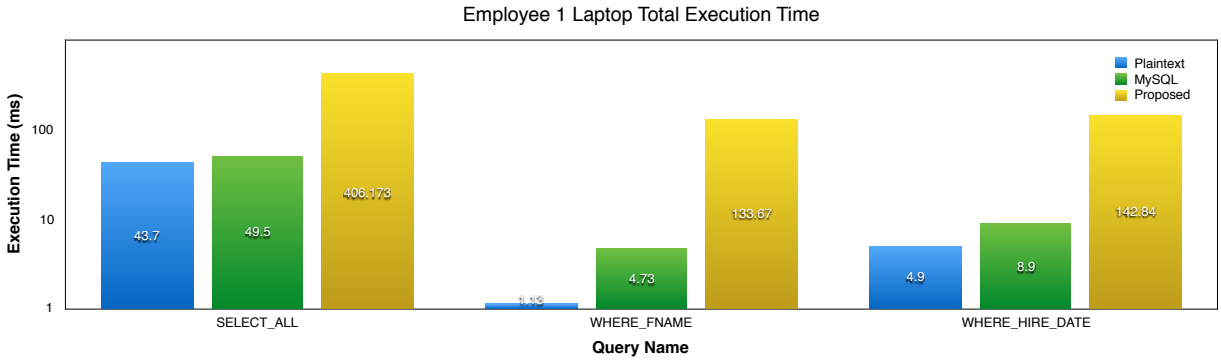

Figure 5.15: Execution time for the first set of queries executed on employee table containing ten thousand records on the laptop

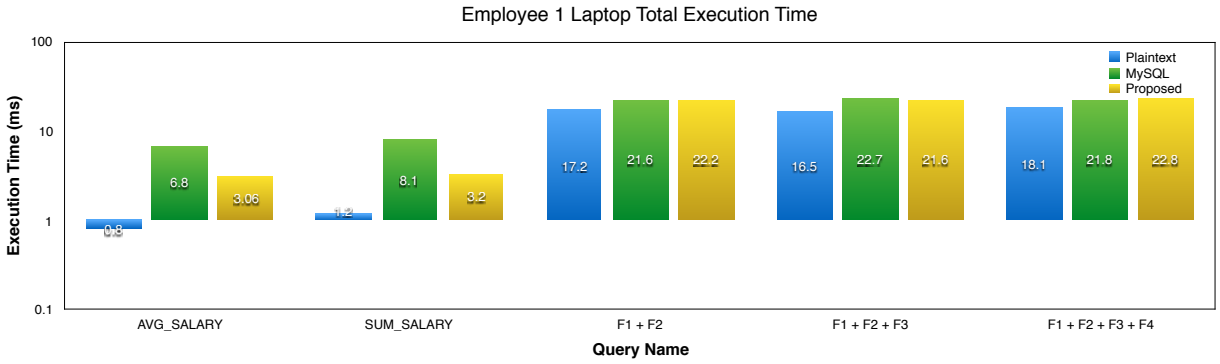

Figure 5.16: Execution time for the second set of queries executed on employee table containing ten thousand records on the laptop

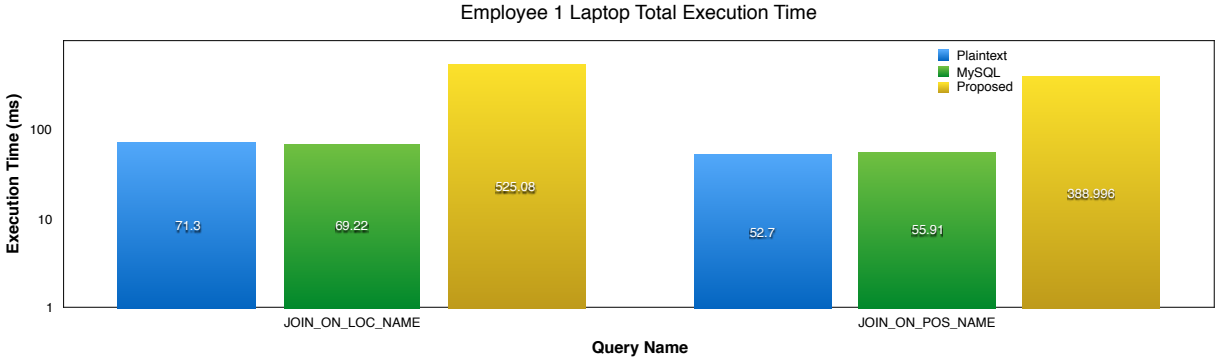

Figure 5.17: Execution time for the third set of queries executed on employee table containing ten thousand records on the laptop

Looking at the total execution time data, we see similar trends to the ones seen in previous tables:

- Select queries with a where clause are expensive, ranging from 31x to 64x
- Average and sum calculations are inexpensive
- The sum of multiple numeric fields are inexpensive
- Execution time for summing up numeric fields does not increase with increasing the number of fields

- Join queries are expensive, mainly because the data from the two tables are retrieved serially, and the actual join is performed on the device before the result is returned to the user

Now we look at a break down of the execution time of the proposed system for the employee table:

Employee Query Name	**Exec Time (ms)**	**AES Decryption (ms)**	**Key Gen (ms)**	**Shamir Decryption (ms)**	**MySQL Related (ms)**	**printf (ms)**
select_all	1025.29	581.332	70.61	1.912	162.08	112.04
where_fname	258.33	155.21	20.33	0.745×10^{-2}	59.67	0.22408
where_hireDate	795.09	434.90	73.78	0.3106	180.32	32.55
avg_salary	19.02	0	0	0.197×10^{-6}	16.45	0.11
sum_salary	17.43	0	0	0.201×10^{-6}	14.96	0.11
f1+f2	201.8	0	0	1.48	33.2	162.6
f1+f2+f3	205.8	0	0	1.34	32.8	166.8
f1+f2+f3+f4	214.12	0	0	1.39	31.2	185.3
join_locName	1818.401 (952.404)	484.22	61.101	1.991	29	120.08
join_posName	1443.41 (458.06)	558	93.2	2.15	18.92	45.12

Table 5.20: A breakdown of the execution time for the employee table containing ten thousand records for the proposed system on the iPhone. The time between bracket for the join queries, constitutes the total time spent on processing the location table and position table for join on location name and join on position name queries, respectively.

Employee Query Name	Exec Time (ms)	AES Decryption (ms)	Key Gen (ms)	Shamir Decryption (ms)	MySQL Related (ms)	printf (ms)
select_all	406.173	264.21	36.92	3.18	28.12	47.85
where_fname	133.67	70.33	19.102	0.897×10^{-3}	35.33	0.09566
where_hireDate	142.84	73.46	15.42	0.912	25.41	7.19
avg_salary	3.06	0	0	0.197×10^{-6}	2.80	0.02
sum_salary	3.2	0	0	0.201×10^{-6}	2.80	0.02
f1+f2	22.2	0	0	1.12	4.2	12.8
f1+f2+f3	21.6	0	0	1.16	6.8	12.94
f1+f2+f3+f4	22.8	0	0	0.96	5.2	16.09
join_locName	525.08 (165.213)	236.441	40.509	3.791	19	44.82
join_posName	388.996 (37.68)	222.43	39.1	4.65	18.92	35.41

Table 5.21: A breakdown of the execution time for the employee table containing ten thousand records for the proposed system on the laptop. The time between bracket for the join queries, constitutes the total time spent on processing the location table and position table for join on location name and join on position name queries, respectively.

Here we see that 75% of the total execution time for the iPhone, and 64% of the total execution time for the laptop, is spent on encryption related operations for the select all query. For the select with a where clause 64% of the total execution time for the iPhone and laptop, is spent on encryption related operations. Again for the average and sum calculations, the bulk of the execution time is spend on getting the data from the server. For the summation of numeric fields, less than 1% of the total execution time is spent on decryption related operations.

Next, we will look at various trends for doubling the number of records for each query from ten thousand, to one million and two hundred thousand records on the laptop, with similar trends observed for the iPhone. For the first three graphs (select all 5.18, select where 5.20, select join 5.19), the increase in execution time is almost linear, as the number of records doubles. Since half the fields are encrypted using AES, and around 67% - 70% of the time is spent on decryption related operations, it does not come as a surprise that the execution time doubles as the number of records doubles because the amount of data to retrieve, and process, doubles.

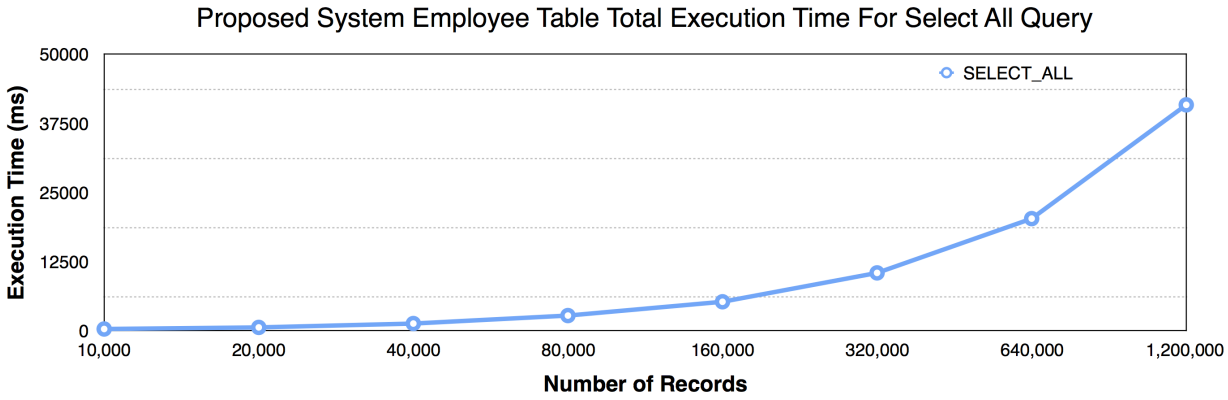

Figure 5.18: Select all execution time trend for doubling number of records in employee table running on the laptop for the proposed system

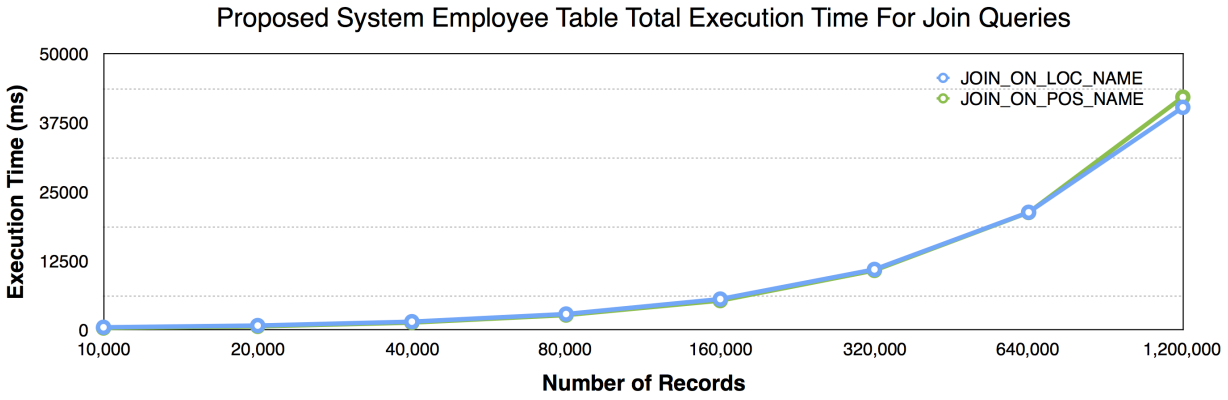

Figure 5.19: Join on location name and position name execution time trend for doubling number of records in employee table running on the laptop for the proposed system

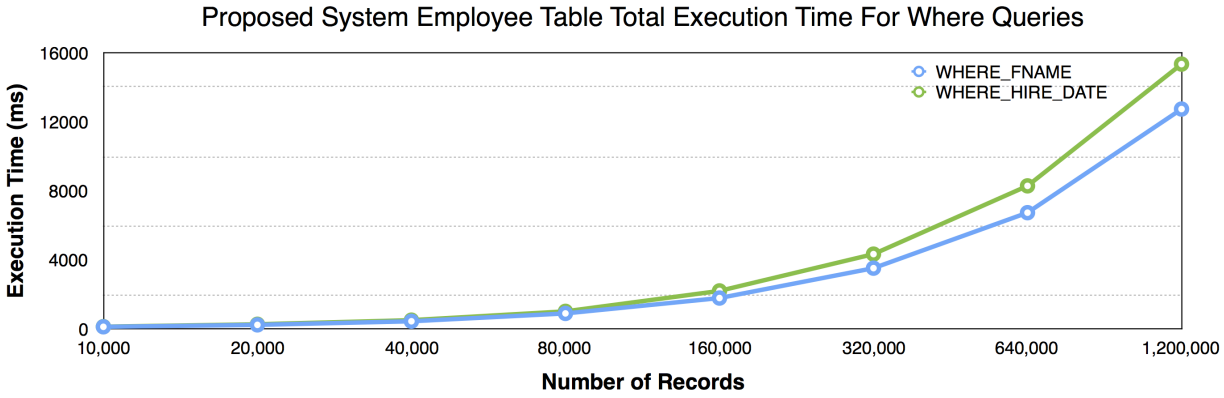

Figure 5.20: Select all with a where clause execution time trend for doubling number of records in employee table running on the laptop for the proposed system

The next trend of interest is the summation of numeric fields (5.21), where the doubling of the number of records causes a linear increase in execution time because the amount of data being retrieved and returned to the user from the server is doubling as well. The trend showing the effect of doubling the number of records on the execution time is shown below:

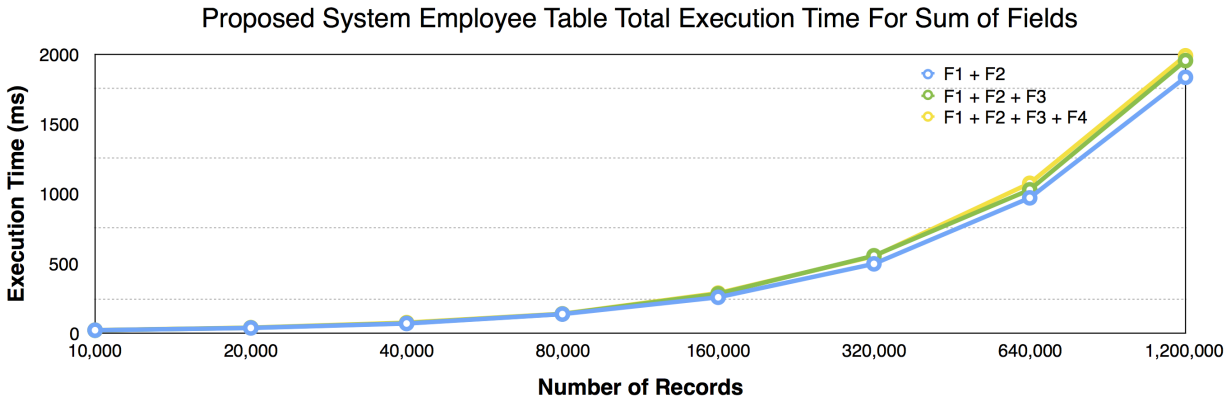

Figure 5.21: Sum of multiple fields execution time trend for doubling number of records in employee table running on the laptop for the proposed system

Finally, the last trend to be examined is the average and sum of a single field, which remains more or less constant with the doubling of the number of records.

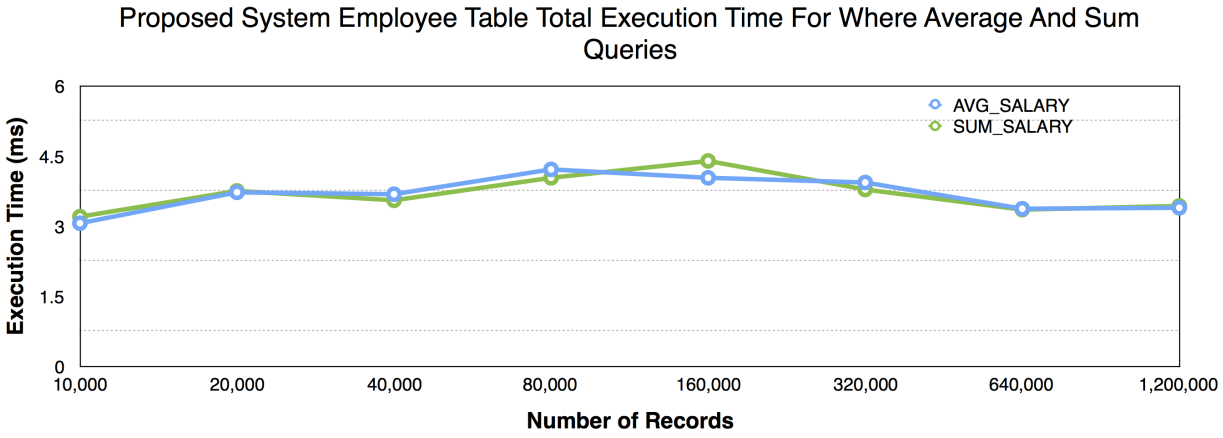

Figure 5.22: Sum and average execution time trend for doubling number of records in employee table running on the laptop for the proposed system

The results examined in this subsection give a picture about the cost in terms of execution time and memory usage associated with various types of queries, for tables with varying number of records, and varying field types. In the next section we summarize the cost findings in terms of memory usage and total execution time.

5.4 Potential Improvements

The implementation of this system in software involves tradeoffs that can be decided based on the requirements of the users. In this section, we will explore one potential improvement that directly affects the execution time - the use of multi-threaded programming. The MySQL server, is indeed multi-threaded, and the utilization of the multi threads will definitely improve the execution time[15]. Multi-threading could potentially negatively impact power consumption, through higher CPU utilization. To determine the effect of using multiple threads on power consumption, a multi-threaded version for the location table was developed. A GUI application by Intel, which provides real-time data on processor frequency and estimated processor power, and can log frequency, power, energy, and temperature data over time, was used to measure the power consumption during the execution of the query [16]. The implementation of the multithread version, had three threads: the

main thread, and two processing threads. The main thread retrieves six records at a time, and adds it to the queue of the first processing thread, and then retrieves another six records and adds them, to the queue of the second processing thread. A diagram depicting the operation of the multi-thread implementation, is shown below:

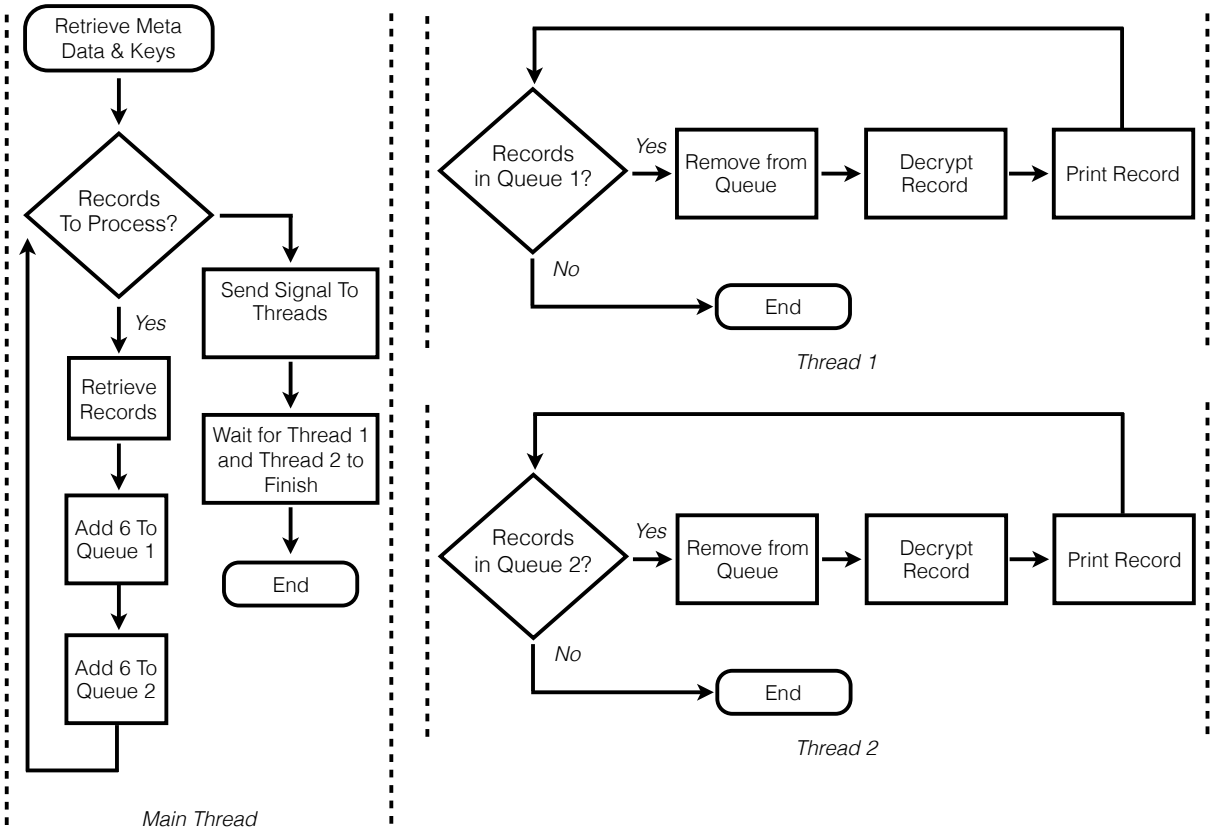

Figure 5.23: A diagram showing the multi-thread implementation for the location table in the proposed system

The processing queues, process any records that are in the queue, and block if the queue is empty. Once the main thread is done extracting all the records, a signal from the main thread is sent to the consumer threads informing them that they can finish anything in the queue and exit. In this case, in order to display the results in order, one of two things can be done: use an array with the primary key as the index to store the records, or store the results as they come in and sort them once all the processing is done. Since the same amount of memory is used, the only piece of information that has to be known before the

processing is know, is the number of records, in order to allocate the appropriate amount of memory. Tests of this implementation have been performed on the location table, and the results have shown average power consumption to be approximately 4% higher in the multi-threaded version, over the single-threaded version, but has a saving of approximately 34% in execution time. Therefore, the overall energy consumption of the multi-threaded version, is lower than the single-threaded version.

Another area of improvement, involves queries that retrieve all the records for processing queries with a where clause. One way to improve those types of queries, is to have a field that remains in plaintext form that can be used to reduce the results filtered for the queries with a where clause. This is similar to the concept of semi-structured data, where tags are used to enforce hierarchies of records and fields within a data set. So for example, if the employee hire date, happens to be a field that can remain in plaintext form, then performing a query on the employee table of employee working in Toronto, who were hired after August 2017, would reduce the dataset that has to be retrieved from the server to just the employees hired after the specified date, instead of getting all the employees determining the ones that work in Toronto, and then determining the ones hired after that specific date. This of course might not be possible for certain tables, but it certainly is something that can be considered. This of course, can reduce the security of the data, because you are exposing a relationship between the data sets - whether those relationships mean something or not, is a different discussion.

Chapter 6
Contributions, Conclusion, and Future Work

6.1 Contributions

In this work, we surveyed various work on homomorphic encryption schemes, and how data privacy can be achieved in databases. Next, we explored two different ways to combine Shamir's Secret Sharing Scheme with AES encryption, in order to *encrypt* the data at a field level, and analyzed each method to determine the more suitable one. Encrypting the database at a field level has two main advantages: no changes are required to the database itself, and the database can be hosted in the cloud without compromising the privacy of the data. We analyzed the advantages and disadvantages of the proposed system, to better understand the scenarios under which the system might be cost efficient for a client. The cost of the proposed system in terms of execution time, and bandwidth was determined for different query types.

Next, an architecture for the proposed system was introduced, and analyzed for potential security threats. Software implementation of the proposed system was completed in C and Objective C, so that it can be benchmarked against plaintext database and MySQL encrypted database - considered to be two viable alternatives. The software was benchmarked on a laptop and and iPhone, to collect empirical data for execution time and memory usage. Full details of how the software executing the various query types, as well as the tools used to perform the instrumentation of the code, was shown. A database containing five tables, and varying number of records and field types, was generated and uploaded to Amazon's RDS service paving the way for benchmarking the proposed system. Next, experiments were designed to perform the benchmarking of the proposed system against the plaintext and MySQL encrypted database, and the data collected from the experiments were analyzed. The experiment analysis provide empirical data on how execution time, bandwidth, and memory usage, changes as the number of fields, field types, and number of records returned are varied. Finally, some potential improvements of the software were suggested.

6.2 Conclusion

In this work, we have proposed a way to combine Shamir's Secret Sharing Scheme with AES encryption in order to encrypt a database at the field level. We accomplished this with no change to MySQL server, which allows for flexible deployment in the cloud on any database technology. Next, we discussed the advantages and disadvantages of the proposed system, and how it compares to the plaintext and MySQL encrypted databases. We also completed an implementation in C, and objective C, to demonstrate the practicality of the system with a database hosted in the Amazon's RDS cloud service. Using the implementation, we determined collected empirical data about the cost in terms of execution time, memory, and bandwidth usage. Finally, a potential improvement that can be applied to the implementation was discussed, that was able to reduce execution time at the cost of some added complexity and memory usage.

A summary of the important findings for the proposed system include:

- The use of better hardware, has a positive effect on the proposed system. The ratio spent on decryption related operations between the iPhone and laptop are the same, with just faster performance on the laptop
- Execution time for sum/average queries, does not change significantly as the number of records increases
- Queries matching on numeric values, are faster than matching on string values - $10x$ on average for the iPhone, and $17x$ on average for the laptop
- Performance of the average/sum queries is significantly better on the proposed system than on the MySQL encrypted database
- Select all queries relative to MySQL encrypted database, is on average $2x$ for the proposed system

In conclusion, the system has many advantages and disadvantages, and its usability is highly application dependent. The most efficient query types involve numeric values such as: sum and multiplication of two fields, sum and average of a single field, and sum of multiple fields calculations. Queries that involve where clauses or joins, are expensive, but remain reasonable in terms of execution time for relatively large number of records. While the bandwidth cost is high for query types that involve retrieving all the data, over time the trend of cheaper communication will slowly diminish this cost. Also improvements in CPU power, will decrease the execution, and thus improve the feasibility of the system.

6.3 Future Work

The future work revolves around two main areas: improving the performance of the proposed system, and determining some operational details of the proposed system. Improving the performance of the proposed system, to make it an attractive choice for different database sizes and field types. The biggest downside to the proposed system is the need to download all the records for certain query types, making it inefficient in terms of data bandwidth and execution time. One thing to explore is multi-threading the decryption process, which can help reduce the execution time of the query, but it does not help with the bandwidth. It is possible to relax the security requirements, in order to reduce the number of records retrieved, but that would be dependent on the client and whether they can tolerate the security downgrade.

Regarding the operational details of the proposed system, some of the unanswered questions that require further research include:

- How will the CSP stay in sync for updates/deletes, if one (or more) of them is/are down?
- How do you deal with field values changing maliciously, or due to transmission errors?
- How will the system perform with different database technologies?
- Can the CSP, or another entity, be involved in the query processing of currently inefficient queries, without compromising the privacy of the data?

Bibliography

[1] National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information. Technical report, NIST, 06 2003.

[2] Oracle Berkeley DB: Performance Metrics and Benchmarks. Technical report, Oracle Corporation, 08 2006.

[3] 3.2.1. system requirements. http://www.mysql.com/customers/, November 2014.

[4] Big data. http://en.wikipedia.org/wiki/Big_data, October 2014.

[5] Digital curation. http://en.wikipedia.org/wiki/Digital_curation, October 2014.

[6] Large database. http://www.scaledb.com/large-database.php, November 2014.

[7] Mysql customers. http://www.mysql.com/customers/, November 2014.

[8] Products & services. http://aws.amazon.com/products/, November 2014.

[9] ios manual pages - cccrypt. https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/man3/CCCrypt.3cc.html, Dec 2015.

[10] Mcrypt. http://mcrypt.sourceforge.net, Dec 2015.

[11] Mockaroo - random data generator. https://www.mockaroo.com, Dec 2015.

[12] Mysql 5.5 reference manual. https://dev.mysql.com/doc/refman/5.5/en/integer-types.html, 2015.

[13] Mysql 5.5 reference manual. https://dev.mysql.com/doc/refman/5.6/en/encryption-functions.html, 2015.

[14] Mysql 5.5 reference manual. http://dev.mysql.com/doc/refman/5.7/en/load-data.html, 2015.

[15] Mysql 5.5 reference manual. https://dev.mysql.com/doc/refman/5.5/en/faqs-general.html, 2015.

[16] Using the intel power gadget api on mac os x. https://software.intel.com/en-us/blogs/2012/12/13/using-the-intel-power-gadget-api-on-mac-os-x, March 2016.

[17] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005*, 2005.

[18] Gordon B. Agnew. Cryptography and system security. University Lecture, 2014.

[19] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.

[20] Niv Ahituv, Yeheskel Lapid, and Seev Neumann. Processing encrypted data. *Communications of the ACM*, 30(9):777–780, 1987.

[21] Amazon. Amazon ec2. http://aws.amazon.com/ec2/, 2014.

[22] Amazon. Amazon ec2. http://aws.amazon.com/rds/aurora/details/, 2014.

[23] Amazon. Amazon ec2 pricing. http://aws.amazon.com/ec2/pricing/, 2014.

[24] Amazon. Amazon rds pricing. http://aws.amazon.com/rds/pricing/, 2014.

[25] Amazon. Aws cloud security. https://aws.amazon.com/security/, 2015.

[26] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[27] Randolph C Barrows and Paul D Clayton. Privacy, confidentiality, and electronic medical records. *Journal of the American Medical Informatics Association*, 3(2):139–148, 1996.

[28] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 103–114. ACM, 2009.

[29] David C Blair and ME Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM*, 28(3):289–299, 1985.

[30] George Robert Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, page 313. IEEE Computer Society, 1979.

[31] Gang Chen, Ke Chen, and Jinxiang Dong. A database encryption scheme for enhanced security and easy sharing. In *Computer Supported Cooperative Work in Design, 2006. CSCWD'06. 10th International Conference on*, pages 1–6. IEEE, 2006.

[32] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[33] Privacy Protection Study Commission et al. Personal privacy in an information society. *Washington, DC: US Government Printing Office*, pages 305–310, 1977.

[34] Diamantino Costa, Tiago Rilho, and Henrique Madeira. Joint evaluation of performance and robustness of a cots dbms through fault-injection. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 251–260. IEEE, 2000.

[35] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in CryptologyEurocrypt 2002*, pages 45–64. Springer, 2002.

[36] George I Davida, David L Wells, and John B Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems (TODS)*, 6(2):312–328, 1981.

[37] George I Davida and Y. Yeh. Cryptographic relational algebra. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 111–116. IEEE, 1982.

[38] Dorothy E Denning. Field encryption and authentication. In *Advances in Cryptology*, pages 231–247. Springer, 1984.

[39] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 307–315. Springer, 1990.

[40] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[41] Yuval Elovici, Ronen Waisenberg, Erez Shmueli, and Ehud Gudes. A structure preserving database encryption scheme. In *Secure Data Management*, pages 28–40. Springer, 2004.

[42] Ulfar Erlingsson, V Benjamin Livshits, and Yinglian Xie. End-to-end web application security. In *HotOS*, 2007.

[43] Robert J Flynn and Anthony S Campasano. Data dependent keys for a selective encryption terminal. In *1978 Proceedings of the National Computer Conference*, page 1127. IEEE Computer Society, 1899.

[44] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for non-specialists. *EURASIP Journal on Information Security*, 2007, 2007.

[45] Amitabh Shah Francois Raab, Walt Kohler. Overview of the tpc-c benchmark. the order-entry benchmark. http://www.tpc.org/tpcc/detail.asp, 2014.

[46] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology–EUROCRYPT 2012*, pages 465–482. Springer, 2012.

[47] Deepak Gupta, Pankaj Jalote, and Gautam Barua. A formal framework for on-line software version change. *Software Engineering, IEEE Transactions on*, 22(2):120–131, 1996.

[48] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227. ACM, 2002.

[49] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 29–38. IEEE, 2002.

[50] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. *Guide to elliptic curve cryptography*. Springer, 2004.

[51] Jingmin He and Min Wang. Cryptography and relational database management systems. In *Database Engineering and Applications, 2001 International Symposium on.*, pages 273–284. IEEE, 2001.

[52] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[53] Murat Kantarcıolu and Chris Clifton. Security issues in querying encrypted data. In *Data and Applications Security XIX*, pages 325–337. Springer, 2005.

[54] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, 2013.

[55] Gary McGraw. Exploiting software: How to break code. In *Invited Talk, Usenix Security Symposium, San Diego*, 2004.

[56] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[57] Microsoft. Azure pricing: No upfront costs. pay for only what you use. http://azure.microsoft.com/en-us/pricing/, 2014.

[58] Microsoft. Sql database. a relational database-as-a-service that makes tier-1 capabilities easily accessible. http://azure.microsoft.com/en-us/services/sql-database/, 2014.

[59] Microsoft. Sql database pricing. a relational database-as-a-service with mission-critical capabilities. http://azure.microsoft.com/en-us/services/sql-database/, 2014.

[60] Microsoft. Virtual machines. launch windows server and linux in minutes. http://azure.microsoft.com/en-us/services/virtual-machines/, 2014.

[61] Raghunath Othayoth Nambiar, Matthew Lanken, Nicholas Wakou, Forrest Carman, and Michael Majdalany. Transaction processing performance council (tpc): Twenty years later–a look back, a look ahead. In *Performance Evaluation and Benchmarking*, pages 1–10. Springer, 2009.

[62] Gultekin Özsoyoglu, David A Singer, and Sun S Chung. Anti-tamper databases: Querying encrypted databases. In *DBSec*, pages 133–146. Citeseer, 2003.

[63] Gavin Powell. *Beginning database design*. John Wiley & Sons, 2006.

[64] Bart Preneel, Christof Paar, and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer, 2009.

[65] Dörte K Rappe. Homomorphic cryptosystems and their applications. 2005.

[66] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 2010.

[67] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[68] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[69] Samba Sesay, Zongkai Yang, Jingwen Chen, and Du Xu. A secure database encryption scheme. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pages 49–53. IEEE, 2005.

[70] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[71] Erez Shmueli, Ronen Vaisenberg, Yuval Elovici, and Chanan Glezer. Database encryption: an overview of contemporary challenges and design considerations. *ACM SIGMOD Record*, 38(3):29–34, 2010.

[72] Erez Shmueli, Ronen Waisenberg, Yuval Elovici, and Ehud Gudes. Designing secure indexes for encrypted databases. In *Data and Applications Security XIX*, pages 54–68. Springer, 2005.

[73] NIST-FIPS Standard. Announcing the advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197, 2001.

[74] Endre Süli and David F Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

[75] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of aes. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, pages 39–56. Springer International Publishing, 2015.

[76] Neal R Wagner, Paul Putter, and Marianne R Cain. Encrypted database design: Specialized approaches. In *IEEE Symposium on Security and Privacy*, pages 148–155, 1986.

[77] Yuanling Zhu and Kevin Lü. Performance analysis of web database systems. In *Database and Expert Systems Applications*, pages 805–814. Springer, 2000.

APPENDICES

.1 Relevant Definitions

This is a list of some basic definitions and concepts that are relevant in this work:

- **Plaintext:** The message, or text, in its original form.
- **Ciphertext:** The message after it has been encrypted using any encryption scheme.
- **Encryption:** The process of converting plaintext, into ciphertext. This process would allow only authorized parties to recover the plaintext (original message), using the secret key that is agreed upon between the authorized parties.
- **Decryption:** The process of recovering the plaintext from the ciphertext. A process that is done using the secret key that is agreed upon between the authorized parties.
- **Deterministic Encryption:** Given a fixed encryption key and plaintext, the encryption will always produce the same ciphertext[44].
- **Ciphertext-only attack:** The attacker has access to ciphertext of several messages, which the attacker analyzes to try and recover the plaintext, or deduce the secret key[68].
- **Chosen plaintext attack:** The attacker is able to chose plaintext, and get the corresponding ciphertext [64, 68]. The plaintext-ciphertext pair is then analyzed to try to recover the plaintext, or try and deduce the secret key.
- **Known-plaintext attack:** The attacker has access to ciphertext and the matching plaintext[68]. The attacker analyzes the plaintext-ciphertext pair, to try and recover the plaintext, or try and deduce the secret key[68].
- **Chosen-ciphertext attack:** The attacker can choose different ciphertext to be decrypted and has access to the decrypted plaintext [68]. Again, the goal is to try and deduce the secret key.

Encryption schemes evaluation criteria: The criteria upon which encryption schemes are evaluated on includes the following[56]:

- Level of Security: Measured in the number of operations required (using the best methods currently known) to defeat the intended objective. In other words, the effort required to recover the plaintext from the ciphertext, or recover the secret key.
- Performance: Efficiency of the encryption scheme when used in a particular way. For example, an encryption algorithm can be rated by the number of bits per second which it can encrypt.
- Ease of implementation: The difficulty of implementing the encryption scheme, in software or hardware environment.

.2 Employee Tables Info

Query	**# of Records Returned**	**Record Numbers**
select all	10,000	1− >10,000
select all where fname = "Gordon"	20	1− >20
select all where fname = "Abdel Rafi"	20	5,000− >5,020
select all where fname = "Test First Name 168"	20	9,981 − > 10,000
select all where hireDate = "2010-04-01"	834	14,15,38,...9999
select empPos + empLoc	10,000	1− >10,000
select empPos + empLoc + empSalary	10,000	1− >10,000
select empPos + empLoc + empSalary + empSin	10,000	1− >10,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 1: Information about the queries being executed for for the employee table with ten thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	20,000	1− >20,000
select all where fname = "Gordon"	40	1− >40
select all where fname = "Abdel Rafi"	40	10,001− >10,040
select all where fname = "Test First Name 168"	40	19,961 − > 20,000
select all where hireDate = "2010-04-01"	1,666	14,15,38,...19999
select empPos + empLoc	20,000	1− >20,000
select empPos + empLoc + empSalary	20,000	1− >20,000
select empPos + empLoc + empSalary + empSin	20,000	1− >20,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 2: Information about the queries being executed for for the employee table with twenty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	40,000	1− >40,000
select all where fname = "Gordon"	80	1− >80
select all where fname = "Abdel Rafi"	80	20,001− >20,080
select all where fname = "Test First Name 168"	80	39,981 − > 40,000
select all where hireDate = "2010-04-01"	3,334	14,15,38,...39999
select empPos + empLoc	40,000	1− >40,000
select empPos + empLoc + empSalary	40,000	1− >40,000
select empPos + empLoc + empSalary + empSin	40,000	1− >40,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 3: Information about the queries being executed for for the employee table with forty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	80,000	1− >80,000
select all where fname = "Gordon"	160	1− >160
select all where fname = "Abdel Rafi"	160	40,001− >40,160
select all where fname = "Test First Name 168"	160	79,841 − > 80,000
select all where hireDate = "2010-04-01"	6,666	14,15,38,...79999
select empPos + empLoc	80,000	1− >80,000
select empPos + empLoc + empSalary	80,000	1− >80,000
select empPos + empLoc + empSalary + empSin	80,000	1− >80,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 4: Information about the queries being executed for for the employee table with eighty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	160,000	1− >160,000
select all where fname = "Gordon"	160	1− >160
select all where fname = "Abdel Rafi"	160	40,001− >40,160
select all where fname = "Test First Name 168"	160	79,841 − > 80,000
select all where hireDate = "2010-04-01"	13,334	14,15,38,...159,999
select empPos + empLoc	160,000	1− >160,000
select empPos + empLoc + empSalary	160,000	1− >160,000
select empPos + empLoc + empSalary + empSin	160,000	1− >160,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 5: Information about the queries being executed for for the employee table with three hundred and twenty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	320,000	1− >320,000
select all where fname = "Gordon"	320	1− >320
select all where fname = "Abdel Rafi"	320	80,001− >80,320
select all where fname = "Test First Name 168"	320	159,681 − > 160,000
select all where hireDate = "2010-04-01"	21,332	14,15,38,...319,999
select empPos + empLoc	320,000	1− >320,000
select empPos + empLoc + empSalary	320,000	1− >320,000
select empPos + empLoc + empSalary + empSin	320,000	1− >320,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 6: Information about the queries being executed for for the employee table with three hundred and twenty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	640,000	1− >640,000
select all where fname = "Gordon"	712	1− >712
select all where fname = "Abdel Rafi"	712	178,001− >178,712
select all where fname = "Test First Name 168"	712	355,289 − > 356,000
select all where hireDate = "2010-04-01"	53,400	14,15,38,...639,999
select empPos + empLoc	640,000	1− >640,000
select empPos + empLoc + empSalary	640,000	1− >640,000
select empPos + empLoc + empSalary + empSin	640,000	1− >640,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 7: Information about the queries being executed for for the employee table with six hundred and forty thousand records.

Query	**# of Records Returned**	**Record Numbers**
select all	1,280,000	1− >1,280,000
select all where fname = "Gordon"	1,067	1− >1,067
select all where fname = "Abdel Rafi"	1,067	266,751− >267,817
select all where fname = "Test First Name 168"	1,067	532,434 − > 533,500
select all where hireDate = "2010-04-01"	21,332	14,15,38,...1,199,999
select empPos + empLoc	1,280,000	1− >1,280,000
select empPos + empLoc + empSalary	1,280,000	1− >1,280,000
select empPos + empLoc + empSalary + empSin	1,280,000	1− >1,280,000
select sum(empSalary)	1	N/A
select avg(empSalary)	1	N/A

Table 8: Information about the queries being executed for for the employee table with six hundred and forty thousand records.

.3 Script For Generating Data

```
def generate_db(num_records,list_fnames,list_lnames):

        conn = sqlite3.connect(num_records+".db")
        c = conn.cursor()

        list_salary = [100000,200000,300000,400000,500000,600000,
        700000,800000,900000,1000000]
        sin_counter = 11111
        salary_counter = 0
        emp_id_counter = 1
        pos_id_counter = 6

        dict_names = dict()

        for i in create_tbl_sql:
                c.execute(i)

        for i in pos_tbl_sql:
                c.execute(i)

        conn.commit()

        for f in list_fnames:
                for l in list_lnames:
                        if salary_counter > 9:
                                        salary_counter = 0
                        if pos_id_counter > 17:
                                        pos_id_counter = 6
                        s = "INSERT INTO employee VALUES ("
                        s = s + "'" + str(emp_id_counter) + "',"
                        s = s + "'" + f + "','" + l + "',"
                        s = s + "'" + str(pos_id_counter%17) + "',"
                        s = s + "'" + str(list_salary[salary_counter])
                        + "',"
                        s = s + "'" + str(sin_counter) + "')"
```

```
                        salary_counter = salary_counter + 1
                        dict_names[emp_id_counter] = s
                        emp_id_counter = emp_id_counter + 1
                        sin_counter = sin_counter + 1

        for i in dict_names.keys():
                #print dict_names[i]
                c.execute(dict_names[i])

        conn.commit()
        conn.close()

list_fnames_10 = ['Abdel','Mark','Mike','John','Joe']
list_lnames_10 = ['Tawakol','Doe']
generate_db('ten',list_fnames_10,list_lnames_10)

list_fnames_100 = ['Abdel','Mark','Mike','John','Joe','Jim','Nick'
,'Phil','Bill','George']
list_lnames_100 = ['Tawakol','Doe', 'King','Bush','Zuckerberg','Carry',
'Fowler','Gates','Nay','Li']
generate_db('hundred',list_fnames_100,list_lnames_100)
```

.4 Encrypt Integer Using Shamir Secret Sharing Scheme

```
signed long int fast_atoi( const char * str )
{
    signed long int val = 0;
    while( *str )
    {
       val = val*10 + (*str++ - '0');
    }
    return val;
}

signed int* encrypt_shamir(char* val)
{
    signed int *encrypted_vals = calloc(3,sizeof(signed int));
```

```
    signed int coeff1;
    int x_vals[] = {1,2,3};

    signed int int_val = (signed int)fast_atoi(val);

    coeff1 = ((arc4random_uniform(SHAMIR_PRIME-1))
              %SHAMIR_PRIME_POLY_COEFF);

    encrypted_vals[0] = ((int_val%SHAMIR_PRIME) +
                         ((coeff1*x_vals[0])%SHAMIR_PRIME)%SHAMIR_PRIME);
    encrypted_vals[1] = ((int_val%SHAMIR_PRIME) +
                         ((coeff1*x_vals[1])%SHAMIR_PRIME)%SHAMIR_PRIME);
    encrypted_vals[2] = ((int_val%SHAMIR_PRIME) +
                         ((coeff1*x_vals[2])%SHAMIR_PRIME)%SHAMIR_PRIME);

    return encrypted_vals;
}
```

.5 Table Information Sample

A sample of the information containing the table details, extracted from MySQL Workbench.

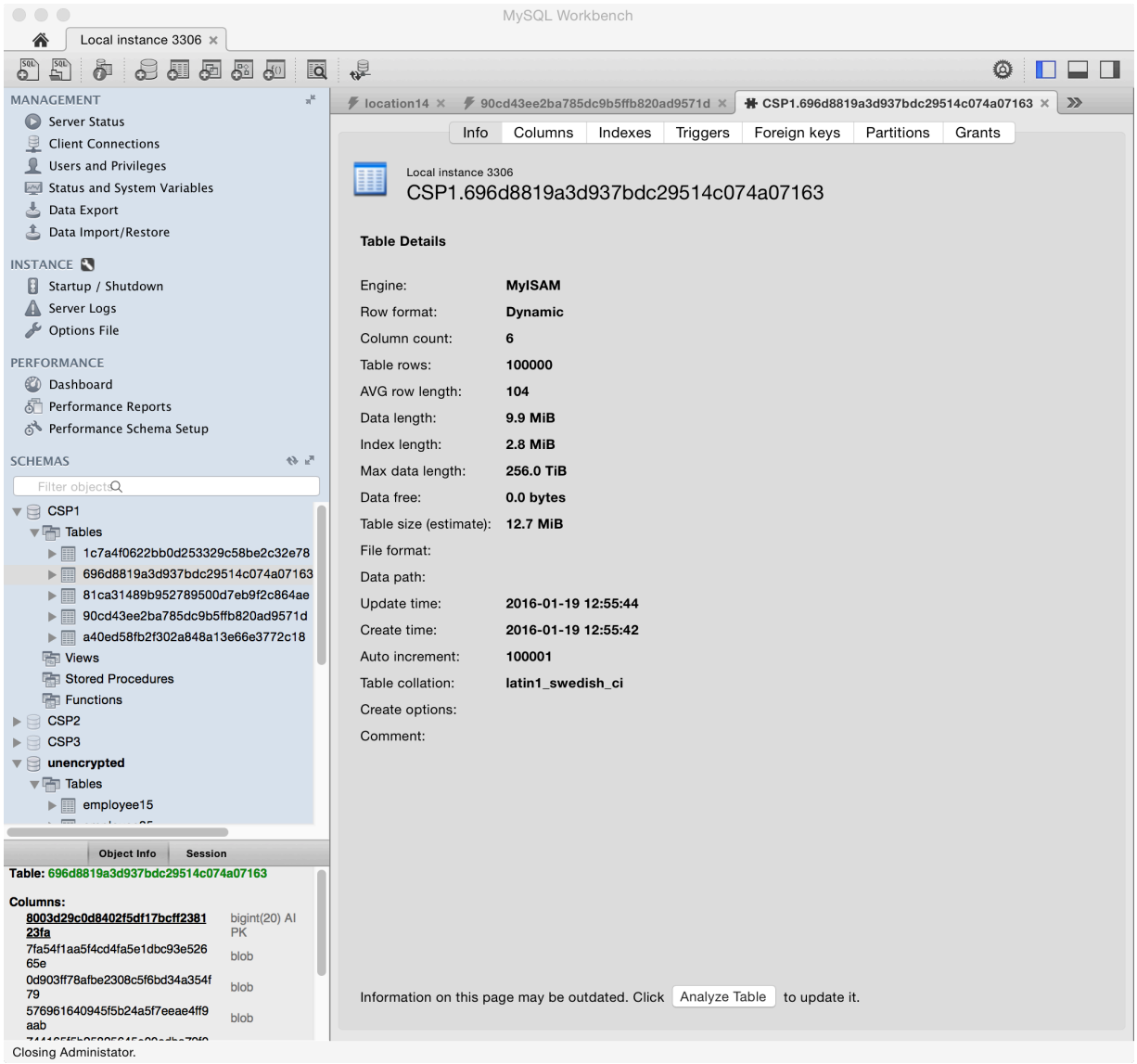

Figure 1: A sample of the employee sales table information, as seen on MySQL Workbench

.6 Trace Sample

A sample of the trace file recorded by Instruments, for item price table queries running on the iPhone.

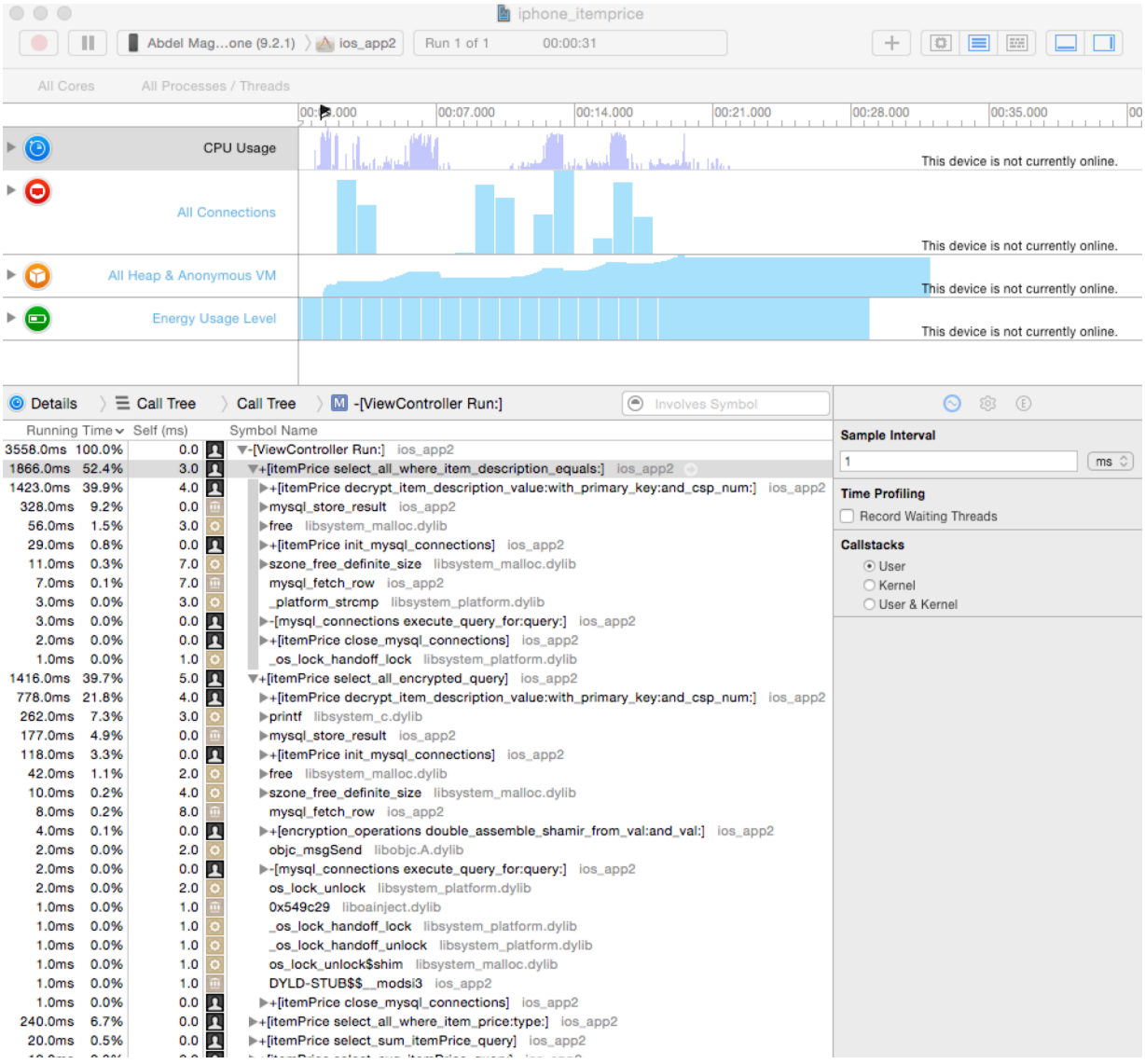

Figure 2: Sample trace file for item price table queries running on the iPhone