COMBINING MULTIPLE ENCRYPTION ALGORITHMS AND A DISTRIBUTED SYSTEM TO IMPROVE DATABASE SECURITY IN CLOUD COMPUTING

by

Amjad F. Alsirhani

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2014

DEDICATION

I dedicate this thesis to my late father, Faleh (1940-2011).

Three years have passed
I'll never forget the day
Someone rang to tell me
That you'd gone away
© Diana Doyle

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Cloud computing is a technology that facilitates the storing and managing of data in a decentralized manner. It includes a number of models and provides numerous services. It has many advantages and relatively few disadvantages, which makes the move to cloud computing quite attractive. However, since the data is out of the owner's control, concerns have arisen with regards to data confidentiality. Encryption techniques have previously been proposed to provide users with confidentiality in terms of outsource storage. These encryption algorithms allow for queries to be processed using encrypted data without decryption. However, a number of these encryption algorithms are weak, enabling adversaries to compromise data simply by compromising an algorithm. We propose a combination of encryption algorithms and a distribution system to improve database confidentiality. This scheme distributes the database across the clouds based on the level of security that is provided by the encryption algorithms utilized. A hybrid cloud model is used in this research, which is a combination of public and private clouds, with the critical activities taking place within the private cloud. We analyzed our scheme by designing and conducting an experiment and by comparing our scheme with existing solutions. The results demonstrate that our scheme offers a highly secure approach that provides users with data confidentiality. It also provides acceptable overhead performance and supports query processing.

LIST OF ABBREVIATIONS USED

AES	Advanced Encryption Standard
AES-CBC	Advanced Encryption Standard – Block cipher mode
AES-ECB	Advanced Encryption Standard – Electronic Code Book
ASPs	Application Service Providers
CCP	Cloud Computing Provider
DBaaS	Database as a Service
DET	Deterministic Encryption algorithm
HOM	Homomorphic Encryption
IaaS	Infrastructure as a Service
JSP	Java Server Page
OPE	Order-Preserving Encryption
PaaS	Platform as a Service
QoS	Quality of Service
SaaS	Software as a Service
SDA	Secure Distributed Approach
SQL	Structured Query Language
SSL	Secure Socket Layer
VPN	Virtual Private Network
WWW	World Wide Web

ACKNOWLEDGEMENTS

First of all, I would like to express my very great appreciation to my supervisors, Professor Srinivas Sampalli and Professor Peter Bodorik, for their guidance, encouragement and useful critique with regards to this research.

I would also like to offer my special thanks to my mother where no words can describe the gratitude I feel.

To my wife, Muram, who has always been there for me, and to my daughters, Hams, and Nagham and my son, Adi, for their patience while I spent much of my time pursuing my master's degree.

To my brothers, sisters and friends—thank you for your constant encouragement, without which this degree would not have been possible.

Lastly, I would like to acknowledge the financial support of Aljouf University for the scholarship granted to me for the duration of my master's degree.

CHAPTER 1: INTRODUCTION

1.1 Overview and Motivation

Cloud computing involves storing data using a third-party or non-central storage mechanism and having the ability to access this data from anywhere at any time. It offers a number of services and includes a variety of models to meet users' needs at affordable prices. Cloud computing providers have experts who operate the cloud so that users do not themselves need to be experts in order to obtain these services. Scalability is a cloud computing characteristic whereby data is scaled around the cloud provider's servers to guarantee high availability. The powerful devices that cloud providers rely on to operate the cloud give users high-performance processing, fast network speed, and a huge amount of storage space.

Despite the perceived benefits of cloud computing, many of which are discussed in this thesis, there are still major security concerns surrounding shifting data to the cloud. One of these concerns is adequately protecting the confidentiality of sensitive data. This ongoing and very important concern has motivated us to seek a way to improve security in a cloud computing environment and to create a viable real-life application. Gaining experience in the field of security and cloud computing is another of the author's motivations, along with becoming more involved in and contributing to computer science research.

1.2 Problem Statement

Cloud computing has many attractive advantages that encourage potential users to consider moving to a new style of computing. However, these advantages, as Hacıg (2005) claims, come at a high price, with users facing greater privacy risks and increased vulnerabilities when they move their (often private) data to a cloud. More importantly, this data is often not under the direct control of the owner, so concerns arise regarding data confidentiality. Under these circumstances, providers are able to compromise the data and access sensitive data, which constitutes an invasion of the database owner's privacy. Anciaux et al. (2007) point out that finding trusted providers to store sensitive data is not an easy task, as many cloud computing providers may not even trust their own employees. Additionally, in some cases, as Weippl

(2012) has stated, providers also reserve the right to change their terms and conditions, which means that data confidentiality and privacy risks are even more critical to consider.

In light of these ongoing security-related issues, encryption emerges as the simplest solution, as encrypting data before sending it to the cloud can prevent providers from obtaining sensitive information. Unfortunately, however, encrypted data cannot be easily queried, making it difficult for users to access their data. Some proposed solutions to this dilemma involve using asymmetric cryptography, with the private key being shared with the providers. However, the provider can still infer sensitive information with this method when they perform the decryption in response to the client's query. Similarly, symmetric key cryptography involves decryption on the provider's side, allowing providers to infer sensitive information in this case as well. Hence, neither symmetric nor asymmetric cryptography offer a suitable solution.

Since there is no single encryption algorithm that is able to support all Structured Query Language (SQL) queries without decryption, there is a need for a system that provides users with security while also supporting a variety of query structures. This leads us to the following questions: How can we guarantee data confidentiality while using untrusted cloud computing provider resources, and what encryption algorithms can support a variety of queries?

1.3 Research Objectives

The proposed scheme in this thesis aims to meet the following objectives:

1. Provide cloud computing users with confidentiality by preventing untrusted providers from obtaining meaningful and sensitive information.
2. Develop a cloud computing system that supports a variety of queries that can process encrypted data.
3. Develop a cloud computing system that provides an acceptable performance overhead.

1.4 Outline

The thesis is organized as follows: Chapter 2 covers the background of this study as well as the literature review. In Chapter 3, the proposal design is presented, followed by the implementation of the prototype and evaluation in Chapter 4. Finally, the conclusion and ideas for future research are presented in Chapter 5.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1. Background

2.1.1 Overview:

Cloud computing is an emerging technology that is changing computing styles. Hacıg and Li (2002) define the cloud as a concept whereby data and programs can be stored in a decentralized manner (i.e., in a cloud) and accessed at anytime, anywhere, through thin clients and lightweight mobile devices. Hore et al. (2004) summarize cloud computing's advantages as including on-demand self-service, ubiquitous network access, location-independent resource pooling, rapid resource elasticity, usage-based pricing, and transference of risk.

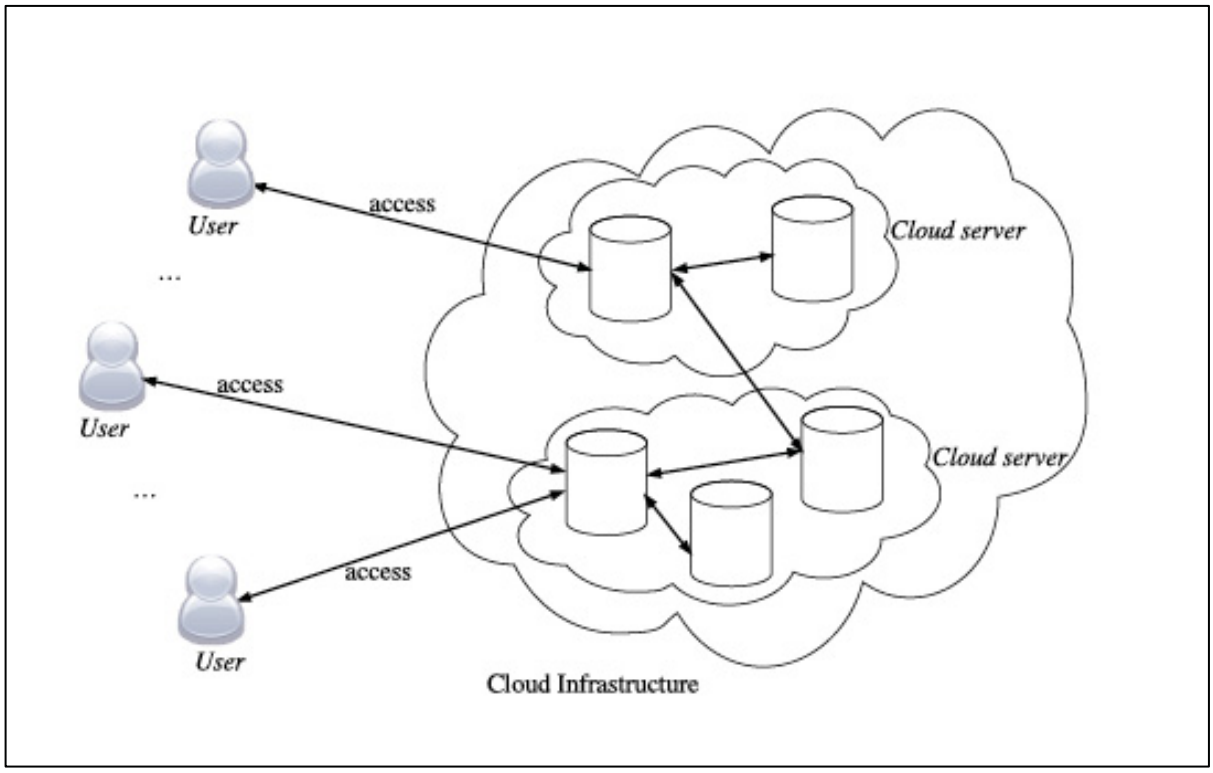


Figure 2-1: Cloud computing architecture
(Adapted from Capitani, Foresti, Samarati and Informatica [2012])

Additionally, cloud providers, as Hacıg and Li (2002) state, may be able to afford to invest in better and more up-to-date security technologies and practices than data owners. Thus, cloud resources are scalable over the Internet network, through which the service is provided on

demand to the users. There is also flexibility in cloud computing, as users can access data from anywhere, such as from a web page interface Furthermore, in addition to scalable resources that can be easily configured, the cloud providers also offer clients guaranteed quality of service (QoS). All of these advantages, combined with an affordable pricing scheme, make the cloud technology very attractive.

2.1.2 Cloud Computing Architecture

The architecture of cloud computing mainly consists of models and services that make the cloud feasible and accessible. In the next two subsections, further details on these models and services are provided.

2.1.2.1 Cloud Computing Models

Cloud computing models provide an idea of how the clouds are accessed and point to where the data centers are located. Sakhi (2012) summarizes cloud computing models into four types: public cloud, private cloud, community cloud, and hybrid cloud. In a public cloud, data centers are located anywhere around the world and are accessed by the public via the Internet. This type of cloud is less secure than other models because it is open to the general public. In a private cloud, data centers are located at a user's location and are accessed only by authorized personnel. An Internet secure channel (e.g., a virtual private network [VPN] or Secure Sockets Layer [SSL]) is usually used to facilitate communications between clients and the cloud. A third model is the community cloud, where a number of private clouds are integrated with each other. Finally, the hybrid cloud involves both public and private clouds that are connected to each other. In this hybrid model, most of the critical activity takes place in the private cloud, for security reasons.

2.1.2.2 Cloud Computing Services

Cloud computing provides a number of services. Figure 1 shows that these services include but are not limited to the following:

Infrastructure as a Service (IaaS)

Platform as a Service (PaaS)

Software as a Service (SaaS)

Database as a Service (DBaaS)

These services arise in-between the back end and the front end, as Nalinipriya and Aswin Kumar (2013) describe. The service follows a bottom-up approach in which the lower layer provides users with computation power and memory in a virtual environment. In the middle, the service layers deliver services such as offering a framework for application development and software outsourcing. A number of studies state that these services eliminate the need for software maintenance or licensing (Bouganim & Pucheral, 2002; Hore et al., 2011). End users, therefore, can access software running at a remote site through Application Service Providers

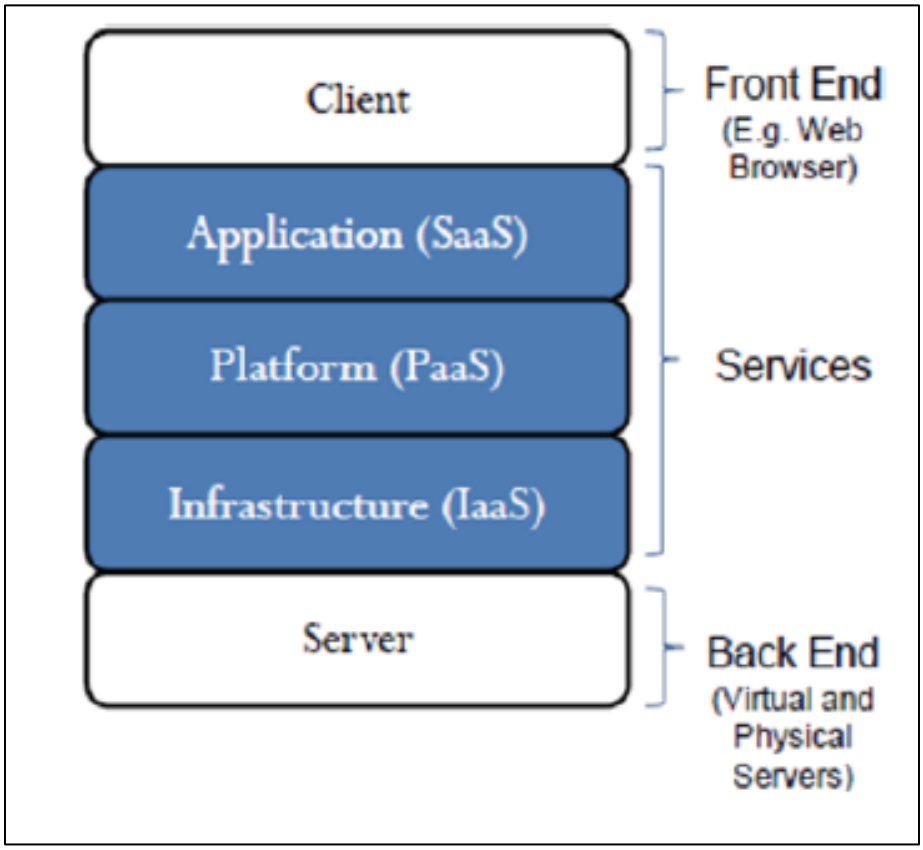


Figure 2-2: Service layers provided by cloud computing
(Adapted from Modi et al. [2012])

(ASPs) and avoid software installation and expensive software license fees. Users instead pay only for the services they use at the cloud. A simple "pay as you go" model is followed, which, as stated by Hore et al. (2011), allows users to pay only for the services they use. As our research concerns DBaaS security, the next section will discuss DBaaS services in detail.

2.1.2.2.1 Database as a Service (DBaaS)

The database is provided as a service by cloud computing operators. Sakhi (2012) describes the process of creating databases in the cloud. There are two ways to create databases involved in cloud computing: by using a virtual machine image, and by accessing the DBaaS. Unlike traditional databases, the provider takes full responsibility for managing and operating the database. Compared to traditional database systems, the DaaS provides QoS in terms of scalability, availability, multi-tenancy, and a resource allocation mechanism. Users require only an interface to access the DaaS. Hence, there is no need to hire database experts or administrators, as is necessitated by traditional database systems. DBaaS has more advantages compared to traditional databases. Take performance as an example: when using a high-powered computation (e.g., CPU, memory capacity and storage space), there is a potential increase in the performance. However, the challenge arises with regards to DBaaS-related security risks, especially if the data that needs to be stored is sensitive information.

2.1.3 Cloud Computing Security

Since communication in most cloud computing models is established via the Internet, any Internet security risk can apply to the DBaaS. Although cloud computing has many advantages, there are still a number of vulnerabilities, including vulnerabilities in Internet protocol, unauthorized access to management interface, injection vulnerabilities, and vulnerabilities in browsers and APIs. Furthermore, Hore et al. (2011) state that there are a number of threats in cloud computing, including changes to business models, abuse of cloud computing, insecure interfaces and APIs, malicious insiders, shared technology issues arising from multi-tenancy, data loss and leakage, and service hijacking. Moreover, as Modi et al. (2012) point out, possible attacks in cloud computing include zombie attacks, service injection attacks, attacks on virtualization, man-in-the-middle attacks, metadata spoofing attacks, phishing attacks, and backdoor channel attacks. A number of these potential attacks have been considered during the building of web applications, whereas others are still under investigation. Our scheme only considers the insider attack (i.e., an attack by the cloud computing provider).

2.2 Related Work

In this section, related work is presented in subsections. Work regarding the confidentiality of data stored through outsourcing is categorized into four groups.

2.2.1 The Bucketization Approach

Hacıg and Li (2002) propose a fragmentation scheme, which is a column-based partition. Figure 2-3 shows the architecture of their scheme, where, on the server side, there are only encrypted fragments (i.e., vertical fragmentation).

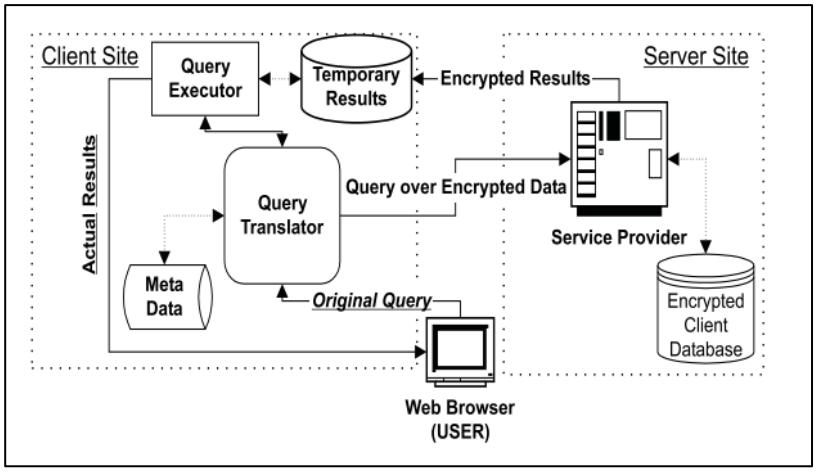


Figure 2-3: The architecture of the bucketization approach (Adapted from Hacıg and Li [2002])

Each fragment has a unique id used to support the query. Queries are processed twice. The first query fetches the whole fragment to the client, based on the fragment's id. Once the client obtains the fragment, it will be decrypted to apply the query again on top of the fetched fragment. Finally, the result will be returned after the second query is executed. While this approach could be suitable for small data sets, it requires a considerable amount of overhead time to fetch the whole fragments. In this approach, there is a case where the whole database has to be fetched to the client side, which raises questions around the benefit of using cloud computing. Additionally, processing the query twice slows overall performance. Hence, this approach is not a suitable solution for this problem because it eliminates cloud computing's main benefit in terms of providing storage. It also requires a lot of overhead time to execute the query.

Other studies have considered this approach (Hore et al., 2011; Hacıg, 2005; Hore et al., n.d.). They mainly focus on the query optimization technique as a means of trying to tackle the Hacıg and Li (2002) limitation, which concerns performance.

2.2.2 Chip-Secured Data Access Approach

Bouganim and Pucheral (2002) propose a hardware-software solution for the database outsource confidentiality issue, claiming that no software solution is guaranteed, given Internet security variability. Their idea is basically to install a smartcard on the cloud side that will work as a mediator. The data is then encrypted by the smartcard before being inserted into the

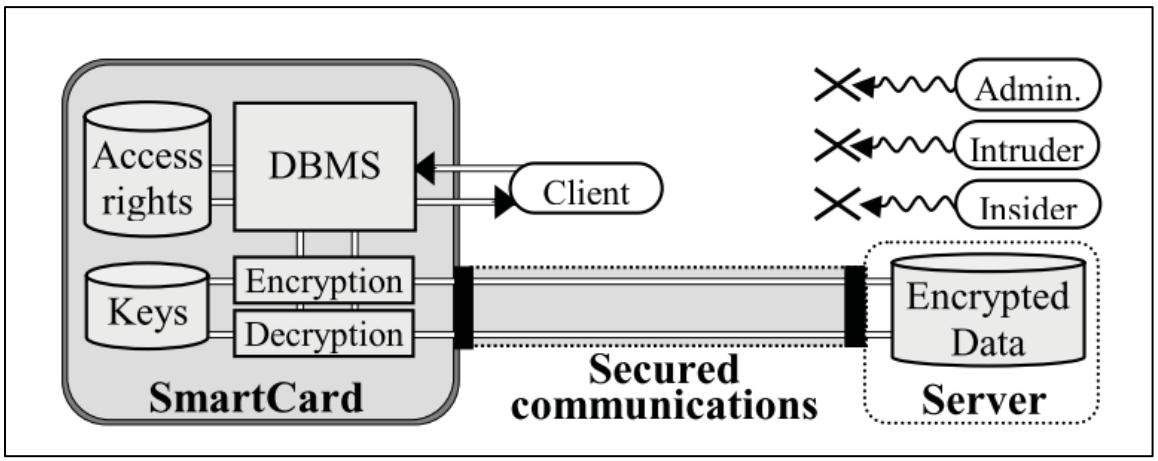


Figure 2-4: The architecture of the chip-secured data access approach (Adapted from Bouganim and Pucheral [2002])

database, and then decrypted before being sent to the user. This method is premised on an assumption of secure communication between the user and the cloud.

The smart card is fully controlled by the client, so the cloud is simply used as encrypted data storage. This approach has limitations in terms of processing power as well as memory capacity and thus does not offer a practical solution for the outsourcing of confidential data due to the weakness of the smartcard and the difficulty of installing the card into the cloud provider's server. The situation worsens if the database system needs to be distributed.

2.2.3 The ChostDB Approach

Anciaux et al. (2007) propose a different approach to ensure data confidentiality when data are stored with untrusted cloud providers. They divide the database into private and public data. The private data is the sensitive data that needs to be kept in a safe place, whereas the

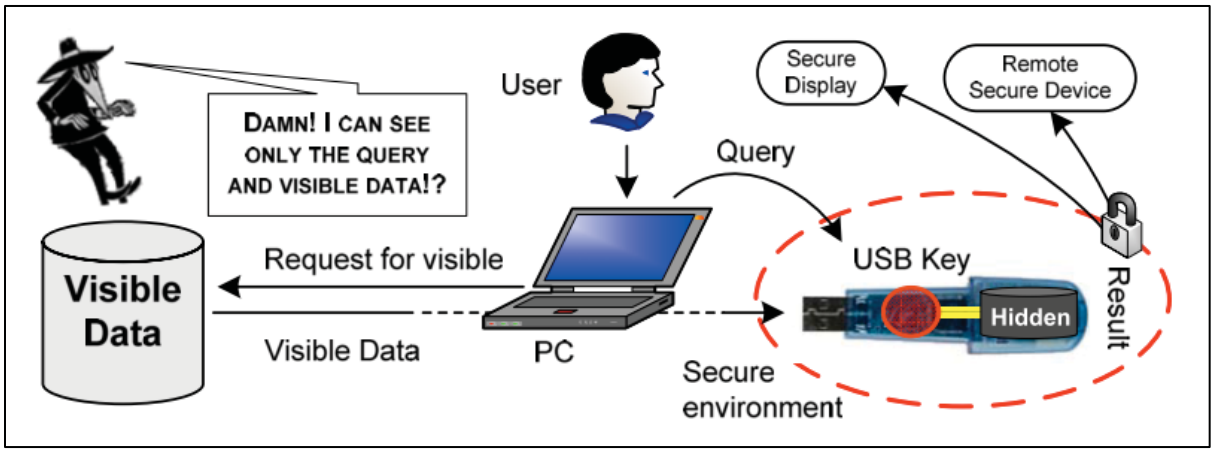


Figure 2-5: The architecture of the ChostDB approach (Adapted from Anciaux et al. [2007])

public data is the non-sensitive data that can be seen by the general public. They encrypt and store the sensitive information on a smart USB key, which has to be on the client side. The insensitive data are stored on a public cloud server in plain text. The user cannot query the data unless the smart USB key is plugged into the user's device. When the smart USB key is plugged in, the two partitions are joined using a distributed technique. This approach could be a solution for one user, but is not really feasible for widespread usage due to the difficulty of distributing the USB keys. Moreover, this approach limits the benefits of cloud computing by storing private data with the client (using a smart USB key). It key also eliminates the scalability of the system. Therefore, this approach cannot be used in practice.

2.2.4 CryptDB Approach

Popa et al. (2012) recently proposed a practical solution to improve the confidentiality level for outsourced data from curious cloud providers. Their scheme consists of a number of components, including encryption algorithms, proxy, and a users' application. This concept relies on the fact that no single encryption algorithm can support all types of queries, so the researchers investigated encryption algorithms that allow for queries to be issued to encrypted data. They came up with six encryption algorithms that can be used to support the fundamental query structure. These algorithms are: RAN, DET, OPE, JOIN, HOM and Search, which are already proposed by (Desai, 2000; Daemen, 2002; Liu et al. 2009; Popa et al, 2012; Stern & Dijk et al., 2010; Wagner & Perrig, 2000), respectively. However, Popa et al. (2012) models them in such way that the keys are not shared with cloud providers.

In RAN, there is no query that can be supported by this encryption algorithm. It is used

Employees ***Table1***

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

Figure 2-6: Sample of table of CryptDB approach (Adapted from Popa et al. [2012])

to randomize the data so that the pattern can be hidden. The next encryption algorithm is DET, which supports the equality query because it is a deterministic encryption algorithm. In deterministic algorithms, if a plain text is $P_1 = P_2$, then the cipher-text is $E_k(P_1) = E_k(P_2)$. Thus, this encryption algorithm can be used to support equality predicate. However, the DET encryption algorithm is weak on revealing the equality. Popa et al. (2012) state that the DET encryption algorithm is one of the weak encryption algorithms they have included in their scheme. The adversary can infer the key if she/he has the plain text as well as its cipher text.

The third encryption algorithm is OPE, where the comparison is supported. This encryption preserves the order of encrypted values using a given key. The encryption has the property such that: if plain text $P_1 > P_2$, then the encryption $E_k(P_1) > E_k(P_2)$ with a given key. This encryption algorithm will support any comparison query, but the algorithm reveals

the order of encrypted values and thus also the maximum and minimum of encrypted values. DET and OPE algorithms are the weakest encryption algorithms used in this approach.

The fourth encryption algorithm is JOIN. This encryption algorithm is needed to support equality between columns. It works basically the same as DET, but Popa et al. (2012) cannot use DET to join columns. JOIN is deterministic encryption that supports equality and has been used for the purpose of joining columns.

The fifth encryption algorithm is HOM, which has a general reputation for being highly secure. It is used to support different kinds of arithmetic queries, such as summation. In HOM, the summation of two plain texts is the product of their encryption. This property allows performing the summation and determining the average of the number of items over which the summation is performed. The encryption does not reveal any information, which means that it is secure enough to encrypt sensitive data that is outsourced for storage.

The sixth encryption algorithm is Search. Due to the fact that most web applications require a word match query for which the researchers found that Search is the best choice. The Search encryption algorithm is used in this approach to support word match queries.

Thus different encryption algorithms are applied to each column based on the column's data type and also based on the desired queries to be applied on column: the same data in each column may be encrypted more than once using different encryption algorithms. Each encryption algorithm is applied to the base column and the result of the encryption is stored in a new column. In this way, the number of columns is extended; hence, this encryption process is referred to as *extending* the base column by using different encryption algorithms. Each base (plain text) column is extended based on the number of encryption algorithms to be used for that column, where each of the extended columns containing encrypted values supports a different comparison method used in different queries. All of these columns are then stored at the DBaaS, with the assumption that the provider is not trusted to hold sensitive data.

The proxy also plays an important role in their scheme. In fact, most of the critical activities are performed at the proxy server. The proxy is a trusted server that runs in the middle between the public cloud and the user application. All communication between users and the proxy are assumed to move through a secure channel. When a user issues a query to the proxy, the proxy will transform it into a query on extended columns and encrypt it before sending it to the cloud. The proxy receives encrypted results, decrypts them, and and sends them, over a

secure channel, as a query reply to the user with the plain text result. There is no shared key with the cloud, so the encryption and decryption key is stored at the proxy server. Although the proxy is a single point of failure, it is located within the private cloud so that a distributes proxy system within the private cloud - we target the single point of failure issue as future research.

CHAPTER 3: THE PROPOSED DESIGN

In this chapter, the design utilized in the thesis is presented in detail. The chapter is organized as follows: it begins with the architecture of our scheme, which is then followed by the methodology. The methodology is divided into two sections: a subsection for encryption algorithms, and a subsection for the fragmentation technique.

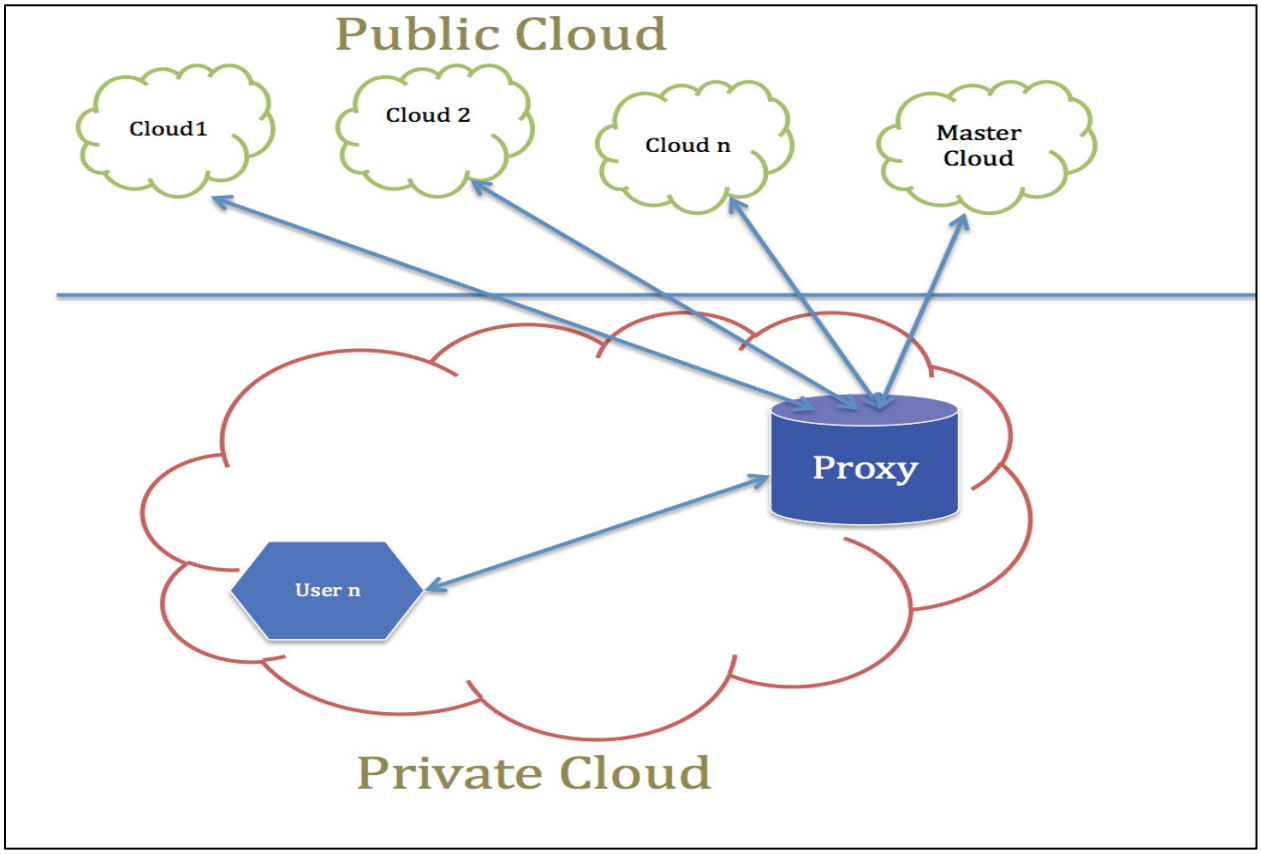


Figure 3-1: The architecture of our scheme

3.1 The Architecture

Our architecture is a hybrid cloud that mainly consists of two parts: public clouds and a private cloud. Based on our fragmentation technique, the encrypted database is only stored on public clouds. Figure 3-1 represents the architecture of our scheme. The public clouds consist of a master cloud and a number of slave clouds. The master cloud, stores an encrypted replica of the entire database while individual public clouds store extended columns. We assume that the public clouds are operated by different cloud providers. In fact, the providers of public clouds should not even know that there are fragments of the database stored on other providers clouds. The idea is to improve security by hiding this information among the clouds. Of course, this assumption causes problems with scalability in terms of the number of columns as each of the columns should be stored on different cloud offered by a different provider and that the providers do not collaborate. When we run out of cloud providers, we can start storing more than one column at each cloud as long as no two columns are encrypted using the same encryption algorithm, or at least not using the same keys if the same encryption algorithm is used. We acknowledge this scalability problem and target it for future research. More detail with regards to this point is provided in the security analysis section.

On the other hand, private clouds consist of a proxy and user's devices. The proxy is located at the center of communication, between the client and the public clouds. The proxy is a server that works as an agency. Figure 3-2 shows an overview of the architecture. There is no immediate communication between users and public clouds. Instead, the proxy translates any received query from the users to the cloud.

When the proxy receives the results from the master cloud, it decrypts them and responds to the user with the final result. The proxy performs all of the encryption and decryption. One of the more significant roles of the proxy is to act as the query parser. In query processing, the proxy has to decide which of the encryption algorithms to use, and which clouds to query.

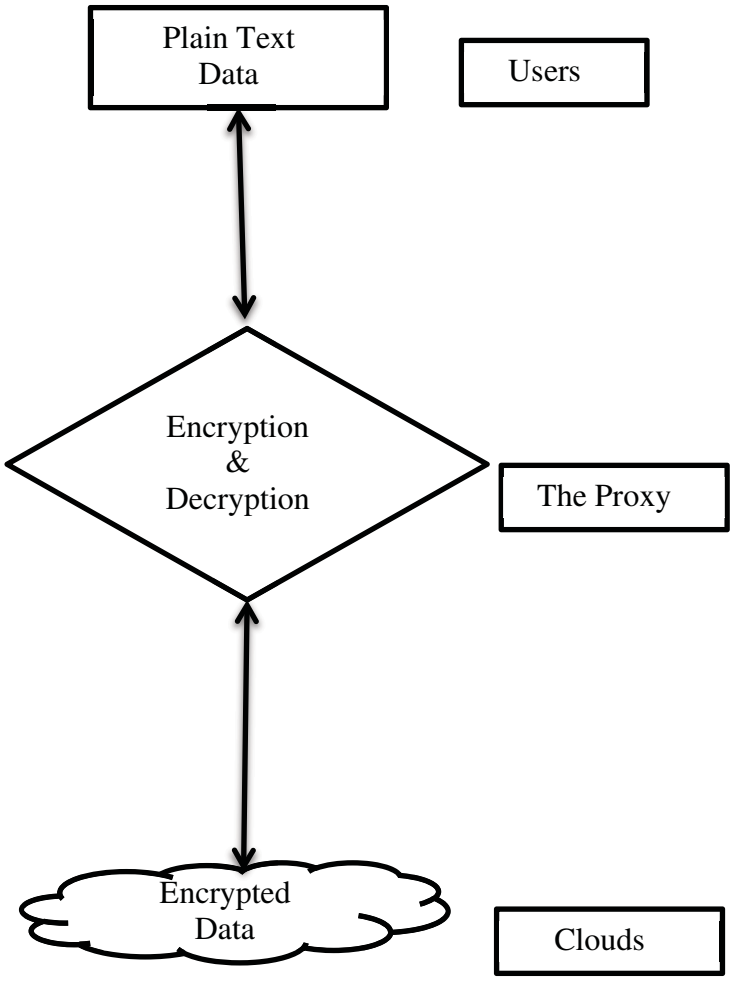


Figure 3-2: Overview of our scheme

3.2 Methodology

The proposed scheme is a combination of encryption algorithms and a fragmentation technique, the latter, which is a novel contribution to the research.

Encryption algorithms have already been proposed with a variety of properties, but to the best our knowledge, the scheme proposed by Popa et al. (2012) is the only approach that uses these algorithms together to provide confidentiality for outsourced data storage. We used these encryption algorithms in our scheme as well to encrypt the fragments. Furthermore, because it has been proven that AES-CBC is a highly secure encryption algorithm, we utilized it in our scheme for the master cloud database. Of course, other encryption algorithms are also candidates for use. The AES-CBC master cloud is proposed in this scheme to provide the user

with increased confidentiality. In section 3.2.2.1, additional details are provided on the master cloud.

The fragmentation technique is the truly novel aspect of our scheme. In comparison with the work conducted by Popa et al. (2012), our work is significantly advanced. Whereas Popa and colleagues use only one cloud to store the whole relation, the table in our scheme is vertically fragmented and each column is stored on a different cloud.

In the following subsections, the encryption algorithm and novel fragmentation technique are discussed in greater detail.

3.2.1 Encryption Algorithms

The algorithms used in our scheme are the same as the encryption algorithms used in Popa et al.'s (2012) approach. These algorithms include AES, OPE, HOM, Search, and DET. In our scheme, the usage is slightly different due to differences in the architecture between the two approaches. As was stated in the section on related work, the researchers extend the column as per the encryption algorithms needed for the column's data type. In our scheme, the concept of extending columns also exists, but the column is fragmented.

3.2.1.1 Advanced Encryption Standard (AES) Algorithm

The AES algorithm is used to encrypt the entire database in the master cloud, except for the index column. The index column is a system-generated primary key used to identify the tuples of the relation and is replicated in each cloud, including the master cloud. When a base column is encrypted using a specific algorithm, the resulting encrypted column is stored, together with the corresponding unencrypted index column, in a slave cloud. Thus, each slave cloud contains a table with two columns, an unencrypted index column, and the encrypted values of a base column. Each tuple has two values – the encrypted value from the base column, and the index value that identifies the tuple in the master cloud from which the encrypted value was derived. The encryption/decryption keys are stored at the proxy so that encryption and decryption can only be performed on the proxy side.

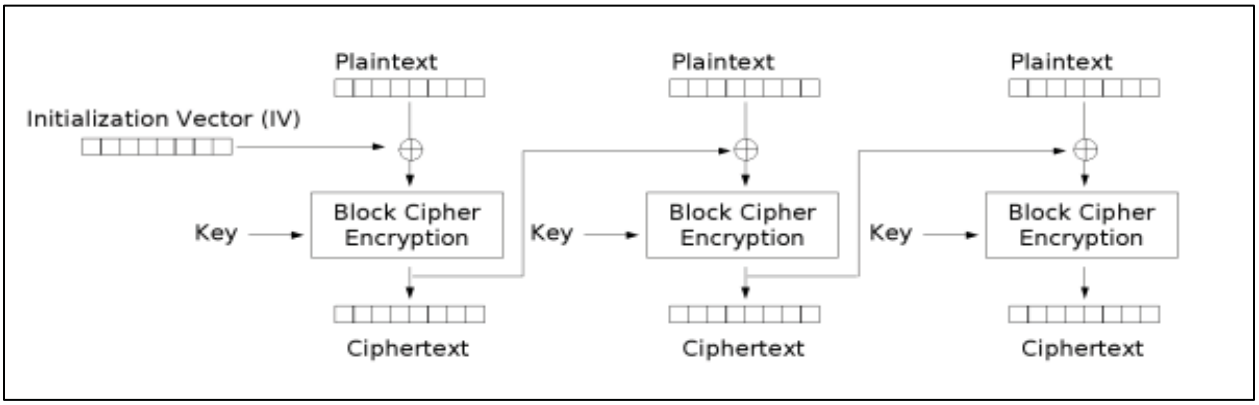


Figure 3-3: Cipher-block chaining (CBC) mode encryption (Adopted from Stallings (1999))

The cipher-block chaining (CBC) mode is used here due to its numerous security advantages, confidentiality being the main one. Hence, we use CBC to encrypt the master cloud replica.

Figure 3-3 shows how the AES-CBC works. The algorithm begins with three parameters, as Stallings (1999) describes: the message, the key, and the initialization vector. The algorithm is $C_i = E_K(P_i \oplus C_{i-1}), where\ C_0 = IV$. First, the message is divided into specific blocks. The first block is XORed with the initialization vector and then encrypted with the key. This results in a new cipher-text, which will be used again as the initialization vector for the next block of plain text. As an example, if the plain text is divided into three blocks, the ciphers will be as follows: $C_1 = E(K, P_1 \oplus IV)$, $C_2 = E(K, P_2 \oplus P_1 \oplus C_1)$ and $C_3 = E(K, P_3 \oplus P_2 \oplus C_2)$. The encryption process goes through four rounds, as Stallings, (1999) describes, including sub-bytes, shift rows, mix columns, and add round key. The blocks, along with the key, are scrambled through these rounds, producing the cipher-text.

As has been proven in the AES algorithm, the longer the key length is, the more secure the data encryption. Therefore, the key length chosen was 256 bits. The number of keys has been limited in our algorithm, due to the space needed to store them. The idea is to avoid any kind of storage mechanism at a private cloud. Storing data at a private cloud is not only against the concept of cloud computing in terms of storing data on the client side, but also the size of the database needed to store keys has to be almost equal to the size of the database in the master cloud. For these reasons, the number of keys is limited.

The only queries that can be processed upon encrypted data using this algorithm are shown in Table 3-1. Therefore, this algorithm cannot support any query that has a where clause. The idea is to fetch either the whole encrypted relation or a portion thereof.

Table 3-1: Example of queries supported by the master cloud

	Query	Predicate
	Select * From Table	No Where Clause
	Select * From Table LIMIT Value	No Where Clause

3.2.1.2 Order-Preserving Encryption (OPE) Algorithm

The OPE algorithm is one of the fundamental encryption algorithms used in relational databases. Generally speaking, most databases need to be able to compare between values, especially numerical ones. With this in mind, the OPE algorithm has been proposed by Liu et al. (2012) so that the query can be applied to encrypted data to provide a comparison operation.

This algorithm is based on linear expression as well as random noise. Algorithm 10 shows our implementation of OPE as described by Liu et al. (2012). The encryption algorithm is $a * v + b + noise$ where the coefficients a *and* b are the keys. The v is the value that needs to be encrypted.

The noise part of the expression is added to improve security. The noise is a random number of ranges, "0" to the coefficient "a" multiplied by sensitivity. The sensitivity depends on the data type of the value that needs to be encrypted. Thus, it is 1 or 0.01, depending on whether the value is an integer or double, respectively.

Basically, the cipher-text produced from this algorithm for a certain value will be in a range between zero and the coefficient "a". The range can be partially controlled, based on the coefficient "a". Therefore, the algorithm will produce a different cipher-text for the same plain text, which will enhance security because the pattern will be hidden.

One of the problems we faced when implementing this algorithm is that the comparison was not accurate if we had repeated values. The accuracy also decreased, as the number of repetitions increased. If the range of a given cipher-text exceeds 100, for example, then any encrypted value will be encrypted by one of these values. Meanwhile, the encrypted query value will also be encrypted by one of these values as well. For instance, if our range is 50, then encrypted values x_1 and $x_2$represent the plain text X. Assuming that the user initiated a query

asking for values greater than *X*, then the query-encrypted value is one of the possible values. Therefore, in this case, the $x_1$is greater than x_2 , whereas, in fact, both $x_1$and x_2 are equal because both represent the *X* value using the OPE algorithm. The retrieved result for the previous query will include any value greater than x_1 , which is not correct. As a result, this issue has to be overcome to make this algorithm fully accurate. We added two functions to the algorithm that are included to determine the maximum and minimum values of the range. Therefore, these values can be used to determine the greater value. This algorithm is the weakest algorithm used by Popa et al. (2012) because it reveals the maximum and minimum values and the order of the data set. It is also susceptible to plain-text attack. More details with regards to the security aspect, and in particular the weakness of OPE, are presented in the security analysis section.

Table 3-2 shows some examples of SQL queries that use the OPE algorithm.

Table 3-2: Examples of queries supported by OPE

	Query	Predicate
1	Select * From Table Where *column_name < encrypted value*	Less Than
2	Select * From Table Where *column_name > encrypted wvalue*	Greater Than
3	SELECT * From Table Where *column_name > encrypted value ORDER BY column_name*	ORDER BY, Greater Than

3.2.1.3 Homomorphic Encryption (HOM) Algorithm

The HOM encryption algorithm is needed in most encrypted relation databases due to the mathematical properties that it can support, such as multiplication, summation, averages, etc. the HOM algorithm can perform queries that require mathematical operations.

Algorithm 11 describes the HOM as proposed by Dijk et al. (2010), who begin by defining the key generation function variables. First, p and q are two random prime numbers, such that $gcd\big(pq,(p-1)(q-1)\big)=1$ because they have to be in the same length. The function in the second step will compute the $n=p*q$ and $\lambda = lcm\,(p-1,q-1)$. The g variable will be integer random number. Then, $\mu=\Big(L\big(g^{\lambda}\ mod\ n^2\big)\Big)^{-1}\ mod\ n$, where $L(u)=\frac{u-1}{n}$. They are used in the encryption function as the public key. The *n and g* are used in the decryption function as the private key. The encryption function has two parameters, which

are the message M that needs to be encrypted and R, which is a random prime number. Then, the cipher-text will be $C = g^m * R^n \, mod \, n^2$. The decryption function, on the other hand, returns the original message m by computing $m = L\left(c^\lambda \, mod \, n^2\right) * \mu \, mod \, n$. In our scheme, R is a primary key that used in the encryption and decryption.

This algorithm has the ability to compute the summation or average over cipher-text, making it a suitable solution for applications that require mathematical functionality. The product of two cipher-texts will decrypt to the sum of their corresponding plaintexts. We calculate the sum of values by $D(E(a) * E(b)) = a + b$ and divide the sum by the number of elements in case of the average is desired. A number of queries can be performed upon an encrypted database using HOM. Table 3-3 shows some examples of these queries.

Table 3-3: Examples of queries supported by HOM

	Query	Predicate
1	Select *AVG (column_name)* From Table	Average
2	Select *SUM (column_name)* From Table	Summation

3.2.1.4 Search Encryption Algorithm

Many cloud computing applications need to perform word search queries. The only way to perform a search query upon encrypted data is to use a deterministic encryption algorithm. In our proposal, we use the concept of the search algorithm in the same way that Popa et al. (2012) previously proposed with regards to search queries. This algorithm (Search) was originally proposed to search for words in text files by reserving the letter position as one of the algorithm's parameters. Popa et al. (2012) change the algorithm to support a word search in the context of databases (e.g., searching for someone's name). Serving the same purpose, RC4 is not only a deterministic encryption algorithm but is also faster at encrypting and decrypting. This makes it one of the best choices for supporting these kinds of queries.

The RC4 encryption algorithm is a popular encryption algorithm that has been used in many protocols, such as SSL, TLS, WEP, and WPA (Stallings, 1999). The algorithm begins with a key generation, which is a pseudorandom stream of bits. However, in our scheme the keys are statically chosen from the proxy storage. The selected key is then combined with the stream bit of the original message by the xor operation to produce the cipher-text stream. The

decryption, on the other hand, is the inverse of the procedure. Algorithm 1, adopted from Stallings (1999) and shown in Figure 1, was used as a basis for our implementation.

Algorithm 1:Search Encryption algorithm (Adopted from (Stallings (1999))

```
Input:
String : The Message;
Byte : The key;
Output:
String : Encryption of the message
String : Decryption of cipher-text
---------------------------------------------------------------
Description:
Public Class Search {
    Private byte [] S = new byte[256];
    Private byte [] T = new byte[256];
    public Search (byte[]key) {
        int keylen , j;
        byte t;
        for ( int i = 0; i< 255; i++)do
            S[i] = (byte) i;
            T[i] = (byte) key[ i % keylen ];
        end for
        j = 0;
        for ( int i = 0; i< 255; i++)do
            j =((j + S[i] + T[i])%256);
            swap (S[i], S[j]);
        end for
    }
    public byte[] encrypt (byte[] M ) {
        int i = 0, j = 0, t, k;
        byte [ ] M, C, S;
        Set C= new byte[M.length];
        for (int jj = 0; jj< 255; jj)do
            i=( ( i + 1 ) % 256);
            j=( ( j + S[ i ] ) % 256) ;
            swap ( S[ j ] , S[ i ] );
            t =(( S[ j ] , S[ jj ] )% 256 );
            k = S [t];
            C [jj] = (byte) (k ∧ M [jj] );
        end for
        Return C;
    }
     public byte[] decrypt (byte[] C) {
         Return encrypt(C);
    }
}
---------------------------------------------------------------
```

Algorithm 1b The call of Search Encryption Algorithm Class

Input:
String : The message;
Output:
String : Encryption of the message
String : Decryption of cipher-text

Description:

```
Public Main {
public String encryptEachChar(String str){
        Set ccChar[ ] = str.toCharArray();
        SetSearch search;
        for(int= 0; i< ccChar.length; i++)do
            Byte [] key = SearchStoredKeys.ByteArray(i);
            Set search = New Search(key);
            String S = " "+ccChar[i];
            String encryptedChar = search.encrypt(S.getBytes())
            Set EChar += encryptedChar
        end for
        Return EChar;
}
public String decryptEachChar(String str){
        Return encryptEachChar(str);
}
}
```

In this encryption algorithm, the two identical plain text messages have two identical cipher-texts. In spite of the fact that RC4 has known weaknesses, in practice it is a popularly used algorithm. As we need different encryption algorithms for our thesis, RC4 was chosen.

Table 3-4 below shows some examples of queries that use the Search encryption algorithm.

Table 3-4: Examples of queries supported by Search

	Query	Predicate
1	Select * From Table Where column_name = encrypted value	Equal
2	Select column_name(s) From table1 INNER JOIN table2 ON table1.column_name=table2.column_name;	Join

3.2.1.5 Deterministic Encryption Algorithm (DET): Advanced Encryption Standard (AES) – Electronic Code Book (ECB)

The AES algorithm has many operating modes in which security and performance differ (Stallings, 1999). The ECB mode is one of the deterministic encryption algorithms in which the cipher-texts produced from the algorithm by the same plain texts and key are the same, making it possible to perform the equality operation upon encrypted data. Popa et al. (2012) used this algorithm to support equality and join operations.

Figure 3-4 shows how the AES-ECB works. As Daemen (2002) explains, the AES-ECB is a block-cipher encryption algorithm in which the message is segmented into a number of blocks. Each block is encrypted independently with the same key. Unlike the CBC mode, the ECB mode can be processed in parallel, which means that this algorithm is faster. The combination of these encrypted blocks is the cipher-text of the message.

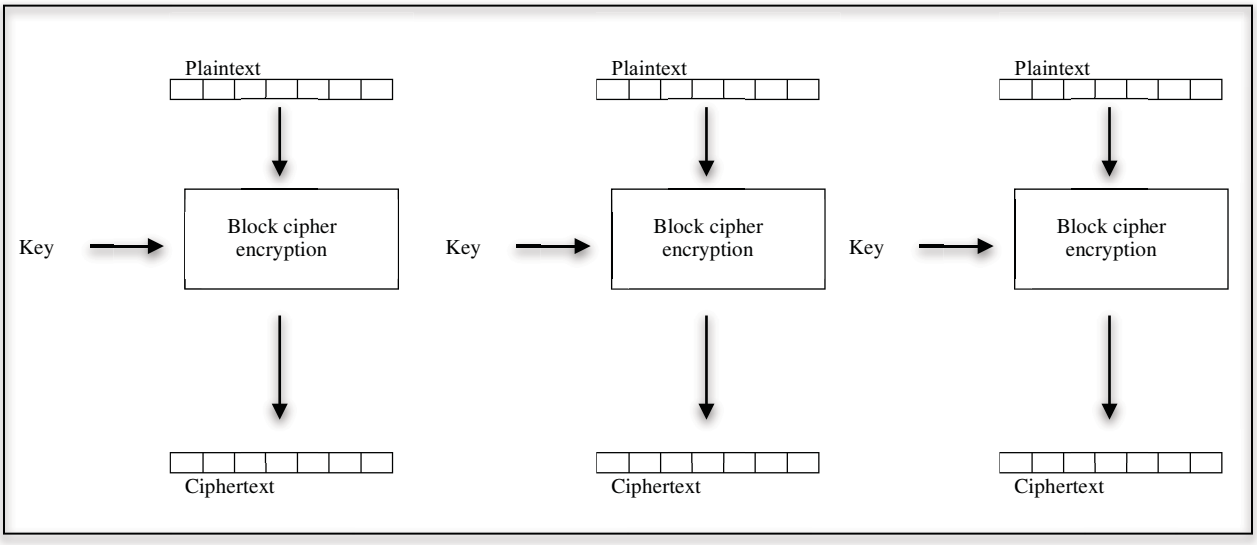


Figure 3-4: Electronic Code Book (ECB) Encryption Algorithm (Adopted from Stallings [1999])

Although this algorithm has the weakness of showing patterns, the fragmantation technique used in this proposal will improve the confidentiality of the user's data. Samarati et al. (2001) provide an in-depth analysis regarding how obtaining data from a single column does not reveal any senistive information.

3.2.2 Fragmentation Technique

In the previous sections, the algorithms used in Popa et al.'s (2012) approach are explained, along with the potential uses of these algorithms. These algorithms are also used in our proposed approach, but the novel aspect of our proposal is the fragmentation technique, which, we argue, improves a user's data confidentiality. The fragmentation technique involves two aspects: a master database and slave clouds (i.e., column-based fragmentation).

3.2.2.1 Master Cloud Computing

In the first phase, the replica of the entire database is encrypted using a highly secure encryption algorithm. The replica is then sent to the master cloud. The master cloud is the core part of our scheme, due to the fact that it maintains the entire relation database in one place. Hence, the encryption algorithm that is to be used in the master cloud has to be secure enough to hold sensitive data. A number of studies have proven that AES-CBC is a secure and reliable algorithm for the outsourced storage of sensitive data (Damen, 2002; Blomer, 2003). Consequently, several studies have used AES-CBC to store sensitive data (Mohamed, 2012;

Arasu et al., 2013; Nalinipriya & Kumar, 2013). Therefore, in our scheme, the AES-CBC encryption algorithm is used to encrypt all inputs to the master cloud database. The index is the only column kept in plain text, so that only the user can query the database through that index. The index is synchronous with other fragments' indexes as well.

The idea of storing an entire replica is to obtain the greatest benefit from cloud computing by preventing any kind of data storage on the client side. In most cases, the query has to travel to the master cloud to obtain encrypted data; however, this is not always the case. Furthermore, only two kinds of queries can be submitted to the master cloud database because any data encrypted by the AES-CBC algorithm cannot be queried directly without decryption. Table 3-5 shows some examples of queries, assuming there is only one table in the database.

Table 3-5: Master cloud queries

	Query	Result
1	Select * From Table	Return whole table
2	Select *column_name(s)* From Table	Return whole column(s)
3	Select * From Table Limit *value*	Return number of tuple(s) based on the limit value
4	Select * From Table Where index in (*value_1, value2_2, value_n*)	Return tuple(s) based on the index value

Table 3-6 shows how data stored in the master cloud looks. Despite the fact that the real columns' names are hidden in the master cloud, they are written here in plain text in order to illustrate the idea.

Table 3-6: The encrypted relation stored at the master cloud

Name	Age	Salary	Credit Card
m+PrBgyXiQzU8hPgKCZXXQ==	ojZg+e256Yftyh8yEHie8Q==	WthcvBEgklV81IG/Q+O/cQ==	daCDh7GSuV6UxPafoBeGNA==
gO27qNIhGZfa8oevIxBlHA==	Tq6nDLODH70AidmBhp0kpg==	0fnmLnr6leJZcoReZKbxpg==	2vQd6ExdlbGoFG3pinmQ+g==
1oxm8ph+1imprNJntZpmgQ==	vugao7dX+ncRUD2DMCy2Qg==	vI9IyWZb+nFi/UxPCP9evw==	TcX/+pwNuA7aomK8iSRJAQ==
McpBTWCICWVwv9z/XTgXvQ==	TuxV6ltKBxWORPkV1mPe7w==	DbJmtmJMpStWnakPv8Yfzw==	/WoAjCn96rvKwy9aXB4QwA==
HRP1lnTlwyKXQk30eaAEww==	r7kyUblEV/MupkdhibDK7g==	QD9ha97N6OAdYXEVzsxEIw==	HVm8pBFdRU0jKZ47fmmhtw==
m5vwG3aqY6Sffqi/OSck0g==	icY7QSaXvtiv1VQEU8AdaQ==	sccqI9pQxMwbQIcJBaAOUg==	s951a8V4HLTnFKOJGEOlHQ==
u0LkEDrLbqp3CM/iTMsV+w==	JU5MpxAZ3TStFdiSLAszOA==	doGfV1KFSALqI3k9rACbLw==	rhGbkTNH16pZ2C7MoQSy1w==
bnHQ8lyFWxr5O2nAOQWaJg==	6A40qBBIOSzTztHrG+Mq5A==	J9wDlN/rp5KRVBfzZ5Pz0A==	j5E/lOOXGuxhWU5+wil4Jg==
VBoDknQyOoU+44bdVqdyWA==	UGArgoCGTwSe4JPxXTpWPQ==	+e3jf+vMAmmW27tcWMFpdA==	QDkBU75QNHd/EMxgegz3yg==
KZZmHzCkDsLHF9u41I/Apg==	uUe1y4QB37NARzfl9a3UAg==	udqaXmG3cSexeLJX7rRy6A==	ms0IuAU58msOEl8RNr2c0g==
XHXG8yg0IysGREuLrOQvZA==	iNSTnoEDyasuVnC2G4Ug6g==	GRN6eZZhHRvRuTI3gEjy0A==	wLbnL+ePw53i2dhPnUsV9w==
ZXcvAKV8/1OOYuYlHIBBTA==	c5p4K0S+kMJ7WmNo9zdLdg==	cELpiENBox3cgn/3pnG30w==	Mz5af+WXdd3i0t/Lo/8o+Q==
E/gffN3XMxE+pABE++Ho1Q==	uHyavgv4JZ6xvkSIMXnLPQ==	rgszGdhA72gwAufgj9KplQ==	u3SEBQRN0tM9O/jQ5TkbmQ==
re2FwXElTdR4gyL05M4w7w==	s5IEz0IS4I8fm1OUk7VeTA==	ko5fDq1u3YZ/cjBevg3ypw==	0z9de/33GYviodZEuL4dwg==
MSCxynny49sQaawbdyyoHA==	o23NGMtMb0Xck//Z4Pwpbw==	yy1ogZFFojzgwfNbRzSbPA==	S2r9cra2ieZCX2B+dRP/PQ==

3.2.2.2 The Slave Cloud

In the second phase, the vertical fragmentation approach is applied to a second replica of the database. The table is fragmented based on the columns. For each fragment, a new column is created with a synchronous plain text index amongst the fragments and a replica in the master cloud, as shown in Figure 3-5. In the master cloud, there is only one encryption algorithm used, but this is not the case with slave clouds. Slave clouds may need more than one encryption algorithm to support different kinds of queries. Hence, the columns will be extended based on the number of encryption algorithms needed. The number of columns in any slave cloud is $C_n \geq 2$, because each fragment has at least one encryption algorithm column, along with the plain text index column.

The algorithm used to encrypt a fragment depends on the type of data in that fragment. The choice of encryption algorithms also depends on the application needs. Generally speaking, numeric columns usually require mathematical operations, whereas string columns require search and matching operations. The encryption algorithms are used in an approach similar to that of Popa et al. (2012), so that the encrypted database will be directly queried without any decryption. Finally, the fragment will be sent to its slave clouds. Figure 3-5 illustrates how this process proceeds.

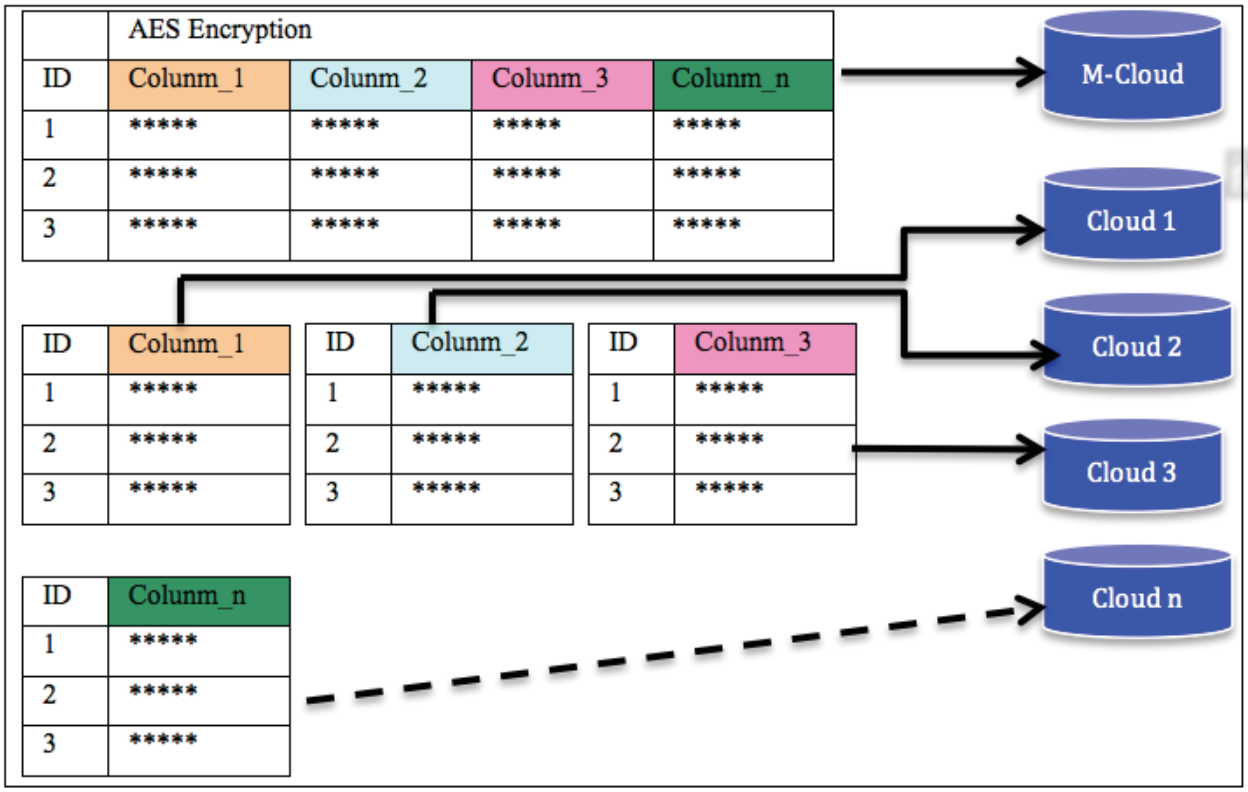


Figure 3-5: The fragmentation technique used in our scheme

3.2.2.3 The Proxy Server

The proxy is an important part of our scheme, as it performs most of the processing. Processes performed by the proxy server include creation, insertion, encryption, decryption, query parsing and the retrieval of results. The proxy sever has to be on the private cloud side and run through a highly secure channel, such as SSL.

3.2.2.3.1 Initialize Database

The scenario of our scheme begins with a plain text query from the user to create a table. When the proxy receives the query, it will create two tables: the master table and the slaves' tables. The proxy first sends the created query table to the master. Next, it will fragment the relation vertically. These fragments have only one column so far, yet the proxy has to determine what type of data is in each fragment so that the number of columns will be extended to the number of encryption algorithms needed. Finally, before the proxy sends the created table query, it will determine which fragment in which slave clouds will be created. Figure 3-6 describes the process of creating tables.

3.2.2.3.2 The Insertion Query

The insertion process is a significant phase in the process. When a user initiates the insert query, the proxy will receive that query in plain text. The proxy will then encrypt each row using the AES-CBC and send them to the master cloud. Concurrently, the same data will be inserted into the slave clouds with the appropriate encryption algorithm(s), depending on the data type. Algorithm 2 shows how the proxy sends the insertion query to the master cloud.

Algorithm 2: The Insertion to the Master Cloud

Input:
String [] : Record;
Output:
Int : The Last Index Id

Description:
public int setInsertValues(String[] Recod) {

```
Table table;
if (Record.length >0) Then
    Set table = new Table(C1, C2, .., Cn);
else
    Return id = -1;
end if
if (id != - 1) then
    Set futureId = getlastIdFromDB;
    Set key[] = StoredKeys.toByteArray(futureId + 1);
    for (i = 0 to i< # column)do
        Set Ci = AES.encrypt(key, Table.get(i));
        preparedStatementMaster.setString(i, Ci_Encrypt);
    end for
    preparedStatementMaster.executeUpdate();
end if
Return lastIndex;
}
```

To create any fragment of the database, the proxy has to determine the data type for that column. The proxy then encrypts the data using the encryption algorithms that are appropriate to that data type. Finally, the encrypted data will be sent to the cloud database. For numeric values, the algorithms used are OPE, HOM, and DET, whereas when the data type is string, the Search encryption algorithm is used. Algorithms 3 and 4 describe the insertion process for both numeric and string values, respectively.

Algorithm 3: The Insertion to the cloud with number value

Input:
String [] : Record;
Output:
Int : last Index

Description:
public int setInsertValues(String[] Record) {

Table *table*;
if (*Record.length* >0) **Then**
Set *table* = new Table(C_1, C_2, C_3); **else**
Return *id* = **- 1;**
end if
if (*id*! = −1) **Then**
Set C_1 = **ope.encryptInt(table.getC**$_1$());
Set C_2 = **Hom.encryptInt(table.getC**$_2$());
Set C_3 = **Det.encryptInt(table.getC**$_3$());
preparedStatementCloud$_i$.*setInt*(1, *id*);
preparedStatementCloud$_i$.*setInt*(2, C_1);
preparedStatementCloud$_i$.*setInt*(3, C_2);
preparedStatementCloud$_i$.*setInt*(4, C_3);
preparedStatementCloud$_i$.*executeUpdate*();
end if
Return lastIndex;
}

Algorithm 4: The Insertion to the cloud with string value

Input:
String : Record;
Output:
Int : Last Index

Description:
public Int setInsertValues(String[] Records) {

```
Table table;
if (Record.length>0) Then
    Set table = new Table(C1);
else
    Return -1;
end if
if (id! = −1) Then
    Set C1 = New Search.encryptEachChar(table.getC1());
    preparedStatementCloudi.setInt(1, id);
    preparedStatementCloudi.setString(2, C1);
    preparedStatementCloudi.executeUpdate();
end if
Return lastIndex;
}
```

3.2.2.3.3 The Selection Query

Retrieving the query result from the database requires a number of steps. First, the proxy has to parse the query syntax to determine not only which slaves need to be accessed but also the encryption algorithm needed to encrypt or decrypt. After this is determined, the proxy will encrypt where clause value(s) and then send the query to the slave clouds. If no where clause is found in the parser phase, then only the master cloud database is queried. Otherwise, the slave clouds will process the query and return only the plain text index.

Finally, the proxy will query the master cloud database by using the indexes that were retrieved from the slave clouds. After the proxy receives the encrypted results from the master, it will decrypt the data and send it to the user. Table 3-7 shows a number of examples of queries submitted to the master cloud database after obtaining the indexes.

Table 3-7: Examples of queries that are initiated from the user and the proxy

	Query	The initiator	The receiver
1	Select * From Table Where column_name > value	The user	The proxy
2	Select *id* From Table Where column_name > E(value)	The proxy	The slaves clouds
3	Select * From Table Where index in (*id_1, id_2, id_n*)	The proxy	The Master cloud

3.2.3 The Key Management

According to Stallings (1999), using a random generation keys technique increases the security level of an encryption algorithm. In our scheme, however, the keys are not shared with public clouds but are instead stored in the proxy. Therefore, we do not generate a key for each cell because, if we did so, we would have database on the proxy for the keys that is almost the same size as the original database stored on the master cloud. Accordingly, we design our key generation mechanism so that the number of the keys is limited to ten keys. The concept is to eliminate the usage of proxy storage, so the number of the keys can be more or less than ten keys. For simplicity, we only store ten keys at the proxy to encrypt/decrypt master relation.

The key technique used to encrypt and decrypt the master cloud's tables is a key for all cells within a tuple. If the number of tuples in the master relation is greater than the number of keys, then there are at least two of the tuples encrypted by the same key. The keys are chosen in a semi-random manner. We call it semi-random key generation because it is not truly random as the key is chosen based on both the primary key index value and the number of the key stored at the proxy which is ten keys in our case. The tuple's index is one of the algorithm's important parameters. Algorithm 5 illustrates how the keys are assigned to a tuple. In the initial stage, a number of keys are generated and stored in an array. Then, the algorithm computes the $x = index \,\%\, Keys\_Array.length$. Based on the x, the $K = Keys\_Array[x]$. Using this technique reduces the space needed to store keys at the proxy while there is no shared key with the cloud provider. Of course, this technique is based on the assumption that the public cloud provider, where the encrypted tuples of the master table are stored, does not know how the keys

are generated - otherwise the attacker can treat the encrypted master replica as 10 smaller tables, each one containing tuples encrypted using one key.

Algorithm 5: The Key Management Algorithm

Input:
int : Primary key Index;
Output:
String : Key to encrypt or decrypt value

Description:

```
public class StoredKeys {
    Public static String[]keys =newString[]{K1, K2, K3, ., ., .Kn} ;
    Public static String toHexString (byte[] array){
        Return DatatypeConverter.printHexBinary(array);
    }
    Public static byte[] toByteArray(int i) {
        Set s = keys[getKeyId(i)];
        Return convertHexToByte(s);
    }
    Public static byte[] convertHexToByte (String s) {
        Return DatatypeConverter.parseHexBinary(s);
    }
    Public static int getKeyCount(){
        Return keys.length;
    }
    Private static int getKeyId(int id) {
        Return id % StoredKeys.getKeyCount();
    }
}
```

CHAPTER 4: IMPLEMENTATION AND EVALUTAION

In this chapter, the implementation of the prototype as a proof of the concept is described. First, we will illustrate the tools that used and then the proxy's functionality will be described. Finally, evaluation of the implementation will be presented.

4.1. Implementation

4.1.1 Cloud Computing Tools and Configuration

As a first step in proving our concept, we created a public cloud computing account at Rackspace, which offers open source cloud computing. We then created a number of servers equal to the number of fragments that we need, plus one more for the master cloud. These servers have the following features:

- Linux OS, XAMPP Server
- CPU: 2 vCPUs
- RAM: 4 GB
- System Disk: 160 GB
- Network: 400 Mb/s

Both the master cloud and the slave clouds are running the Linux operating system. They are also running the XAMPP server to house MySQL DBMS. This tool is needed to store a master relation and the fragments in the slaves.

The proxy, on the other hand, is a server at the private cloud. It is running the OS X. It is also running with the Tomcat server.

- OS X
- Tomcat server 7.0.53
- CPU: 2.4 GHz Intel Core i5
- RAM: 10 GB RAM
- System Disk: 500 GB
- Network: 150 Mbps

The proxy is significant to our scheme because all queries have to go through the proxy. In the next section, the proxy's functionality will be described in detail.

4.1.2 The Proxy's Functionality

The proxy's processes can be divided into two major functions: query translation and encryption/decryption. These functions will be discussed in detail in the next subsection.

4.1.2.1 The Query Translation

In our scheme, the proxy is located in-between the users and the public cloud, where the data is stored. Therefore, the user cannot query the database directly; instead, the proxy will process the query.

The user first initiates a query and sends it to the proxy. The proxy will rewrite the query based on the Where clause and send it to the slave clouds. As a proof of concept we implement the process of querying the slave clouds only in serial fashion. Utilizing the serial fashion has no performance effect on a query that has no predicates as the master cloud storing the encrypted replica is accessed. The effect increases with the increases in the number of predicates as for each predicate a slave is queried and the slaves are queried in a serial fashion, one after another as opposed to in parallel. We propose the parallel request implementation as future work. The proxy will receive only the plain text indices of the match results, if any. The union or intersection algorithm is applied to the indexes if there is more than one where condition/predicate in the Where clause. If these conditions are separated by OR, the union algorithm is used; otherwise, the intersection algorithm is used. Algorithm 6 shows how the proxy queries the slave clouds to retrieve the indexes.

Second, when the slaves return the indexes to the proxy. The proxy rewrites another query to the master relation and fetches the tuples that have the same index. Algorithm 7 shows how the proxy rewrites the second query to query the master relation.

Algorithm 6: Normalized Return Indexes

Input:
String : Where Clues;
Output:
Int : Indexes From the Clouds

Description:

```
Private ArrayList < Integer >getNormalizedId (String arg)
{
    Set ArrayList<Integer >ids = new ArrayList <Integer();
    Set AND = " and ";
    Set OR = " or ";
    Set  andOrSplit[] = null;
    if (arg.contains(AND)) then
        andOrSplit = arg.split(AND);
        Set <Integer>localHashIds = new TreeSet <Integer >();
        for (String str : andOrSplit)do
            if (localHashIds.isEmpty()) then
                ArrayList <Integer>tempIds = getIdsFromNodes(str);
                for (Integer id : tempIds)do
                    localHashIds.add(id);
                end for
            else
                ArrayList <Integer>tempIds = getIdsFromNodes(str);
            end if
        end for
    else if (arg.contains(OR)) then
        andOrSplit = arg.split(OR);
        for (String str : andOrSplit)do
            ids.addAll(getIdsFromNodes(str));
        end for
    else
        ids = getIdsFromNodes(arg);
    end if
    Return ids;
}
```

Algorithm 7: Building query for the master cloud

Input:
String: Where Clues;
Output:
String : Encrypted result from the master cloud

Description:

```
public ResultSet getSelectResultSet(String Where_Claues) {

    Set rs= null;
    Set sql= "SELECT * FROM table";
    Set  where = "where";
    if (Where_Clues.contains(where)) Then
        Set  Where_Claues = Where_Claues.substring
              +(Where_Claues.lastIndexOf(where);
              +where.length()).trim();
        Set ArrayList<Integer >ids = getNormalizedId(Where_Claues);
        Set  sql += " where id in (";
        for (int i = 0; i<ids.size(); i++) do
            Set  sql += ids.get(i);
            if (ids.size()>i + 1) Then
                Set  sql += " , ";
            end if
        end for
    end if
    Statement  statement = conn.createStatement();
    Set  rs = statement.executeQuery(sql);
    Return rs;
}
```

4.1.2.2 The Query Parser

Building the queries is the most important function of the proxy. As the prototype of our scheme requires it, we created a number of functions to build slave queries. These functions are created to query the string and numeric values. Algorithms 8 and 9 show how the parser works for string and numeric values, respectively.

Algorithm 8: Building a query for string values with equality predicate

Input:
String [] : one term of where claues
Output:
String : query to be sent to a certain cloud

Description:

```
private String WhereClauseString (String[] where_Clause) {

    Set sql = "Select id From Table Where ";
    if (whereClause[1].trim().equals(" = ")) Then
        Set name = new Search().encryptEachChar(whereClause[2]);
        Set sql = sql + "name " + whereClause[1] + " '" + name + "'"
            end if
    Return sql;
}
```

Algorithm 9: Building a query for number values with comparison and equality predicates

Input:
String [] : where claues
Output:
String : query to be sent to n cloud

Description:

```
private String WhereClause_Number (String[] where_clause) {

    Set sql = "Select id From Table Where ";
    if (where_clause[1].trim().equals(" = ")) Then
        Set Number = Det.encrypt(whereClause[2]) ;
        Set sql = sql + "Det " + whereClause[1] + " '" + Number + "'";
    else if (whereClause[1].trim().equals(">||<")) Then
        Set Number =ope.(whereClause[2]);
        Set sql = sql + "ope " + whereClause[1] + " " + Number;
    end if
    Return sql;
}
```

4.1.2.3 The Encryption Algorithm

In our scheme, the proxy performs all of the encryption and decryption. In the encryption process, the proxy encrypts any inserted values that come from the user before storing them at the clouds. The selection query value also has to be encrypted by the proxy before querying any slave clouds. None of the clouds store data in plain text except the indexes, and the slaves do not decrypt any query. In our scheme, five encryption algorithms are used. Some of these are standard encryption algorithms, while others are proposed for the purpose of querying outsourced encrypted data. These algorithms are explained in section 3.2.1's encryption algorithms and include AES, DET, Search, OPE, and HOM. Algorithms 6 and 8 shows our implementation of OPE and HOM, as described by Liu et al. (2012) and Dijk et al. (2010), respectively, whereas the others are already explained in section 3.2.1 (encryption algorithms).

Algorithm 10: Ope Encryption Algorithm

Input:
Int : Plain text value ;
Double : Plain text value ;
Output:
Int : Encrypted value

Description:

```
public class OPE {
        Set a= K1;
        Set b= K2;
        Set sensivity = 0.01;
        public OPE (double a, double b) {
            Set this.a = a;
            Set this.b = b;
        }
        public double encryptInt (double v){
            Return a * v + b + (int)generateNoise(a * 1);
        }
        public double encryptDouble (double v){
            Return a * v + b + (int)generateNoise(a * sensivity );
        }
        public double generateNoise (double sensivity){
            Return (Math.random() * sensivity);
        }
        public double getMaxInt (double v){
            Return a * v + b + (int)(a * 1);
        }
        public double getMinInt (double v){
            Return a * v + b ;
        }
        public double getMaxDouble (double v){
            Return a * v + b + (a * sensivity);
        }
        public double getMinDouble (double v){
            Return a * v + b ;
        }
    }
```

Algorithm 11: Hom Encryption Algorithm

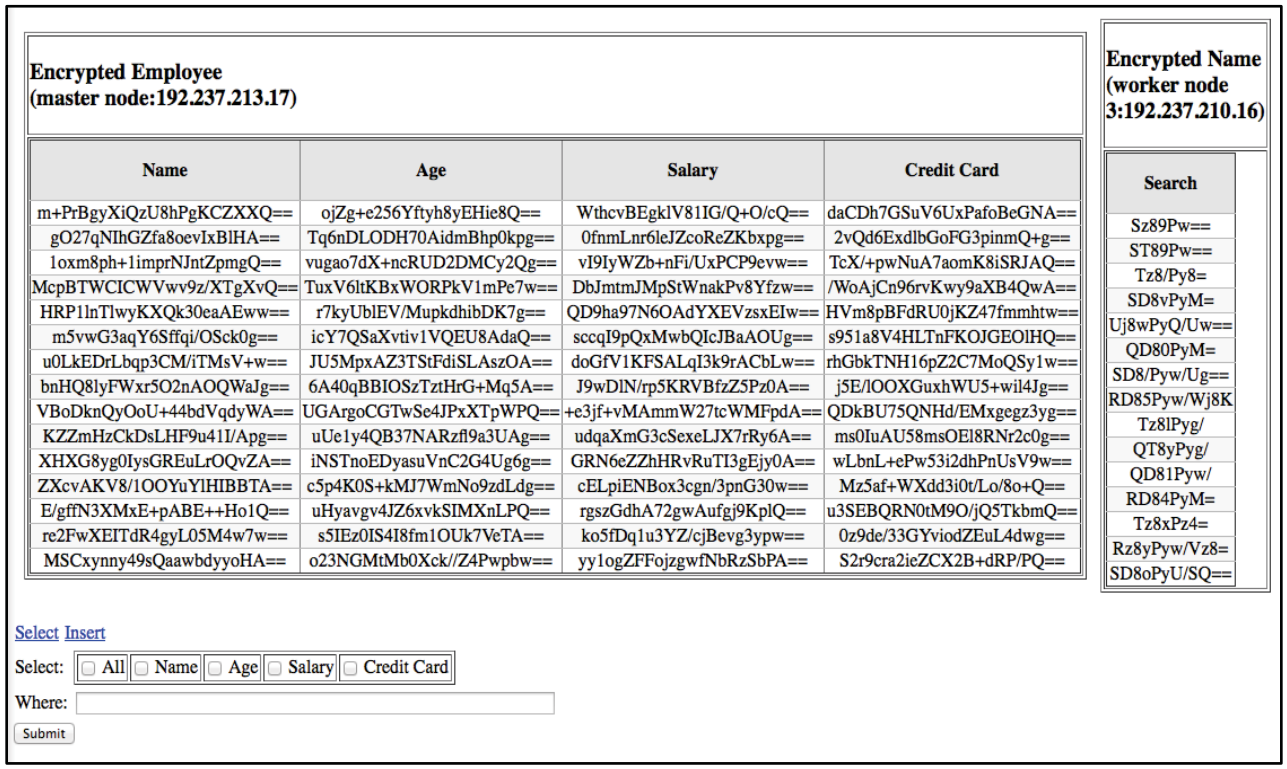
Input:
Int : Plain text value ;
Output:
BigInteger : Encrypted value

Description:

```
Public Class HOM {
    Private BigInteger p, q, lambda, g;
    Public BigInteger n, nsquare;
    Private int bitLength;
     Public HOM (int bitLengthVal, int certainty) {
         KeyGeneration(bitLengthVal, certainty);
    }
     Public HOM () {
        KeyGeneration(512, 64);
    }
     Public void KeyGeneration (intbitLengthVal, intcertainty){
        bitLength=bitLengthVal;
        p=new BigInteger(key);
        q=new BigInteger(key);
        g=new BigInteger(key);
        n= p * q;
        nsquare=n * n;
        lambda= (p − 1) * (q − 1)/ gcd(p − 1, q − 1);
    }
     Public BigInteger Encryption (BigInteger m ) {
         r = newBigInteger(key);
        Return g^m * r^n mod nsquare;
     }
     Public BigInteger Encryption (BigInteger m, BigInteger r ) {
         Return g^m * r^n mod nsquare;
     }
     Public BigInteger Decryption (BigInteger c) {
         Set u = (L(g^lambda mod nsquare))^((−1)mod n);
         Return L(c^lambda mod nsquare) * u mod n;
     }
}
```

4.1.2.4 The Interface

In this section, the interface of the prototype is described. The interface is designed as a web page (i.e., JSP files), where a sample of only 15 rows of the master encrypted database and the slave encrypted fragments are presented. The user can choose which kind of query she/he wants to submit (e.g., insert, select). If the user chooses insert, there will be input boxes for all of the columns; when the user inputs the data and clicks the submit button, the information will be inserted into all of the clouds. On the other hand, if the user chooses the select option, there is an input box for the user to input the query parameters and a submit button. The result and the execution time will appear soon after the submit button is clicked.



Encrypted Employee (master node:192.237.213.17)

Name	Age	Salary	Credit Card
m+PrBgyXiQzU8hPgKCZXXQ==	ojZg+e256Yftyh8yEHie8Q==	WthcvBEgklV81IG/Q+O/cQ==	daCDh7GSuV6UxPafoBeGNA==
gO27qNIhGZfa8oevIxBlHA==	Tq6nDLODH70AidmBhp0kpg==	0fnmLnr6leJZcoReZKbxpg==	2vQd6ExdlbGoFG3pinmQ+g==
1oxm8ph+1imprNJntZpmgQ==	vugao7dX+ncRUD2DMCy2Qg==	vI9IyWZb+nFi/UxPCP9evw==	TcX/+pwNuA7aomK8iSRJAQ==
McpBTWCICWVwv9z/XTgXvQ==	TuxV6ltKBxWORPkV1mPe7w==	DbJmtmJMpStWnakPv8Yfzw==	/WoAjCn96rvKwy9aXB4QwA==
HRP1lnTlwyKXQk30eaAEww==	r7kyUblEV/MupkdhibDK7g==	QD9ha97N6OAdYXEVzsxEIw==	HVm8pBFdRU0jKZ47fmmhtw==
m5vwG3aqY6Sffqi/OSck0g==	icY7QSaXvtiv1VQEU8AdaQ==	sccqI9pQxMwbQIcJBaAOUg==	s951a8V4HLTnFKOJGEOlHQ==
u0LkEDrLbqp3CM/iTMsV+w==	JU5MpxAZ3TStFdiSLAszOA==	doGfV1KFSALqI3k9rACbLw==	rhGbkTNH16pZ2C7MoQSy1w==
bnHQ8lyFWxr5O2nAOQWaJg==	6A40qBBIOSzTztHrG+Mq5A==	J9wDlN/rp5KRVBfzZ5Pz0A==	j5E/lOOXGuxhWU5+wil4Jg==
VBoDknQyOoU+44bdVqdyWA==	UGArgoCGTwSe4JPxXTpWPQ==	+e3jf+vMAmmW27tcWMFpdA==	QDkBU75QNHd/EMxgegz3yg==
KZZmHzCkDsLHF9u41I/Apg==	uUe1y4QB37NARzfl9a3UAg==	udqaXmG3cSexeLJX7rRy6A==	ms0IuAU58msOEl8RNr2c0g==
XHXG8yg0IysGREuLrOQvZA==	iNSTnoEDyasuVnC2G4Ug6g==	GRN6eZZhHRvRuTI3gEjy0A==	wLbnL+ePw53i2dhPnUsV9w==
ZXcvAKV8/1OOYuYlHIBBTA==	c5p4K0S+kMJ7WmNo9zdLdg==	cELpiENBox3cgn/3pnG30w==	Mz5af+WXdd3i0t/Lo/8o+Q==
E/gffN3XMxE+pABE++Ho1Q==	uHyavgv4JZ6xvkSIMXnLPQ==	rgszGdhA72gwAufgj9KplQ==	u3SEBQRN0tM9O/jQ5TkbmQ==
re2FwXEITdR4gyL05M4w7w==	s5IEz0IS4I8fm1OUk7VeTA==	ko5fDq1u3YZ/cjBevg3ypw==	0z9de/33GYviodZEuL4dwg==
MSCxynny49sQaawbdyyoHA==	o23NGMtMb0Xck//Z4Pwpbw==	yy1ogZFFojzgwfNbRzSbPA==	S2r9cra2ieZCX2B+dRP/PQ==

Encrypted Name (worker node 3:192.237.210.16)

Search
Sz89Pw==
ST89Pw==
Tz8/Py8=
SD8vPyM=
Uj8wPyQ/Uw==
QD80PyM=
SD8/Pyw/Ug==
RD85Pyw/Wj8K
Tz8lPyg/
QT8yPyg/
QD81Pyw/
RD84PyM=
Tz8xPz4=
Rz8yPyw/Vz8=
SD8oPyU/SQ==

Select Insert

Select: All Name Age Salary Credit Card

Where:

Submit

Figure 4-1: Sample of the interface

Figures 4-1 shows samples of the interface where the user can issue the query to the proxy. The proxy encrypts any values appearing in a where clause or the insert statement and sends it to the cloud. Finally, the proxy decrypts the retuned result and sends it to the user.

Our scheme supports a simple select, insert, delete, and update where figure 4-1 shows only select and Insert options. However, it does not support the aggregation select query due to the lack of the encryption algorithms that support this kind of query. Investigating the area of supporting aggregation query could be a fruitful future research.

4.2 Evaluation

We evaluate of our method by determining the total delay using an analytical model and through modeling. In the evaluation, we concentrate on the total delay (in ms) per user SQL request. As the user will receive the whole query result in one response, the total delay in our experiments is also latency. Our method is compared to two base methods: (a) The first is an unsecured method in which the DB is stored in one cloud and there are no efforts to provide confidentiality. Thus, data is stored and communicated in plain text. We refer to this method as the Unsecured approach. (b) The second method is that of Popa et al. (2012) approach, which is the method we extended by fragmentation of data. Recall that in Popa et al. (2012) approach, the data is encrypted and stored in one master cloud. In the master cloud, each column is encrypted using encryption algorithms that support desired operations, such as equality selection supported by the encryption method DET and summation supported by the encryption method HOM. We refer to this approach as Secured centralized approach, or simply Centralized approach/method. In our method, in addition to storing the whole relation encrypted using AES in the master cloud, we partition the relation vertically and store each column together with an index column in a different cloud. The column is encrypted using the same methods as were used in Popa et al. (2012) approach. Thus, our method will be referred to as Secure Distributed Approach (SDA), or simply Distributed approach/method.

4.2.1 Evaluation Method

Our evaluation has several major steps in which experiments, in which queries are issued, are used for evaluation. Experiments report delays due to communication, crypto processing (encryption/decryption), query processing, and the total delay. When modeling is used, each experiment is repeated a number of times, and we report the average delay and variance.

4.2.1.1 Major Steps

1. We first develop an analytical model in which delays are predicted on the basis of various parameters that characterize delays for communication, cryptographic operations, query processing, and proxy delays. To do the evaluation using the developed analytical model, we need a set of experiments to measure the parameters that characterize the costs/delays. For instance, we measure the average delays for sending messages of various sizes. We then

perform evaluation using a set of queries for which delays of user queries are predicted using the analytical method. The queries will be described presently.

2. We perform the evaluation using modeling. In other words, we build the proxy, store the encrypted and plaintext relation in the master cloud, and also store encrypted columns of the relation in different clouds for our method. We perform the set of experiments, using the same queries as for the analytical model, and measure delays.

3. We analyze the results.

4.2.1.2 Set of queries

The set of queries used for evaluation has been selected and categorized to three groups as follows:

1. Select statement in which the Where clause contains one simple predicate.

The Select statement retrieves a certain number of tuples based on the selectivity factor. In the Secure distributed method, the proxy sends the request directly to the slave cloud in order to obtain the indices from the slave cloud that match predicate and these indices are then used to query the master cloud. Therefore, the communication delay for a query with one predicate is two round trips. It begins from the proxy to the slave cloud, back from the slave cloud to the proxy, then from the proxy to the master cloud, and finally back from the master cloud to the proxy Table 4-1 shows the first set of queries as well as their selectivity factors. The relation size is kept in fixed to 8000 tuples.

Table 4- 1: Query 1 - Selectivities and Parameter Values

Query	Sel. Fac. for Master Cloud	Select * From Employee where Age Op a			# of tuples
		Operator & values		Selectivity factors for Cloud 1	
		Op	a	Cloud 1	
Query1.1	0.2	<	28	0.2	8000
Query1.2	0.4	>	42	0.4	8000
Query1.3	0.6	<	43	0.6	8000
Query1.4	0.8	>	27	0.8	8000
Query1.5	1	>	5	1	8000

2. Select statement in which the Where clause contains two predicates.

The Select Statement has a Where clause containing two predicates. In the Unsecured and Secure Centralized approaches, the proxy has to query only the master. Recall that in SDA, that is the Secured Decentralized approach, the proxy first queries the slave clouds that returns, as the query result, indices that serve as primary keys to the selected tuples. After the proxy receives these indices from the slave clouds, the proxy finds their intersection and uses the result to query the master. As the number of predicates is increased, , the number of round trips increases,. It should be noticed also that each slave cloud returns a result with a different selectivity factor, while their product yields the selectivity factor for the query answer. Table 4-2 shows the second set of queries as well as their selectivity factors

Table 4- 2: Query 2 - Selectivities and Parameter Values

Query	Sel.Fac. for Master Cloud	Select * From Employee where Age Op a and salary Op s						# of tuples
		Operator & values				Selectivity factors for Clouds		
		Op	a	Op	s	Cloud 1	Cloud 2	
Query 2.1	0.2	>	31	<	367	0.3	0.7	8000
Query 2.2	0.4	>	23	<	618	0.9	0.4	8000
Query 2.3	0.6	>	28	<	300	0.8	0.8	8000
Query 2.4	0.8	>	23	<	200	0.9	0.9	8000
Query 2.5	1	>	5	<	99	1	1	8000

3. Select statement in which the Where clause contains three predicates

The last set of experiments is a Select statement that has a where clause with three predicates, i.e., it is a conjunction of a three of predicates. The proxy first encrypts the values appearing in the predicates and then, for the unsecured and Secured centralized methods the query is sent to the one (master) cloud which returns results. In the case of our SDA method, the proxy builds sub-queries, encrypts each query value, and then sends the sub-queries to the slave clouds. As stated in the previous chapter, the returned results from the slave clouds are indices serving as primary keys for tuples stored in the master cloud. As in the experimentation we only use a conjunction of predicates (predicates connected with the AND operator), the proxy performs an intersection over the returned results to build the query for the master cloud. The proxy delay in this experiment increases in comparison to the previous experiments because the query has more than one predicate. It requires more processing at the proxy side in

terms of building and sending sub-queries to the slave clouds and performing intersection of the sets of returned indices from the slave clouds in order to query the master. Additionally, the total delay is further increased if the proxy handles sub-queries in a serial fashion. Table 4-3 shows the set of queries as well as their selectivity factors.

Table 4- 3: Query 3- Selectivities and Parameter Values

Query	Sel. Fac. For master	Select * From Employee where Age Op a and salary Op s and Age Op a									# of tuples
		Operator & values						Selectivity factors for clouds			
		Op	a	Op	s	Op	a	C # 1	C # 2	C # 3	
Query3.1	0.2	>	51	>	367	>	51	0.3	0.7	0.3	8000
Query3.2	0.4	>	23	>	618	>	23	0.9	0.4	0.9	8000
Query3.3	0.6	>	28	>	300	>	28	0.8	0.8	0.8	8000
Query3.4	0.8	>	23	>	200	>	23	0.9	0.9	0.9	8000
Query3.5	1	>	5	>	99	>	5	1	1	1	8000

4.2.2 Evaluation Using Analytical Model

4.2.2.1 Analytical model

Processing a query incurs the following costs/delays: communication, crypto (encryption/decryption), query processing, and proxy delays.

4.2.2.1.1 Communication Delay

1. We model the communication delay of a message transfer by using the generally accepted Equation 4.1:

$$D_s = a + b\,x \qquad \text{Equation 4.1}$$

The delay is a linear function of a set-up overhead and the size of the message. As we are using Internet for communication, we are not able to determine the maximum size of packets and hence we model communication on a message basis, as opposed to a more detailed modeling in which a message is sent in packets if it exceeds the maximum packet size. The communication delay is affected by the propagation delay, serialization, data protocol and latency, routing and switching latencies, and queuing and buffer management and each of these factors has an impact on the total delay (Stallings, 1995). Since the messages are going over

Internet, however, we have no control or real knowledge on the individual delay components and hence resort to a simple model represented by Equation 4.1.

4.2.2.1.2 Crypto Delay

This is a delay for encryption/decryption, and is modeled by Equation 4.2:

$$D_c = c\,n \qquad \text{Equation 4.2}$$

Crypto delay is directly proportional to n, where n is the number of items to be encrypted/decrypted and c is the cost of encrypting/decrypting one item.

4.2.2.1.3 Query Processing Delay

Under the general term of query processing delays, we include delays for the basic SQL operations of Insert, Delete, Update, and Select.

1. **Insert**

It depends on the number of tuples to be inserted so the time complexity to insert to database is $O(n)$, where the n is the number of tuples.

2. **Delete / Update / Select**

In order to find the tuples affected by the operation, the tuples need to be identified. Delay thus depends on finding the tuples. Before we provide equations, we note that the delay to select/find tuples of a relation depends on whether or not there is a fast access data structure that can be exploited to find the desired tuples. Thus, (i) if there is no fast access method, all tuples are scanned and hence the delay is directly proportional to the number of tuples in the relation, which is modeled by $p\,n$, where n is the number of tuples in the relation and p is the delay to handle one tuple. If there is a fast access method, then the delay is proportional to log n, where n is the number of tuples, and it is modeled as p' log n. Once the tuples are identified, they are processed as per delete, update, or select operation that we assume causes equivalent/same delays. However, this processing delay is directly proportional to the number of affected tuples. We need to note that the above simplifications are reasonable only under the assumption that the Select queries are one-variable only, i.e. that they do not involve a join. As we deal with an evaluation in which we only have a single relation, that means that the assumption is that the query does not specify a self-join. Furthermore, the equation also does

not properly represent delays for an SQL query that has a Group-by operator that would result in sorting of resulting tuples.

Since our experimental relation is relatively small-sized, we do not build indices explicitly, and we model the Delete / Update / Select by equation 4.3 under the assumption that there is no fast access method available and hence the processing delay is directly proportional to the number of tuples in the relation:

one tuple.

$$D_p = c + p\,n + p\,s\,n \qquad \text{Equation 4.3}$$

In Equation 4.3, c is the overhead delay, n is the number of tuples in a relation, p is the delay to process one tuple, and s is the selectivity factor. The delay to process a query is thus modeled by overhead delay, c; delay of p n for scanning all tuples of the relations for those tuples that satisfy the predicate, which could be a conjunction of simple terms; delay of p s n to handle the result. .

4.2.2.1.4 Proxy Delays

The proxy, as stated in the previous chapter, is playing an important role. It performs several tasks that include: encryption, decryption, query parsing, and key management. The proxy parses the query and rewrites it into another query or queries depending on to which slaves sub-queries need to be sent. The number of sub-queries will be issued if there is a conjunction of simple terms. The time complexity for rewriting a query is modeled as O(1) – although there are a number of slaves, the number is small and fixed.

The proxy does both the encryption and the decryption so the delay is already modeled within a separate section dealing with crypto delays. However, the key management is one of the proxy duties, so its delay has to be included in the proxy overhead delay. The key management algorithm was described in detail in the implementation section. The time complexity is equal to a search in an array by the element's position, which is O(1) because the keys are chosen based on their position in the array. On the other hand, the proxy has to determine which encryption algorithm is needed to encrypt the query(s) value. Since we have a fixed number of the encryption algorithms, the delay to determine the encryption algorithm is also O(1).

The proxy also deals with the unions and intersects, as a result of querying more than one slave cloud in our method. We implement the union and the intersect algorithm by using the TreeSet build-in Java function which is based on the binary search technique. The retrieved results of each slave cloud will be stored on an array at proxy. The proxy performs the union and the intersect upon the two arrays. In the case there are more then two slave clouds involved, the result of intersect/union of first two arrays will be stored on a temporary array where the intersect with a third array can be performed and so on. Thus, the delay for an union is $O(n + n)$ and the intersect operation is $O(m \log n)$ where m is the size of the first array and n is the size of the second array. Equation 4.4 is used to model the proxy's delay:

$$D_x = ax + by + c \qquad \text{Equation 4.4}$$

In Equation 4.4, the variable "x" is the number of element in the first array, and "y" variable represent the size of the second array.

4.2.2.2 Determining the Parameters of the Model

To determine the values of the model's parameters, we perform the following set of experiments.

4.2.2.2.1 Communication Delays

For communication delays, we use the ping utility program to send a message of varying size between the proxy and a cloud in order to determine the parameters in Equation 4.1. We perform the experiment repeatedly in order to obtain an average and variance of parameters. The results are reported in table 4-4.

Table 4- 4: The communication delays in millisecond

Message Size	Minimum	Average	Maximum	Standard deviation
2008 byte	60.232	64.469	78.285	3.035
4008 byte	61.640	66.743	186.621	13.314
8008 byte	64.331	68.381	96.689	3.461
16008 byte	67.193	71.534	93.650	4.370
32008 byte	74.874	79.371	132.211	6.320

Columns show the minimum, average, maximum delays as well as the standard deviation of a message sent hundred times. The message delay is for a round-trip, i.e., it

includes the delay for sending a message from the proxy to a cloud and also the delay of the response message from the cloud to the proxy. Thus, we assume that the delay for sending a message of a certain size from the proxy to a cloud is the same as for sending the same message from the cloud to the proxy. The row headings identify the size of messages that were sent one after another.

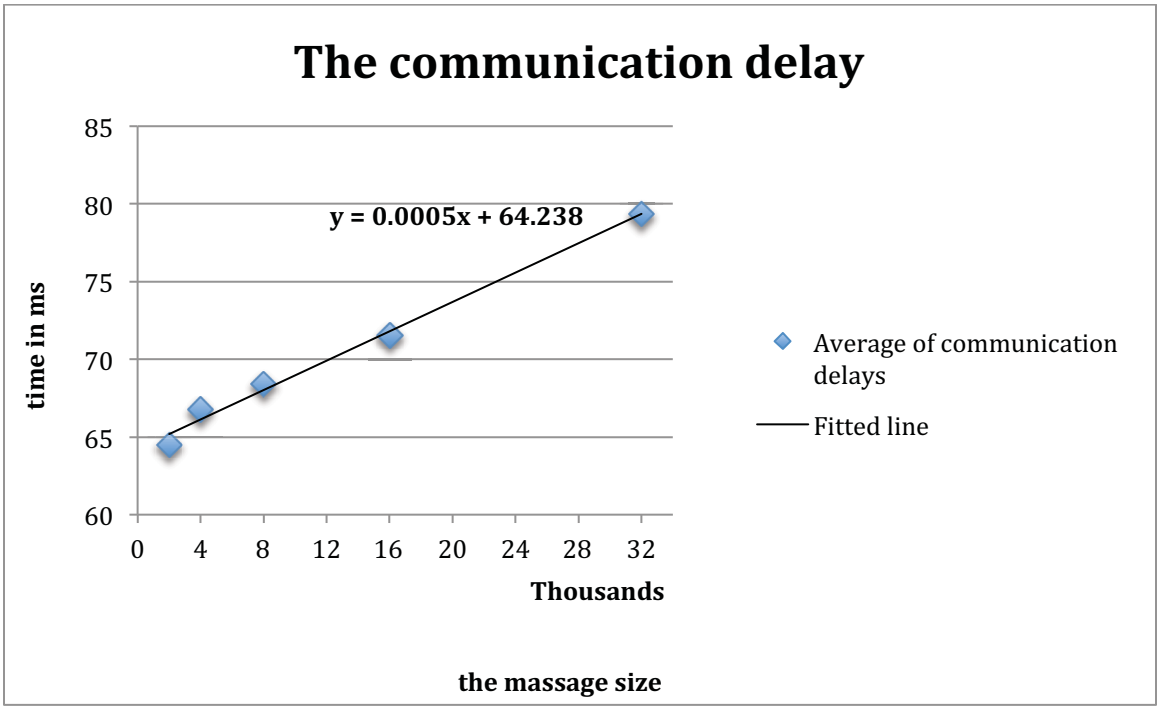


Figure 4- 2: Determining the communication delay parameter

Figure 4-2 shows the parameters that are used for Equation 4.1. The values of the constant "a" and "b" are 0.0005 and 64.238, respectively, and were determined using the standard line fitting method that minimizes the sum of the squares distances of points from the fitted line. We observe that the communication delay is 64 ms for a message of 2000 byte, and it is slightly increased with a message size due to a small value of the coefficient "a". We assume for simplification purposes in this thesis, that the communication delay of sending a message from the proxy to a cloud is the same as sending a message of the same size from the cloud to the proxy.

4.2.2.2.2 Query Processing Delays

We make an observation that in most online transaction processing relational DB systems, there is a general rule of thumb that states that most queries are of retrieval type (SQL Select

statements), while the rest are update queries (SQL Insert, Delete, and Update statements). For that reason and simplification on the volume of experimentation, we also concentrate on the Select statement. Recall that Insert operation is relatively simple in that its delay depends on the number of tuples to be inserted. It was also shown in the previous sub-section that the delays of the Delete and Update statements are similar to the delays of the Select statement.

To determine the parameters of the analytical model for the Select statement, we performed the following experiments. We use a relational DBMS on a cloud, and we issue Select statements that use the Where clause to vary the selectivity factors. The range of selectivity factors begins from 0.2 to 1.

We also vary the number of tuples in a relation and also the number of resulting tuples, i.e., the selectivity factor in a range clause. Note that for our modeling experimentation, we use the same configuration for all DBMSs, i.e., for DBMSs for the master and slave clouds. That is; the DBMSs run on servers such that their parameters used to configure them are the same, and the configuration parameters for the DBMSs are also the same. Hence, we just need to measure the query processing delays on one DBMS. We remotely log to the DMBS, perform the experiments (issue queries), and use the DBMS control panel to measure the processing delays. Our measurements are reported in table 4-5.

Table 4- 5: The query processing delays in millisecond

Query	Selectivity factors	Total number of tuples				
		500	1000	2000	4000	8000
Query1.1	0.2	2	3	7	13	27
Query1.2	0.4	3	5	9	17	33
Query1.3	0.6	3	7	13	23	38
Query1.4	0.8	4	9	14	26	59
Query1.5	1	5	10	19	39	72

Table 4-5 shows the query processing delay in millisecond. There is a positive relationship between the number of the tuples of the relation and the query processing delay.

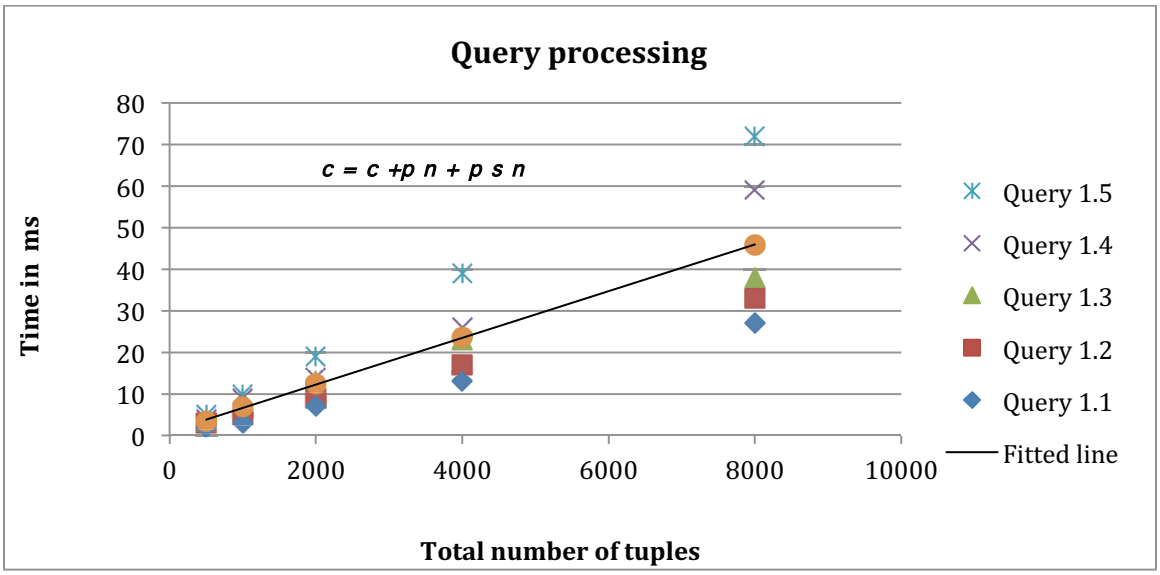


Figure 4- 3: The query processing delays

Figure 4-3 shows the delays that were presented in Table 4-5 and also shows the results of line fitting that minimizes distance square. Recall that *n* is the number of the tuples in the relation,s is the selectivity factor. This is what was expected as the selected tuples can be determined on one pass through the relation regardless of the type of a predicated (point query or a range query) or the number of terms as we do not use any two-variable query predicates that are processed by self-joins.

4.2.2.2.3 Crypto Delays

For crypto delays, we determine only the parameter of the AES-CBC encryption algorithm because it is the only encryption algorithm that is being used at the master cloud. Since the size of the plain-text record that needs to be inserted to the master cloud is much less than the same cipher record, we determine each of the encryption and decryption separately. For simplicity in determining the parameter values, we assume that the size of plain text record is 25 bytes while that it is 100 bytes for the corresponded cipher-text record. We determine the parameter by calculating the time to encrypt/decrypt by varying the size of message. Table 4.2 shows the crypto delays in milliseconds. We performed the experiments on an isolated system that has no other activities/program and hence did not observe variation in delays when the experiments were repeated. Thus, deviations are not reported.

Table 4- 6: The Crypto delays in millisecond

The number of retrieved tuples / Encryption\ Decryption	200	400	800	1600	3200
Encryption (1 tuple = 25 bytes)	261	266	275	267	292
Decryption (1 tuple = 100 bytes)	0.93	1.12	1.44	2.12	3.64

Figure 4-4 shows the delays that were presented in Table 4-6 and also shows the results of line fitting that minimizes distance square. One line is for encryption and one is for decryption. The observation about the parameters value is that the decryption delay takes much less than the encryption delay because the AES-CBC encrypts blocks only in a sequential manner. However, it decrypts the blocks in parallel where the blocks size is always 128 bits in AES.

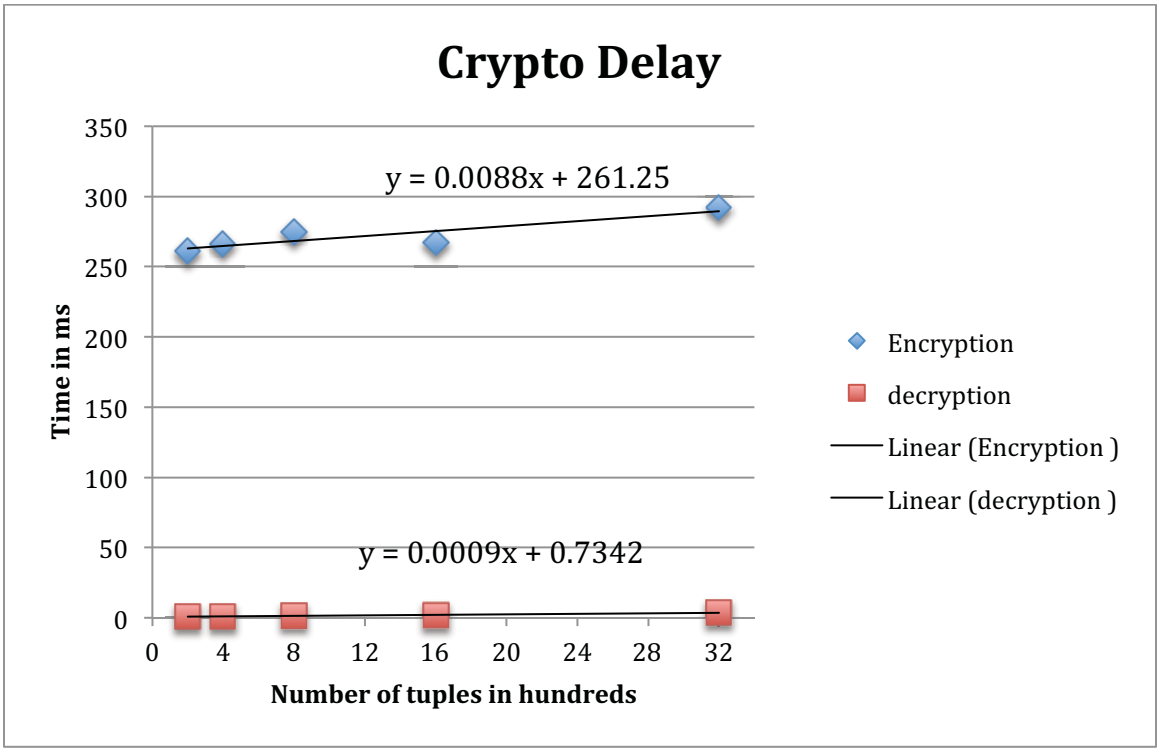


Figure 4- 4: Crypto delay parameters

4.2.2.2.4 Proxy Delays

Proxy performs encryption, decryption, and query translation. The encryption was discussed in previous sub-section. The proxy has to transform a query into a number of sub-

queries when predicates appear in the Where clause. It then sends each sub-query to a slave cloud that has the column of desired values on which a predicate applies. Since each of the slave clouds has only one vertical fragment (one column plus an index column), an intersect or union of the returning result needs to be performed by the proxy. We determine the parameters of Equation 4.4 by calculating the time in millisecond as we vary the size of the two arrays that hold the indexes returned by the slave clouds. Table 4.4 shows the time in millisecond for different sizes of the arrays. We determine the delay of the intersect or union by taking timestamp at the beginning and the end of the function. We also separate the number of elements equally between the arrays. For instance, in case of 2000 elements, we design a query that has returning results from two slave clouds in which each slave cloud has to return thousand elements.

Table 4- 7: The proxy delays in millisecond

# Elements in the arrays / The operation	2000	4000	8000	16000	32000
The Intersect	2	3	5	8	9
The union	3	6	10	14	17

Figure 4-5 shows the delays that were presented in Table 4-7 and also shows the results of line fitting that minimizes distance square. One line is for intersect and one is for the union.

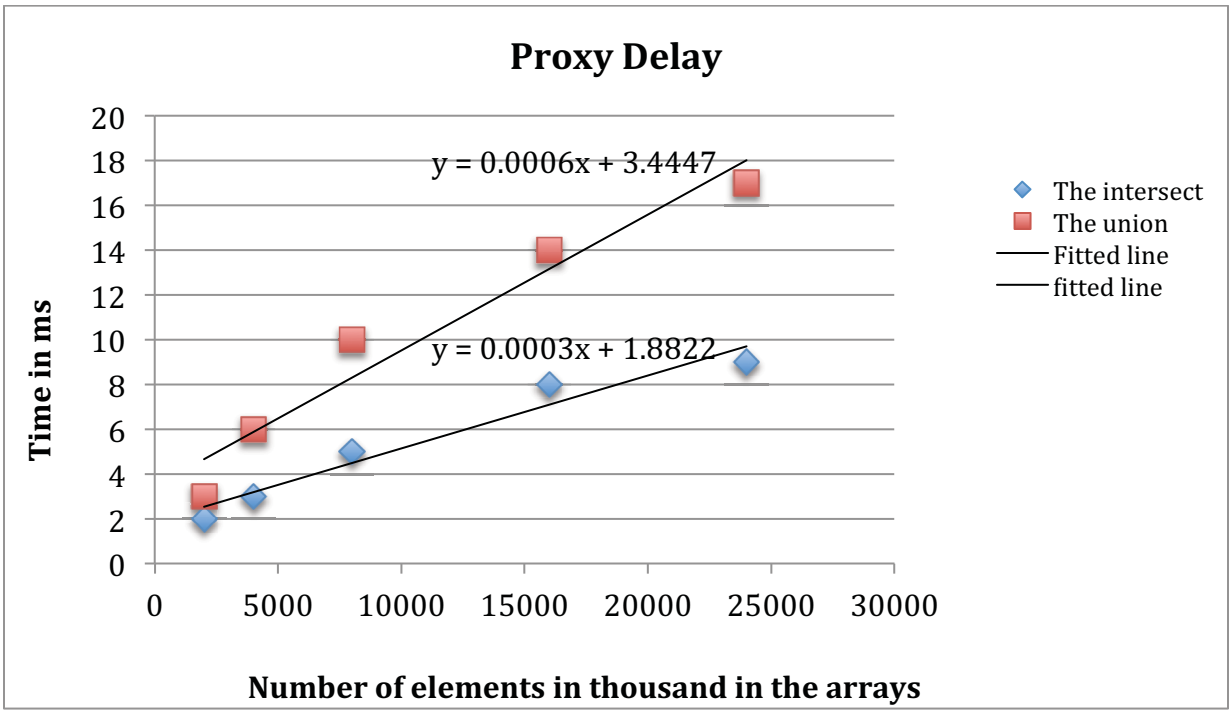


Figure 4- 5: proxy delay parameter

4.2.2.3 Observations

Based on determining the parameters of the analytical model, we observe that the communication delay is the highest delays, in case of small message size. The equation D=0.0005x+64.238 represent the communication delay, where x is the size of the message. See Figure 4-6 for comparison of the magnitude of parameters and the resulting delays in a Query in which the number of tuples in the database is fixed to 32000 tuples. The number of clouds involved is three clouds, including the master cloud, and the number of retrieved tuples is 32 tuples. The communication delay is due to the message size of 3200 bytes when each tuple is hundred bytes of the size. The number of the tuples (32000 tuples) is used as the query-processing variable. The number of retrieved tuples is also used to determine the crypto delay. Lastly, the intersect equation is used to determine the proxy delay since we assume a conjunction of predicates.

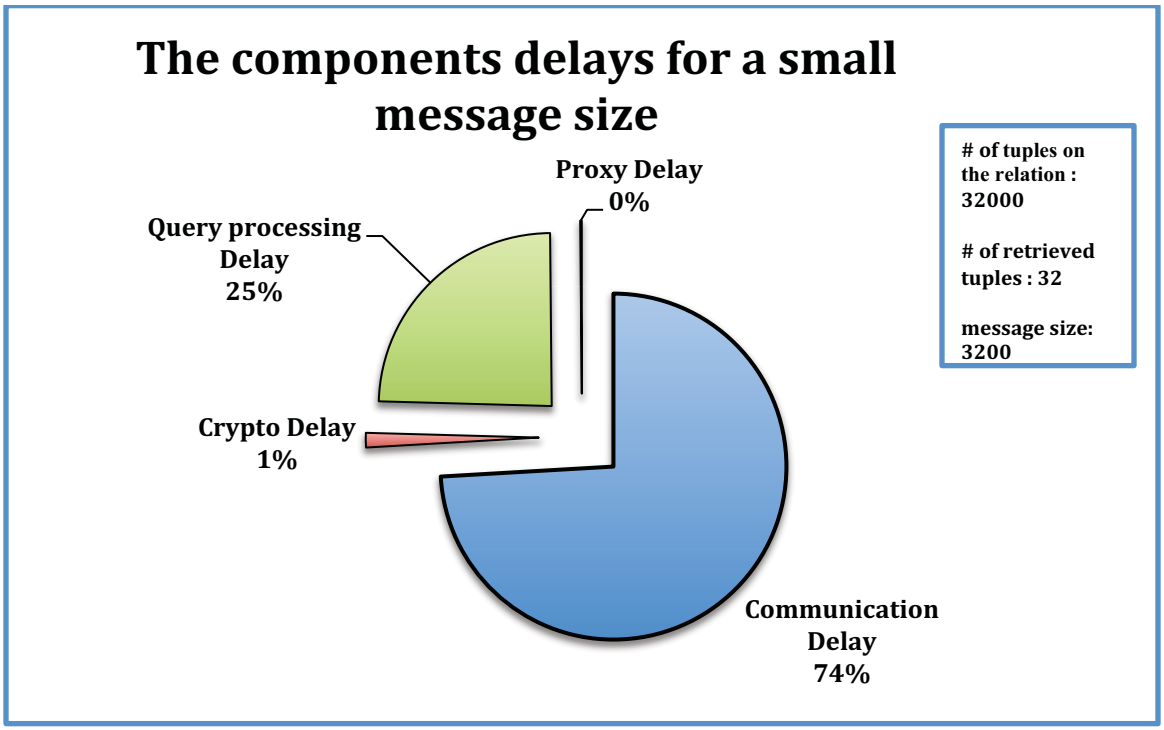


Figure 4- 6: The delays of different components for a small message size

Figure 4-6 shows the delays in percentages that are determined by the analytical model. If we examine the resulting parameters, we can observe that the communication delays are dominant by far. Query processing delays are next but much smaller than the communication delays. The proxy and crypto delays are very small in comparison.

On the other hand, we did the same calculation for a larger message size (see figure 4-7). The result shows that the crypto delay is higher than the communication delay by 19%. Therefore, our scheme is affected by the message size in that if the message size is large then the crypto delay is the highest, and if the message size is small then the communication delay is highest. Examining the equations 4-1, used for communication delay, and equation 4-2, used for decryption delays, the decryption has a higher slope but communication has a higher fixed overhead.

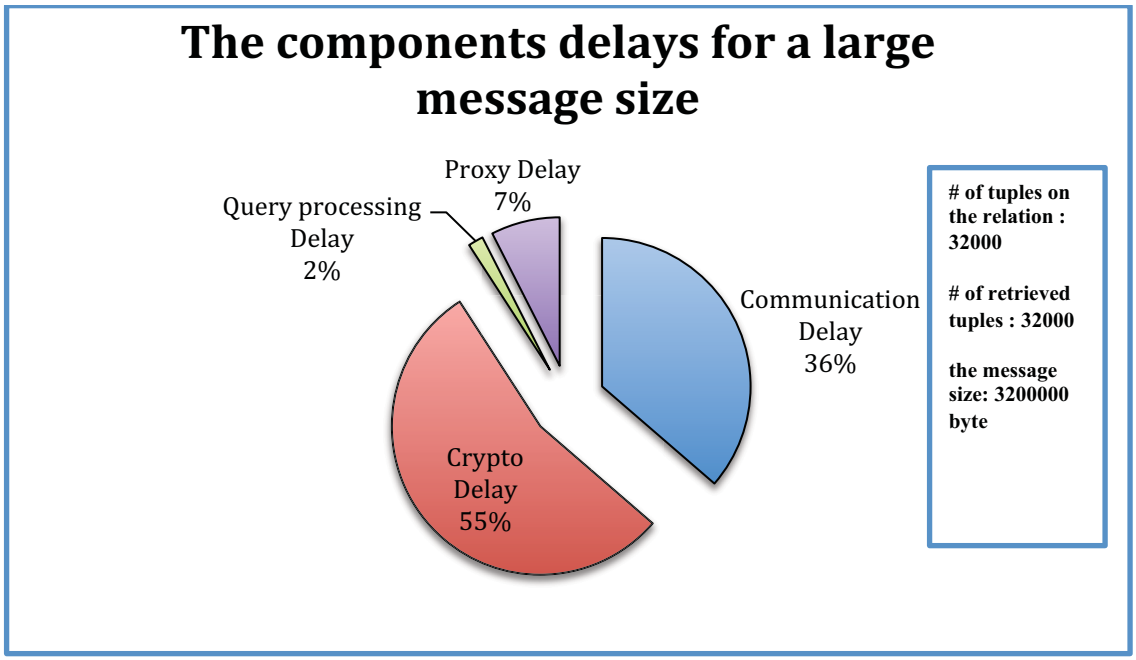


Figure 4- 7: The delays of different components for a large message size

4.2.3 Delays Derived Using the Analytical Model

We are now ready to report on delays predicted by the analytical model. We explore two scenarios for our Secure Distributed method, one is serial and one is parallel. That is, if a user query results in more than one sub-query sent to the clouds in which columns are stored, we analyze two cases: (a) The sub-queries are issued in a serial fashion, one after another, and (b) sub-queries are issued in parallel, i.e., concurrently. Once the results from sub-queries are retrieved, the proxy determines the intersection of tuples, of the results, based on the returned indices - this intersection is then sent to the master to get the query answer. We report on delays obtained by applying the analytical model on the queries described below.

Based on the analytical model, we compare four approaches: Unsecured centralized, Secure centralized, Secure distributed serial, and the Secure distributed parallel. We calculate the total delays of each of these approaches. We calculate the delays using the analytical model while varying the selectivity to vary the number of retrieved tuples and also the number of predicates. In all cases the DB size is fixed at 8000 tuples.

Table 4-1 shows the first set of queries, which have only one predicate. Recall that the results are retrieved from two clouds: one slave cloud and the master cloud. Thus, we apply the same selectivity factor when querying the cloud and also when querying the master cloud.

Table 4- 8: The total delays in millisecond of four schemes for Query 1

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial				Secure distributed parallel			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy	Comm.	Query pro	Crypto	Proxy
0.2	134		339			445				445			
	84	50	144	50	144	211	77	144	2	211	77	144	2
0.4	160		569			674				674			
	104	56	224	56	288	282	82	288	2	282	82	288	2
0.6	186		798			904				904			
	124	62	304	62	432	378	88	432	3	378	88	432	3
0.8	211		1028			1134				1134			
	144	67	384	67	576	461	94	576	4	461	94	576	4
1	237		1258			1363				1422			
	164	73	464	73	720	928	99	720	4	928	99	720	4

Table 4-8 shows the delays in millisecond for Query 1, which has a Where Clause with one predicate. We also vary the selectivity factors (S.F.) for the master cloud when forming the results. The Secure distributed serial and parallel have the same delay since, in this Query, there is no parallelized process. The delays for the Secure Centralized method are less than for the Secure Distributed (serial and parallel) methods because in the centralized case only the master cloud is accessed while the distributed methods, in additional to accessing the master cloud, also accesses one slave cloud.

The next figures show the components contribution on the total delays for Query 1. Figure 4-8 shows the delays of the communication and the query processing for unsecured centralized approach. In this approach, there are two components that have an impact of the total delay, which are the communication and the query processing. The communication delay is higher than the query processing delay.

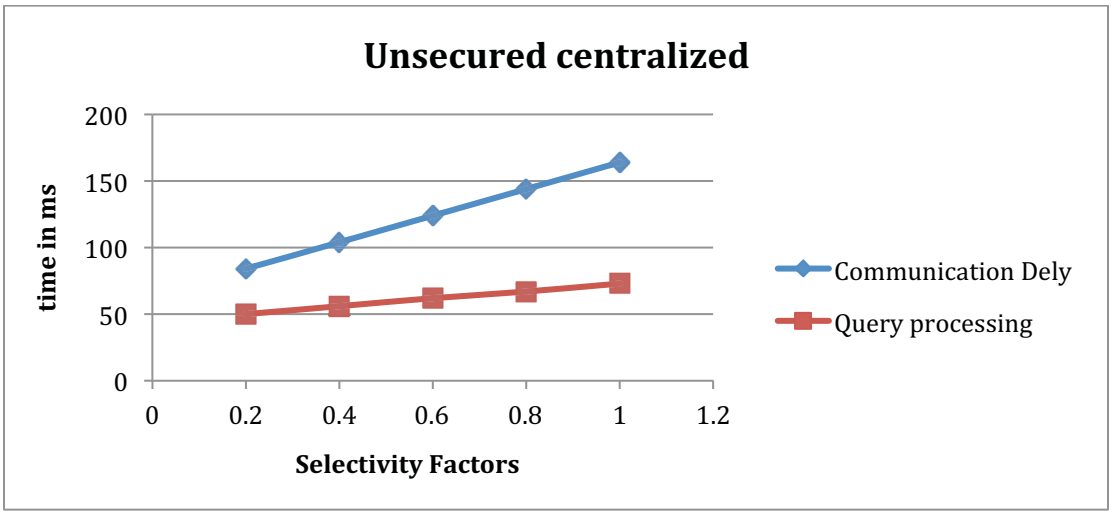


Figure 4- 8: Component delays for Unsecured centralized for Query 1

Figure 4-9 shows the delays of the components for secure centralized approach for Query 1. In this approach, there is one more component that has an impact on the total delay and that is the crypto delay. In fact, the crypto delay is higher than other components when the selectivity factor is 0.4 or greater.

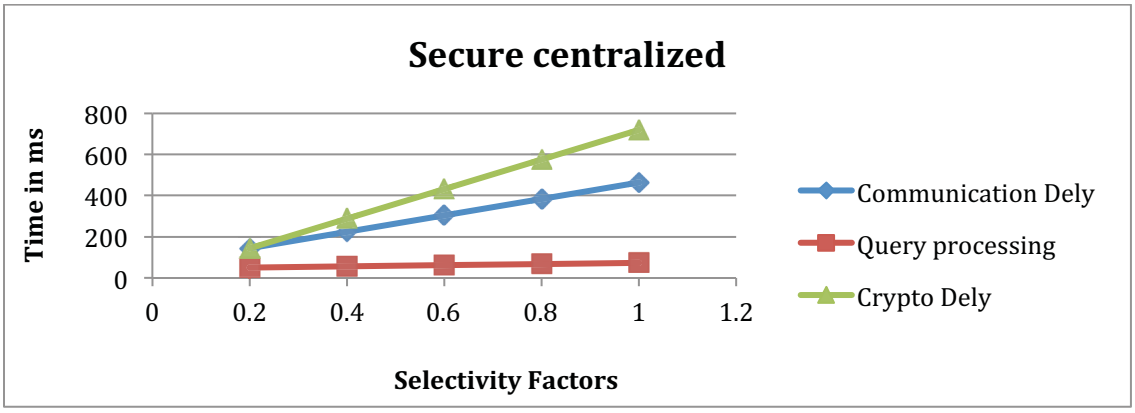


Figure 4- 9: The component delays for secured centralized for Query 1

Figure 4-10 shows the delays of the components for secure distributed serial approach. There are four components that have a major impact on the total delay. Recall that there are two clouds involved to retrieve the final results, the master cloud and a slave cloud. The communication delay is the highest delay when the selectivity factors are less than 0.3. The communication and the crypto delays are intersecting at 0.4 selectivity factor where the crypto delays become higher than the communication delay. On the other

hand, the proxy and query processing delays are almost the same but they are less than others two components as the selectivity factor increases.

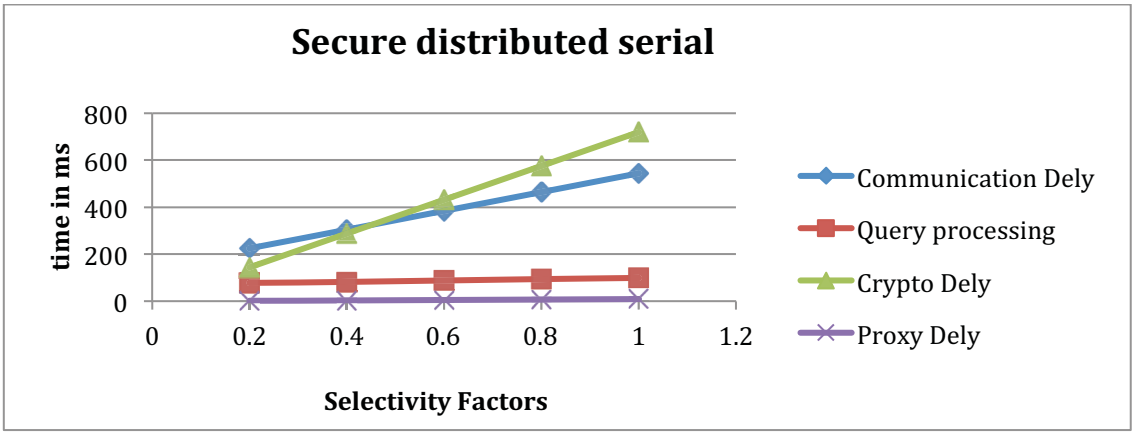


Figure 4- 10: The components delays for secure distributed serial for Query 1.

Figure 4-11 shows the delays of the components for secure distributed parallel approach for Query 1. This approach is different than the previous approach in the way of how the slave clouds are queried. However, in this query processing the parallelism has no effect because we have only one predicate meaning one slave cloud to query and thus the parallel and serial approaches are the same.

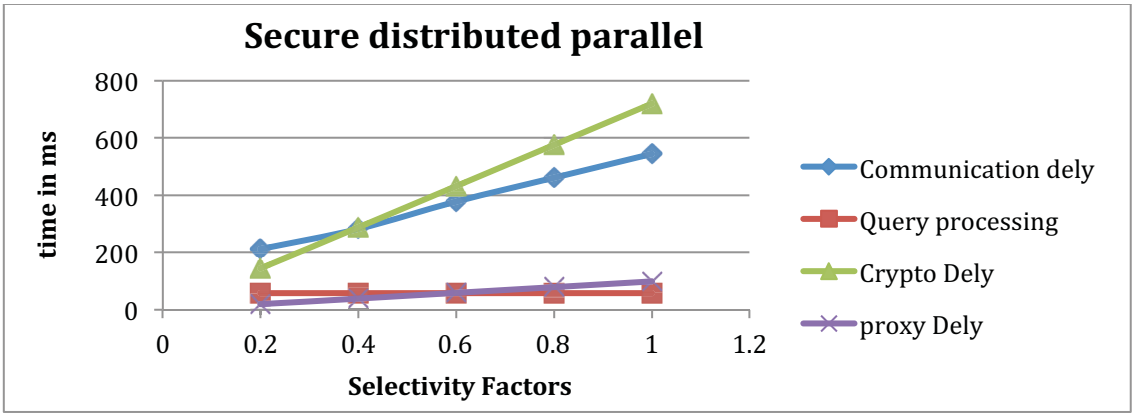


Figure 4- 11: The components delays for secure distributed parallel for Query 1

In Query 2, there is one more predicate in the Where clause, which means that one more slave cloud is involved in our scheme. We vary the final results selectivity factors so that it increases by 0.2. Slave clouds selectivity factors are also reported and are always greater than

the master selectivity factors because their product will determine the selectivity factor for the final result retrieved from the master cloud. We use the usual assumptions of uniform distribution of values in columns and also of independence of values across columns.

Table 4-2 shows the Query 2 that have two predicates, and each slave cloud has different selectivity factors.

Table 4- 9: The total delays in millisecond of four schemes for Query 2

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial				Secure distributed parallel			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy	Comm.	Query pro	Crypto	Proxy
0.2	134		338			536				469			
	84	50	144	50	144	288	103	144	1	220	103	144	2
0.4	160		568			845				740			
	104	56	224	56	288	445	109	288	3	302	109	288	3
0.6	186		798			1008				931			
	124	62	304	62	432	457	114	432	5	381	114	432	4
0.8	211		1027			1244				1163			
	144	67	384	67	576	541	120	576	7	462	120	576	5
1	237		1257			1479				1396			
	164	73	464	73	720	624	126	720	9	544	126	720	6

Table 4-9 shows the total delays in millisecond across the approaches. Recall that in the Secure distributed parallel approach, there are two communications: parallelized slave clouds queries and the master cloud query. Thus, as there is more than one predicate in comparison to the previous Query, the delays for the Secure distributed serial method are increased.

Figure 4-12 shows the components delays of unsecured centralized approach. The communication delay is higher than query processing delay by far and there are neither crypto nor proxy delays in this approach. Hence the total delay is less than for the other methods.

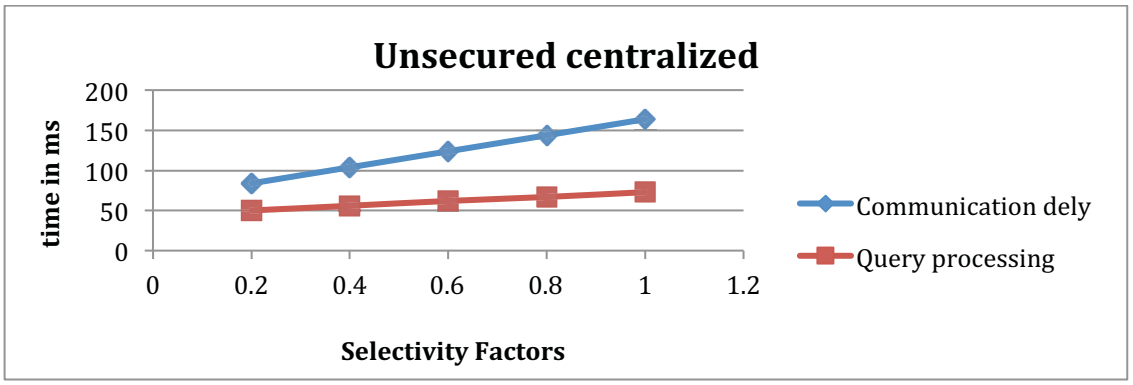


Figure 4- 12: The components delays for Unsecured centralized for Query 2

Figure 4-13 shows the delays of the major components of secure centralized approach. The crypto delays are the highest in most of the selectivity factor points especially if it is greater than 0.4. The communication delay is a bit higher than the crypto delay at 0.2 selectivity factors.

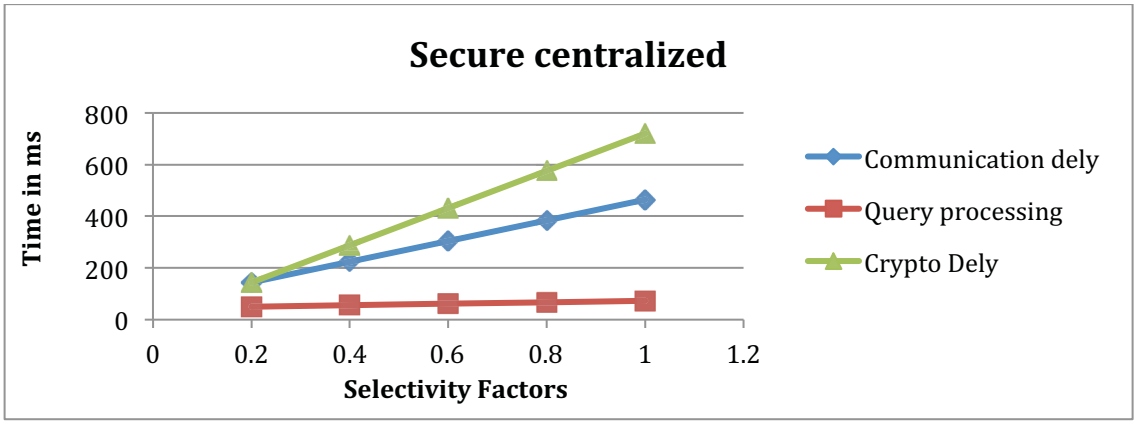


Figure 4- 13: The components delays for secured centralized for Query 2

Figure 4-14 shows the communication, query processing, crypto, and proxy delays of Secure distributed serial approach. Before the selectivity factor of 0.6, the communication delays are higher than others because there is more slave clouds are accessed to form the final results. Recall that the selectivity factors for each of the slave clouds are greater than the final results selectivity factors. Thus, the selectivity factors have an impact on the total communication delay. The crypto delays become the highest after selectivity factor of 0.8. The query processing and the proxy delays are much less than the others delays.

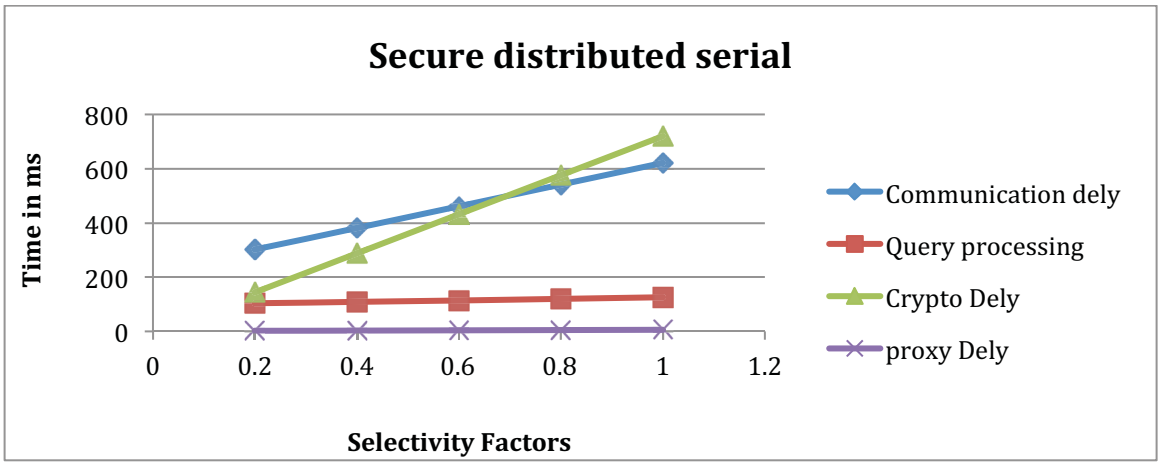


Figure 4- 14: The components delays for secure distributed serial for Query 2

Figure 4-15 shows the delays of the components in the secure distributed parallel approach. The communication delays are the highest until the 0.4 selectivity factor. In general, the communication delays are less in comparison to the communication delays on the secure distributed serial approach because the slave clouds are queried in parallel. However, crypto delays are the same in both approaches.

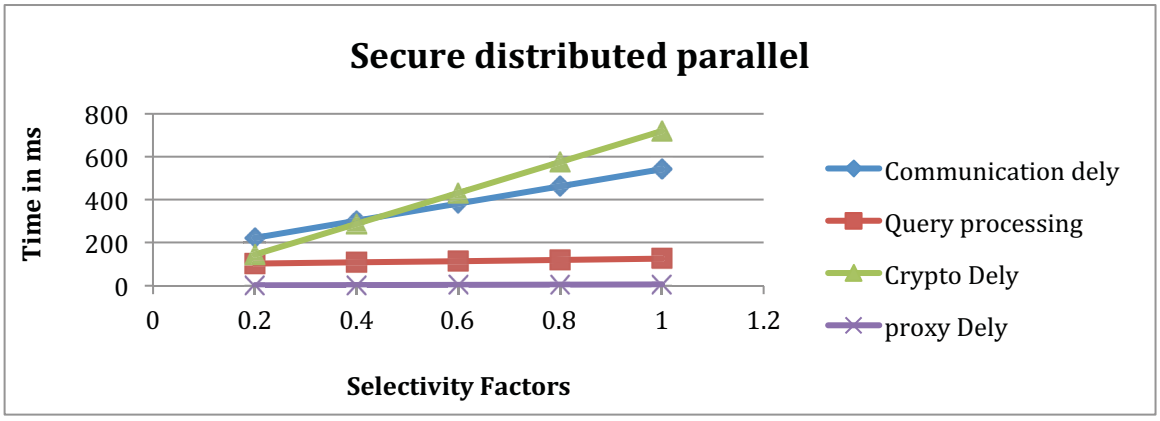


Figure 4- 15: The components delays for secure distributed parallel for Query 2

Table 4- 10: The total delays in millisecond of four schemes for Query 3

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial				Secure distributed parallel			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy	Comm.	Query pro	Crypto	Proxy
0.2	134		339			634				497			
	84	50	144	50	144	357	130	144	3	220	130	144	3
0.4	160		569			879				729			
	104	56	224	56	288	452	135	288	4	302	135	288	4
0.6	186		798			1108				960			
	124	62	304	62	432	529	141	432	6	381	141	432	6
0.8	211		1028			1350				1111			
	144	67	384	67	576	620	147	576	7	463	147	576	7
1	237		1258			1585				1425			
	164	73	464	73	720	704	152	720	9	544	152	720	9

Table 4-10 shows the delays in millisecond of the four approaches for Query 3, which has three predicates. The delays are increased in our scheme because one slave cloud is accessed for each of the predicates. The communication delay, as we stated previously, dominates the other factors that can impact the total delay.

Figure 4-16 shows the communication and query processing delays for unsecured centralized approach. The communication delays increased as the selectivity factor is increased - this is because the communication delay is proportionally effected by the message size so that the bigger the message size, the higher the delay.

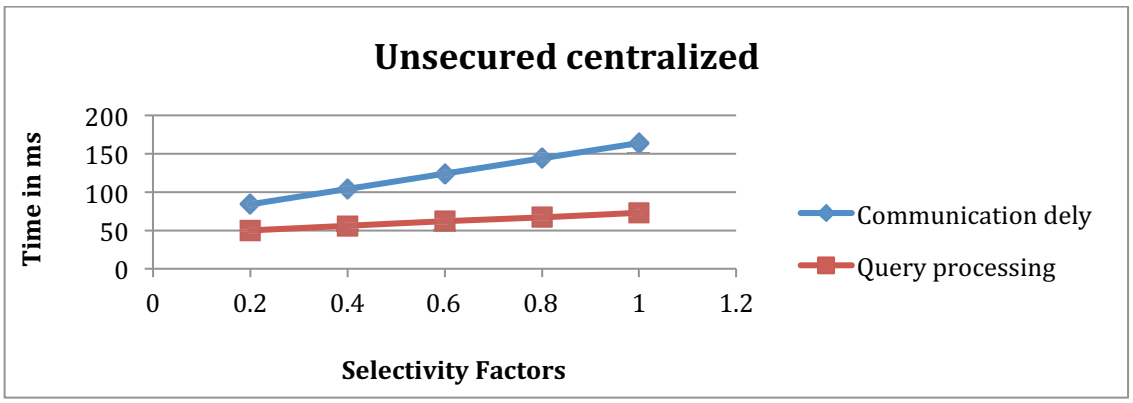


Figure 4- 16: The components delays for Unsecure centralized for Query 3

Figure 4-17 shows the delay of three major components that impact the total delays of the secure centralized approach. The communication and the crypto delay are higher than the query processing delay. The variances between these two components increase as the selectivity factor increases.

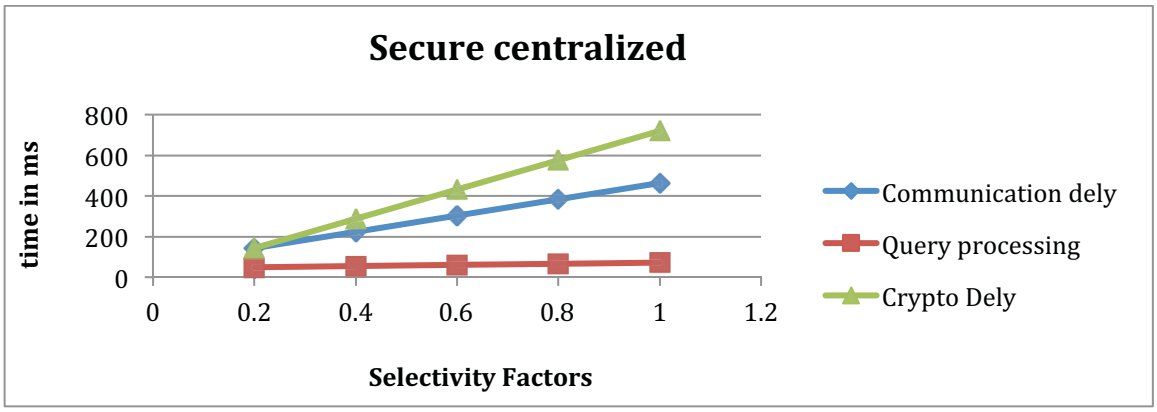


Figure 4- 17: The components delays for Secure centralized for Query 3

Figure 4-18 shows the delays of the different components for secure distributed serial approach for Query 3. The communication delays have the highest delays across the selectivity factors except the selectivity factor of one where the communication and the crypto delays are the same. The communication delays are higher than in the previous Query because of the increases of the number of the slave clouds to perform the query.

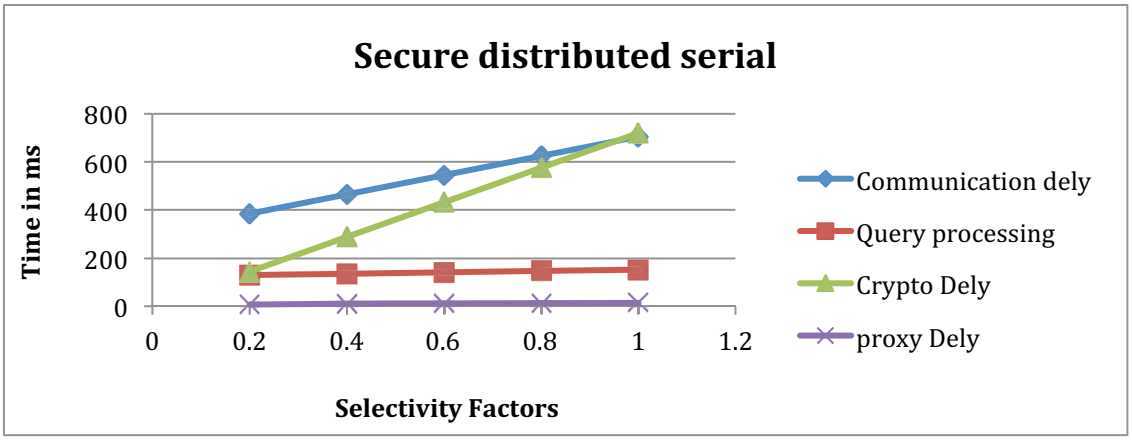


Figure 4- 18: The components delays for secure distributed serial for Query 3

Figure 4-19 shows the delays of components of secure distributed parallel approach for Query 3 . The delays end with the highest delay for crypto delay. The communication and crypto delays are intersecting at 0.5 selectivity factor. The crypto delay keeps increasing ending with the largest delay.

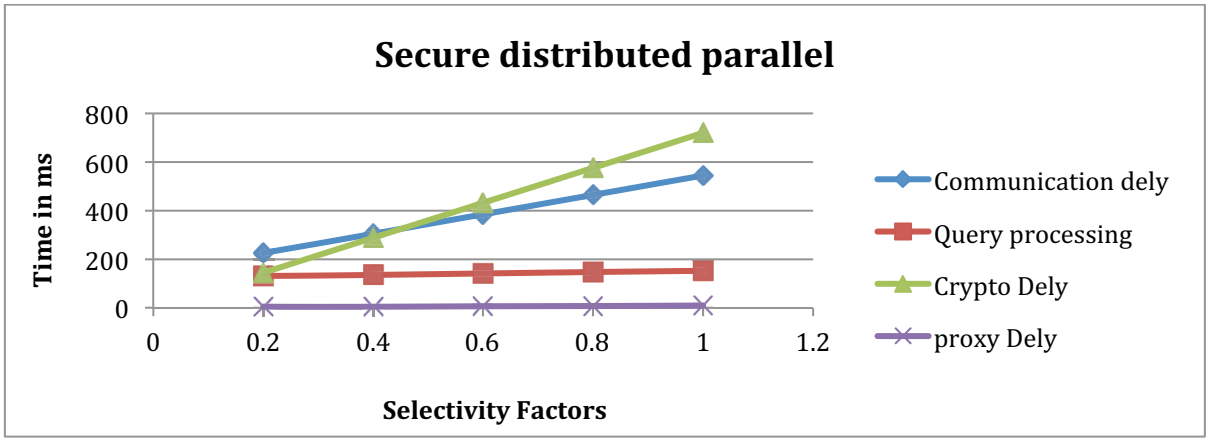


Figure 4- 19: The components delays for secure distributed parallel for Query 3

4.2.4 Evaluation Using Modeling

We created a proxy for each of the methods the Unsecured, Secured Centralized, and Secured Distributed methods. For each method, we instrumented the queries described in the section 4.2 and then measured and reported the component and total delays. We first describe how we measured the component delays and then we report on experimentation and its results.

It should be noted that for simplification, in order to reduce the instrumentation task, we use only one cloud that is used both for the master cloud and also the clouds storing individual columns. Furthermore, we also model in the Secured Distributed method only the case wherein individual sub-queries are issued by the proxy in a serial fashion. We did not instrument a proxy with concurrent processing in order to be able to send sub-queries to the clouds concurrently.

Communication between the cloud and the proxy was over Internet and thus we are confident that the bandwidths on the local ports of the proxy and the cloud were sufficient to attain the maximum bandwidth attainable using Internet. For the instrumentation, we used the following systems: First, we created a public cloud computing account at Rackspace, which offers open source cloud computing. Then, we created a number of servers equal to the number

of fragments that we need, plus one more for the master cloud. All servers in the cloud were configured with the following features:

- Linux OS, XAMPP Server
- CPU: 2 vCPUs
- RAM: 4 GB
- System Disk: 160 GB
- Network: 400 Mb/s

Both the master cloud and the salve clouds are configured using the Linux operating system. They are also configured using the XAMPP server to house MySQL DBMS. This tool is needed to store a master relation and the fragments in the nodes.

The proxy, on the other hand, is a server at the private cloud and have the following features:

- OS X
- Tomcat server 7.0.53
- CPU: 2.4 GHz Intel Core i5
- RAM: 10 GB RAM
- System Disk: 500 GB
- Network: 150 Mbps

4.2.4.1 Determining Component Delays

To measure the component delays we utilize timestamps. We acknowledge that there may be a skew between the clock of the proxy system and the clock of the cloud virtual system, which might have an impact when calculating overall delays. However, considering the magnitude of the measured delays, particularly over the network, the clock skew should be insignificant. To measure the component delays we take a timestamp at the beginning of an operation and a timestamp at its conclusion. Thus, to measure the communication delay, we take a timestamp just before sending a message and then send the message. Upon the reception of the result, we immediately take the timestamp, and we calculate the difference between the two time stamps and use it as a message delay. Similarly, we take a timestamp before any encryption, decryption, and query issuance, and also the timestamp at the conclusion of the operation and use them to calculate the delay.

4.2.4.2 Delays Derived Modeling

We measure the total delays while varying the selectivity factor and the number of predicates in the Where clause. The selectivity factors that are used are 0.2, 0.4, 0.6, 0.8, 1.0 and we hold the total number of tuples in the relation to 8000. We start the experiments with a query that has a Where Clause with one predicate then we increase the number of predicates by one up to three predicates. We submit the queries hundred times and report the averages. The experiments are applied for the methods; Unsecured Centralized, Secured Centralized, and Secured Distributed methods.

In the Query 1, as shown in table 4-5, the query has only one predicates indicating that the number of clouds is increased resulting on an increase on the total delays.

Table 4- 11: The total delays in millisecond of three schemes for Query 1

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy
0.2	130		451			777			
	106.2	23.8	253	30.5	168	566	41	169	1
0.4	115		585			1147			
	84.8	30.2	314	49	222	881	74	191	1
0.6	143		804			1492			
	102.9	40.1	495	62.8	247	1168	103	221	2
0.8	152		983			1757			
	102.9	49.1	593	83	307	1328	142	285	2
1	171		1287			2073			
	115.2	55.8	790	98.7	390	1580	148	343	2

Table 4-11 shows the total delays in milliseconds of the three approaches when the selectivity factor is varied from 0.2 to 1. Recall that the database on all approaches has 8000 records. Take the selectivity factor of 0.2 an example, then the number of retrieved tuples is 1600 tuples. The query only has one predicate where the master cloud and one more slave cloud are involved in order to complete the task in the Secure Distributed method. The results show the impact of the communication delays of the extra round trip in the Secure Distributed method in comparison to the Secure Centralized method. Of course, the Unsecured method has much less delays not only because there is no decryption process after retrieved the result but

also the information is stored in clear text, which means that the tuples are smaller in size resulting in faster communication.

Figure 4-20 shows the delays for communication and query processing in an unsecured centralized method. There is a similarity between the results of unsecured centralized in the analytical model and the modeling. The communication delays are higher than the query processing delays.

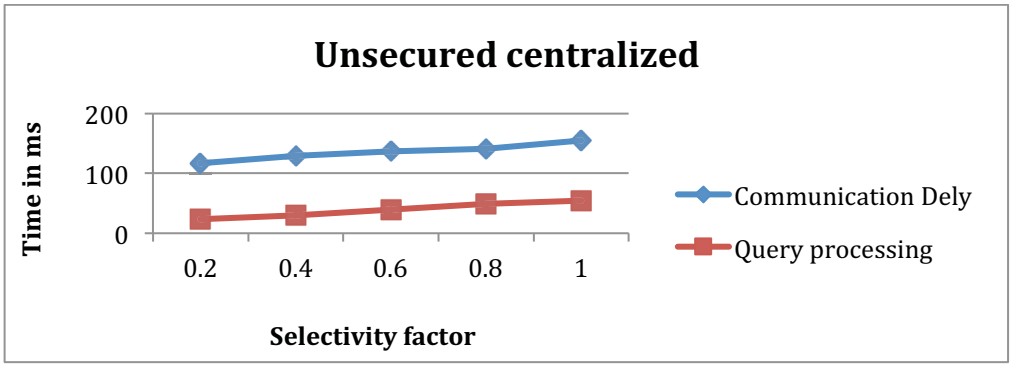


Figure 4- 20: The components delays for secure distributed serial for Query 1

Figure 4-21 shows the delays of the communication, query processing, and crypto delays for secure centralized approach. Unlike the corresponding result of the analytical model, the communication delays are the highest on all selectivity factors.

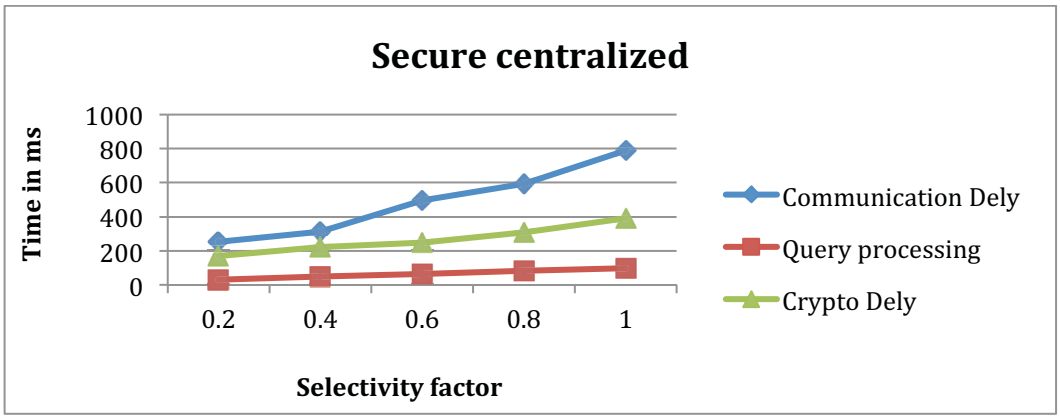


Figure 4- 21: The components delays for secure centralized for Query 1

Figure 4-22 shows how the delays of the components of secure distributed serial method. The communication delays are the highest, and the increase as the selectivity factors increase.

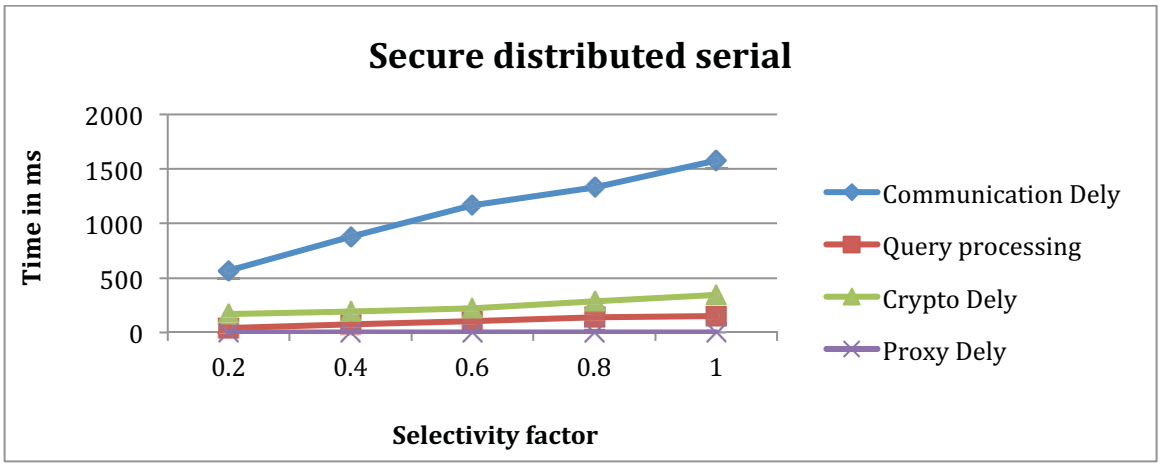


Figure 4- 22: The components delays for secure distributed serial for Query 1

The Query 2 have two predicates so that two slave clouds have to be queried. The selectivity factors for the slave query are larger than the desired final selectively factors.

Table 4- 12: The total delays in millisecond of three schemes for Query 2

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy
0.2	153		493			976			
	128	25	283	42	168	733	79	163	1
0.4	160		710			1459			
	107	35	396	93	221	1162	104	191	2
0.6	185		884			1707			
	143	42	536	109	239	1347	139	219	2
0.8	200		972			2165			
	149	51	549	113	310	1694	179	290	2
1	210		1582			2223			
	151	59	1036	139	407	1687	187	346	3

Table 4-12 shows the total delays in milliseconds for Query 2. In this Query, the number of predicates is increased to two predicates meaning that the number of clouds is increased to three clouds. The effect of the communication is increased by far.

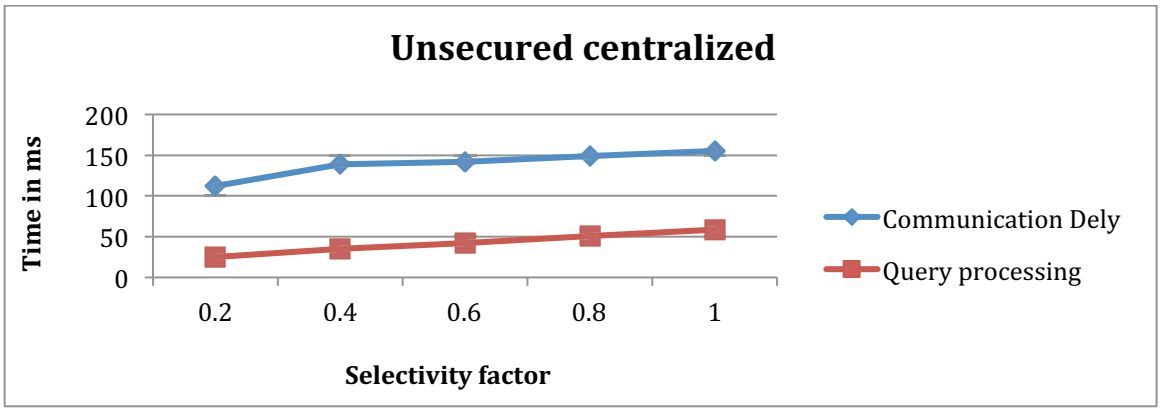


Figure 4- 23: The components delays for unsecured centralized for Query 2

Figure 2-23 shows the delays of communication and query processing delay, as they are the two major delays that have the impact in an unsecured centralized system. The communication is the higher by far than query processing.

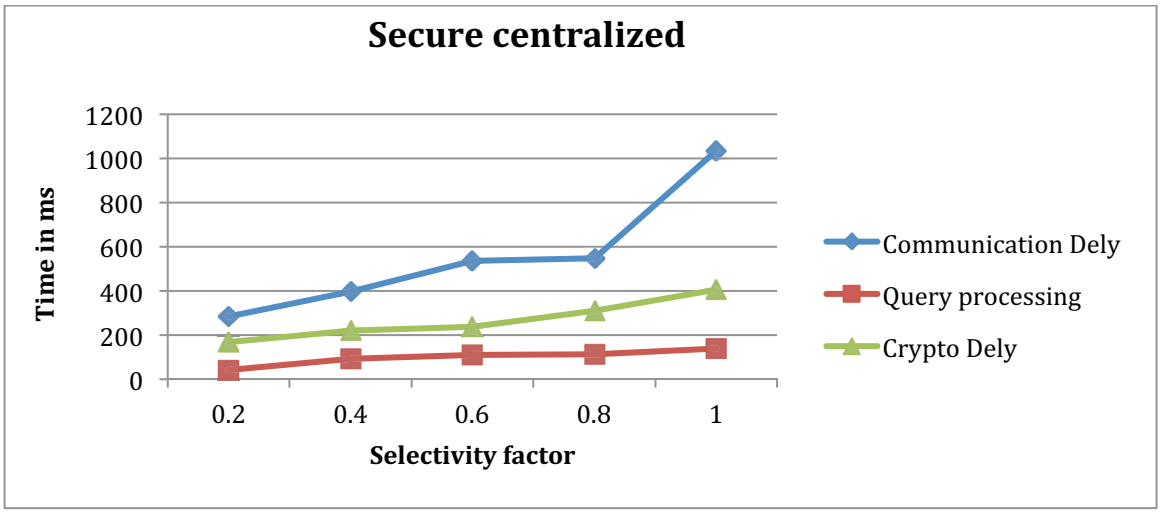


Figure 4- 24: The components delays for secure centralized for Query 2

Figure 4-24 shows the component delays that have an impact on the secure centralized approach. The communication is the highest delay because there is a number of factors effect the communication delays include the bandwidth, the Internet speed, the distances and many others that we can not control.

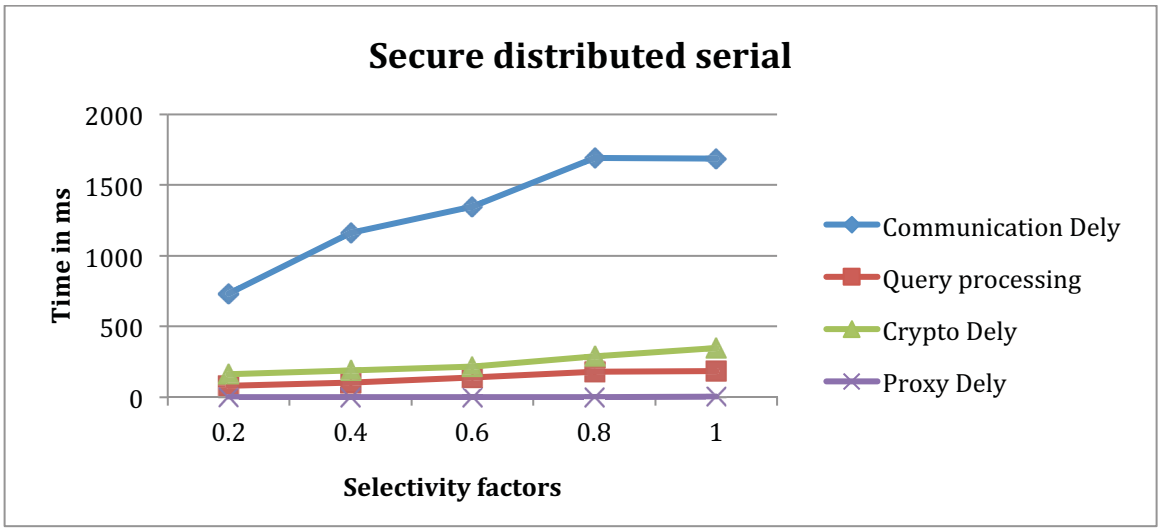


Figure 4- 25: The components delays for secured distributed serial for Query 2

Figure 4-25 shows the delays of the secure distributed serial system. In Query 2, the communication delays are much higher because there are two slave clouds that need to be queried before querying the master cloud.

In Query 3, there are three predicates in each query and hence three slave clouds are accesses. Recall that the selectivity factor represents the final selectivity factors where each of slave clouds has greater or equal selectivity factor. Therefore, any increase of the final result selectivity factor results in increase of the selectivity factor for each of the slave. Table 4-3 shows the queries of Query 3 with the selectivity factors.

Table 4- 13: Delays in millisecond of three schemes for Query 3

S.F.	Unsecure centralized		Secure centralized			Secure distributed serial			
	Comm.	Query pro	Comm.	Query pro	Crypto	Comm.	Query pro	Crypto	Proxy
0.2	155		504			1196			
	129	26	277	62	165	939	91	165	1
0.4	175		660			1729			
	138	37	336	101	223	1417	122	188	2
0.6	207		775			1950			
	164	43	429	109	237	1580	159	209	2
0.8	273		1194			2291			
	220	53	777	115	302	1805	203	281	2
1	220		1441			2763			
	159	61	933	137	371	2196	213	350	4

Table 4-13 shows the total delays in milliseconds for Query 3. In this Query, the number of predicates is increased to three predicates meaning that the number of clouds is increased to four clouds i.e. three slave clouds and the master cloud. The effect of the communication is increased by far.

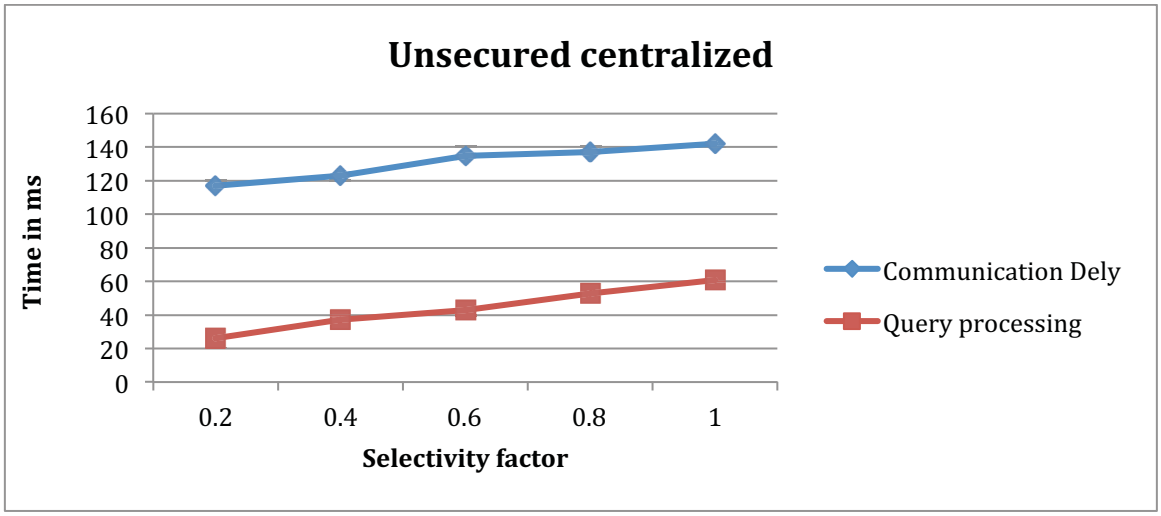


Figure 4- 26: The components delays for unsecured centralized for Query 3

Figure 2-26 shows the delays of unsecured centralized components; the communication and the query processing. There is no difference between the delays of the unsecure centralized approach in three the Queries.

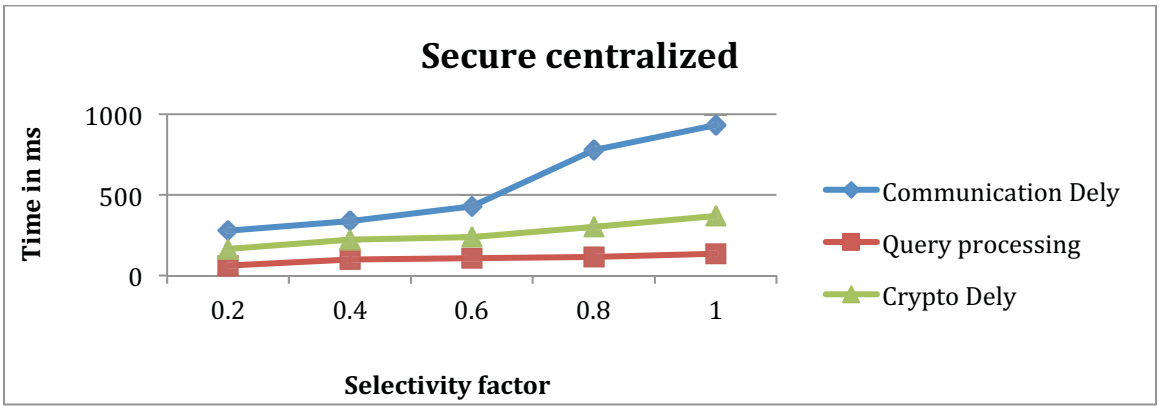


Figure 4- 27: The components delays for secure centralized for Query 3

Figure 2-27 shows the delays of the secure centralized approach. The communication delays are the highest. The crypto delays and the query processing delays are next.

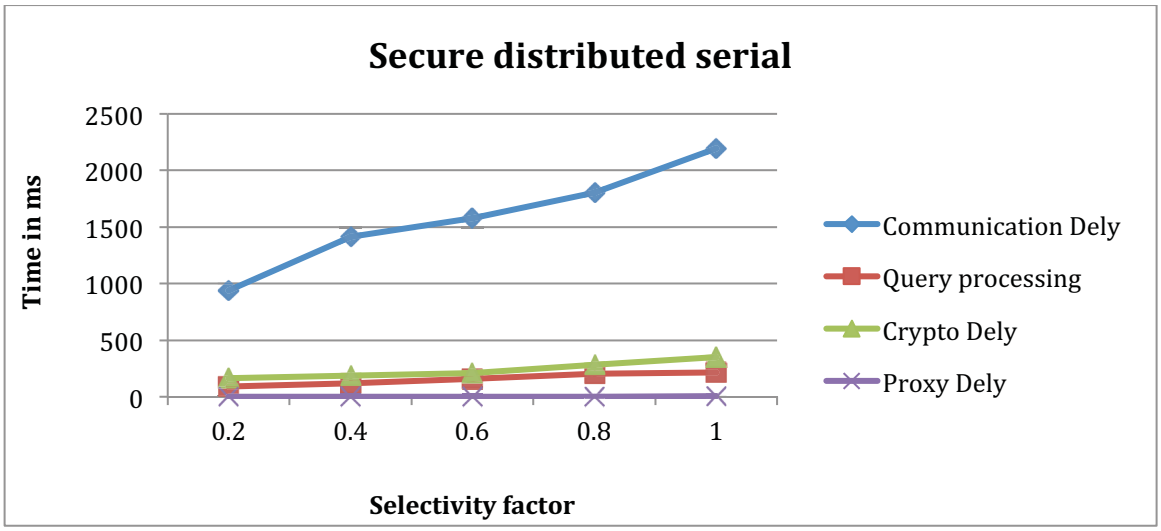


Figure 4- 28: The components delays for secure distributed serial for Query 3

Figure 4-28 shows how the communication delays affect the total delays. Recall that the communication is going through the Internet, so we have no control over it. As we stated, there

are many factors that are involved on the communication so that we cannot estimate the exact delays.

4.2.5 Results Analysis and Discussion

In the previous section, we report the delays of the components for each approach. In this section, however, we are comparing the delays of each component on the analytical model with the corresponding delays of the emulation modeling across all queries for every approach.

4.2.5.1 Communication delay

Figure 4-29 shows the communication delays for all queries in both models for unsecured centralized approach. The communication delays have the same delays in the analytical model in all the queries because the message size is the same between the queries; and the same applies for emulation modeling. Delays derived through emulation modeling, in comparison to those derived through analytical modeling, are somewhat lower at smaller selectivities, while somewhat lower at high selectivities.

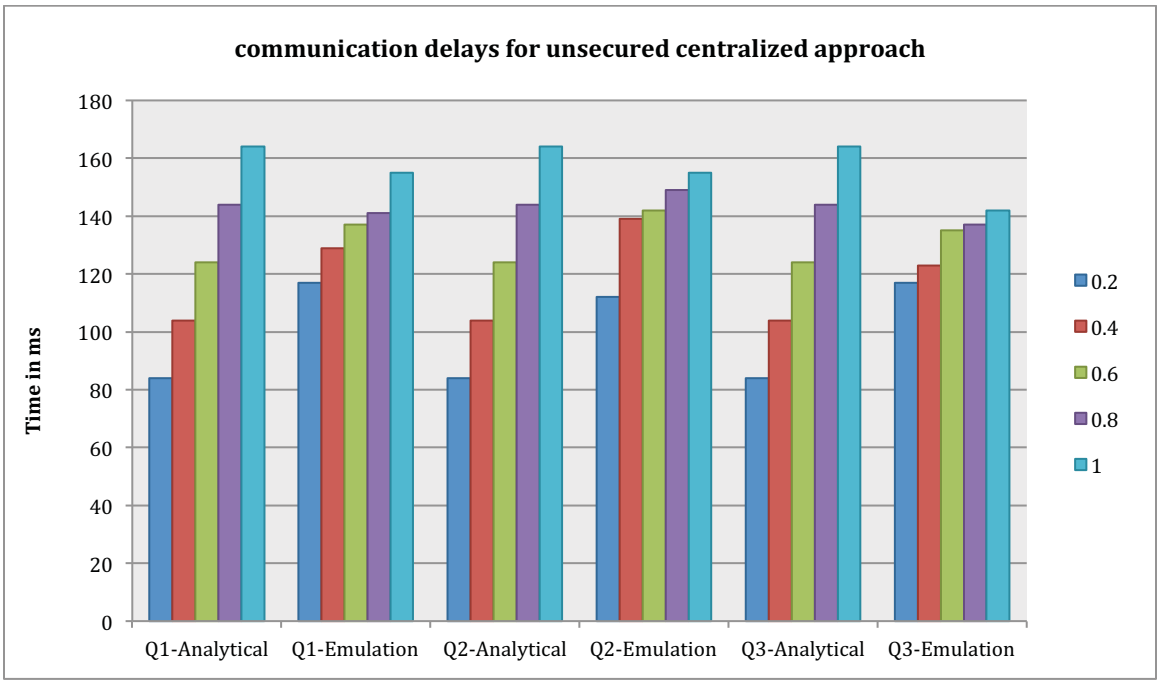


Figure 4- 29: communication delays for unsecured centralized approach

Figure 4-30 shows the communication delays for all queries in both models for secure centralized approach. The delays of the analytical model are the same in all queries as the previous approach. It can be observed that the delays obtained by emulation are higher and becoming more prominent with increased selectivities.

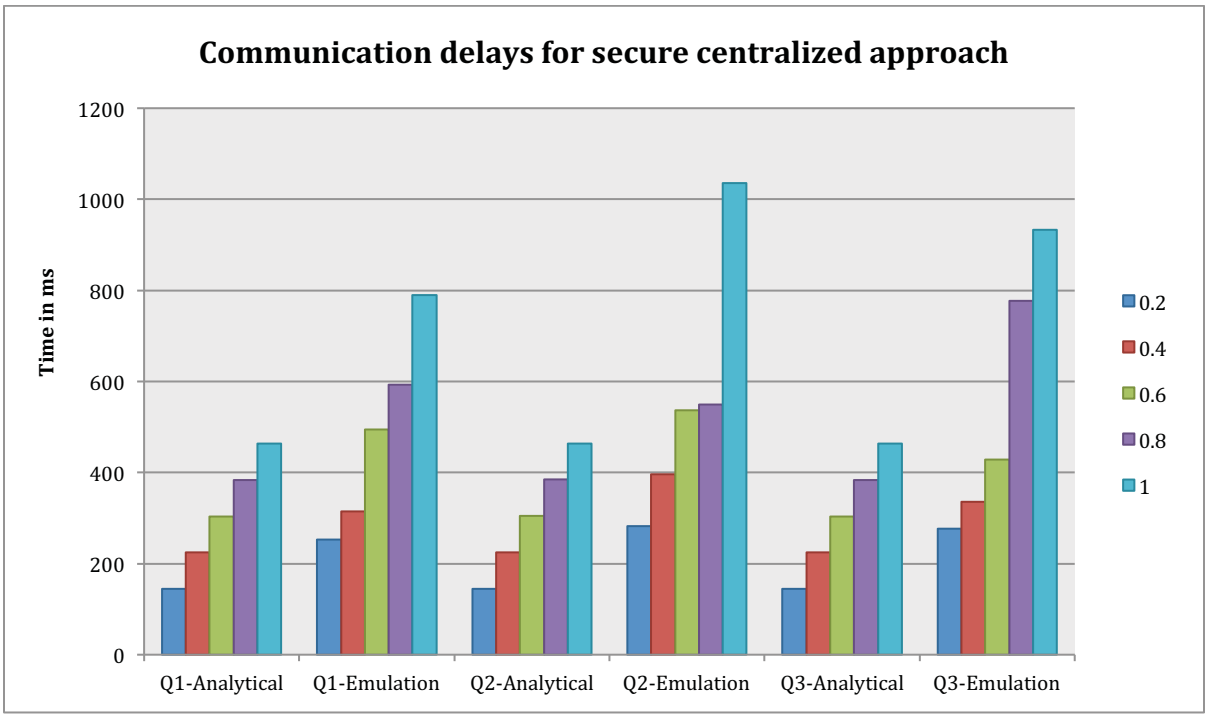


Figure 4- 30: Communication delays for secure centralized approach

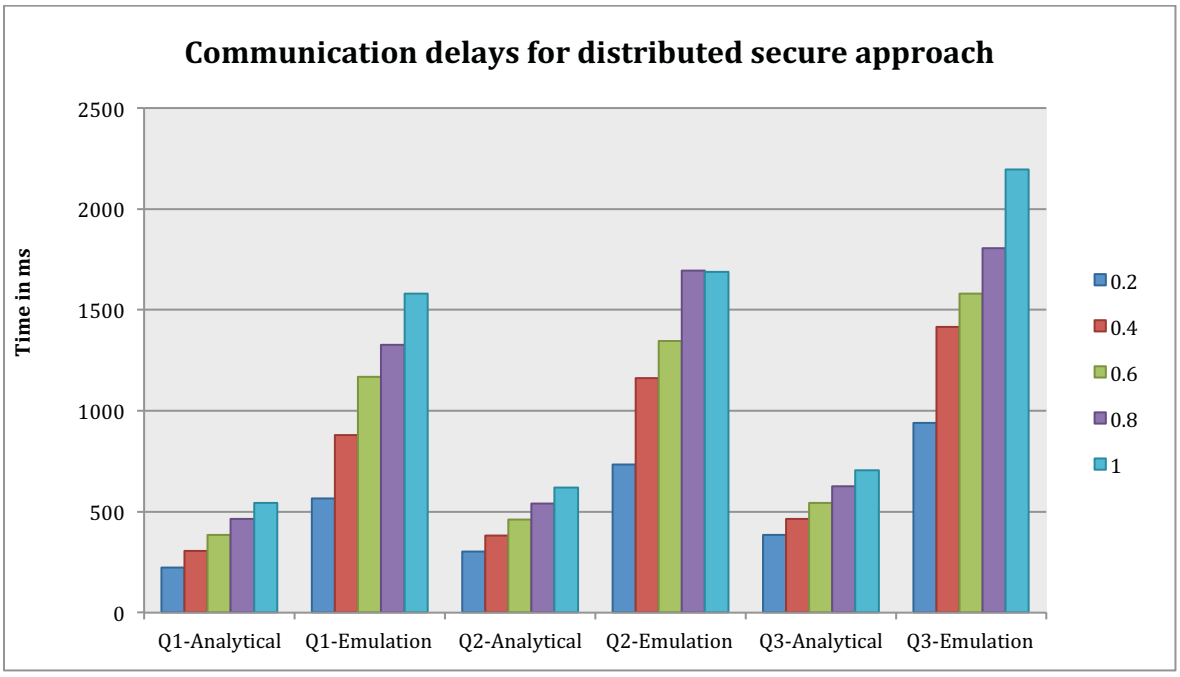


Figure 4- 31: Communication delays for distributed secure approach

Figure 4-31 shows the communication delays for all queries in both models for distributed secure approach. In the previous approaches, the communication delays are increased mostly due to the increases of the selectivity factors. In our scheme, however, the communication is increased as the number of clouds is increased as will as the increases of the selectivity factors. The differences in delays between the analytical model and the emulation model are higher than in previous Queries, and they have the same trend, as in previous Queries, in that the differences in delays derived through emulation and modeling increases with increased selectivities. There are several reasons of the increases of the variances of the delays between the analytical model and the emulation model.

4.1.5.2 Query processing delay

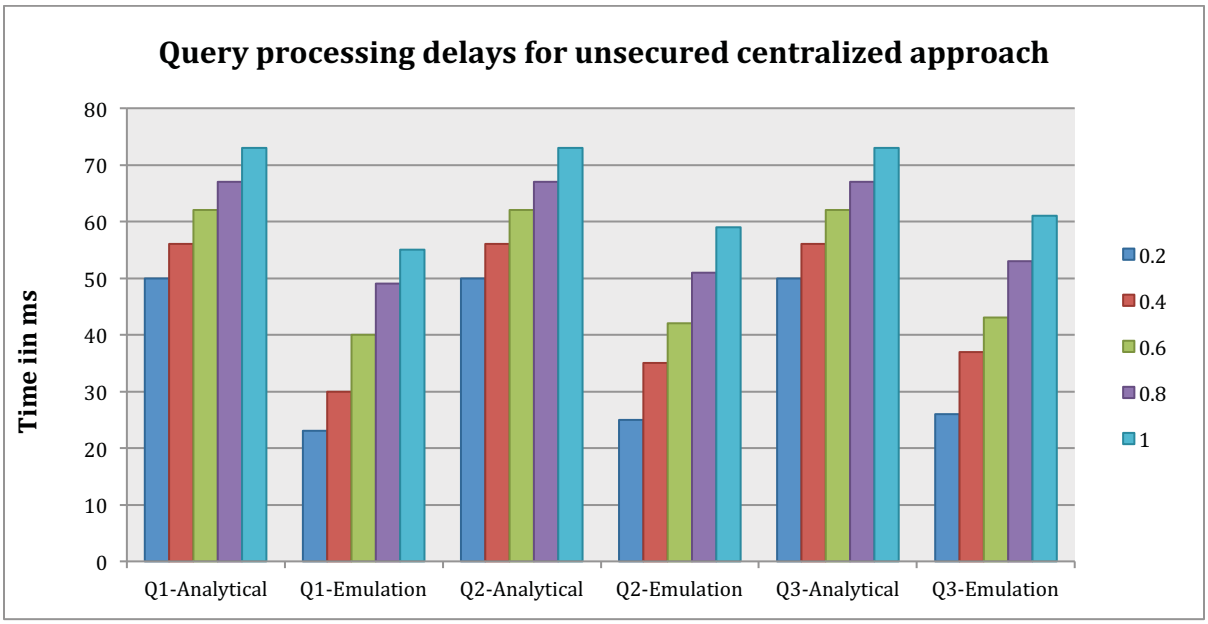


Figure 4- 32: Query processing delays for unsecured centralized approach

Figure 4-32 shows the query processing delays for all queries in both models for unsecured centralized approach. The delays on the analytical model are the same across the queries, and they are the same on the emulation model. The differences between the models are so small in range of 10 -20 ms.

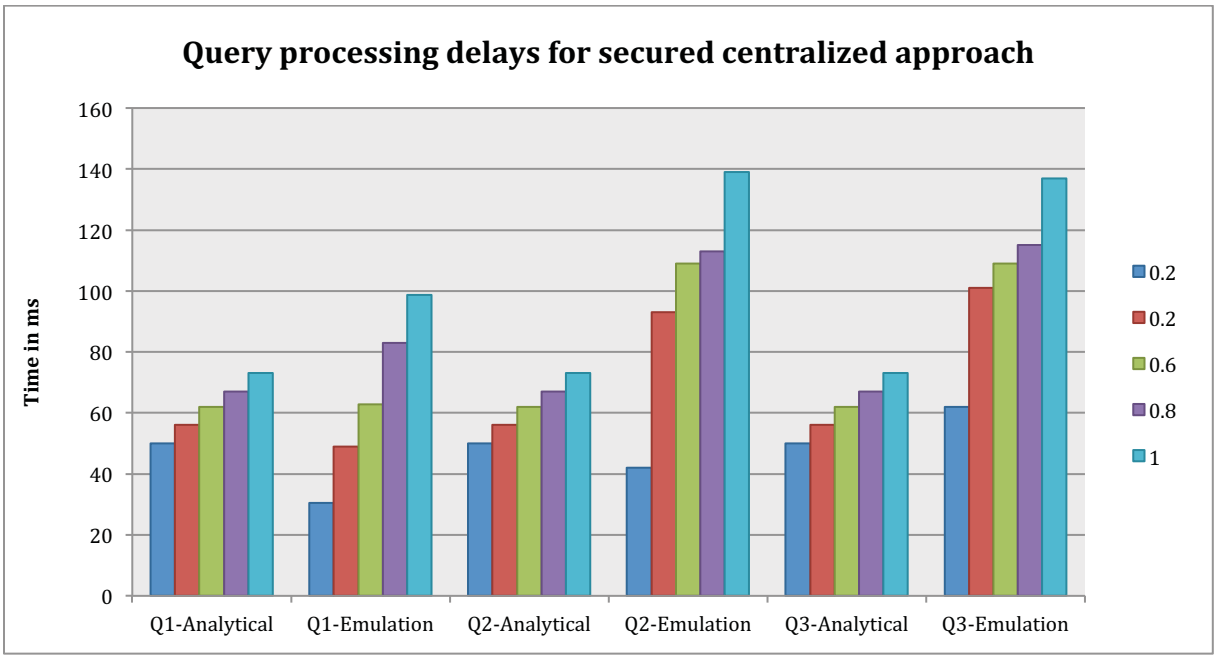


Figure 4- 33: Query processing delays for secured centralized approach

Figure 4-33 shows the query processing delays for all queries in both models for secured centralized approach. The difference on the delays between the models is varying with the selectivity and also with an increase in the number of predicates.

Figure 4-34 shows the query processing delays for all queries in both models for distributed secure approach. It can be observed that for small selectivity, delays through analytical modeling are higher than those obtained through emulation. However, for higher selectivity, delays obtained via emulation are much higher than those obtained by analytical modeling.

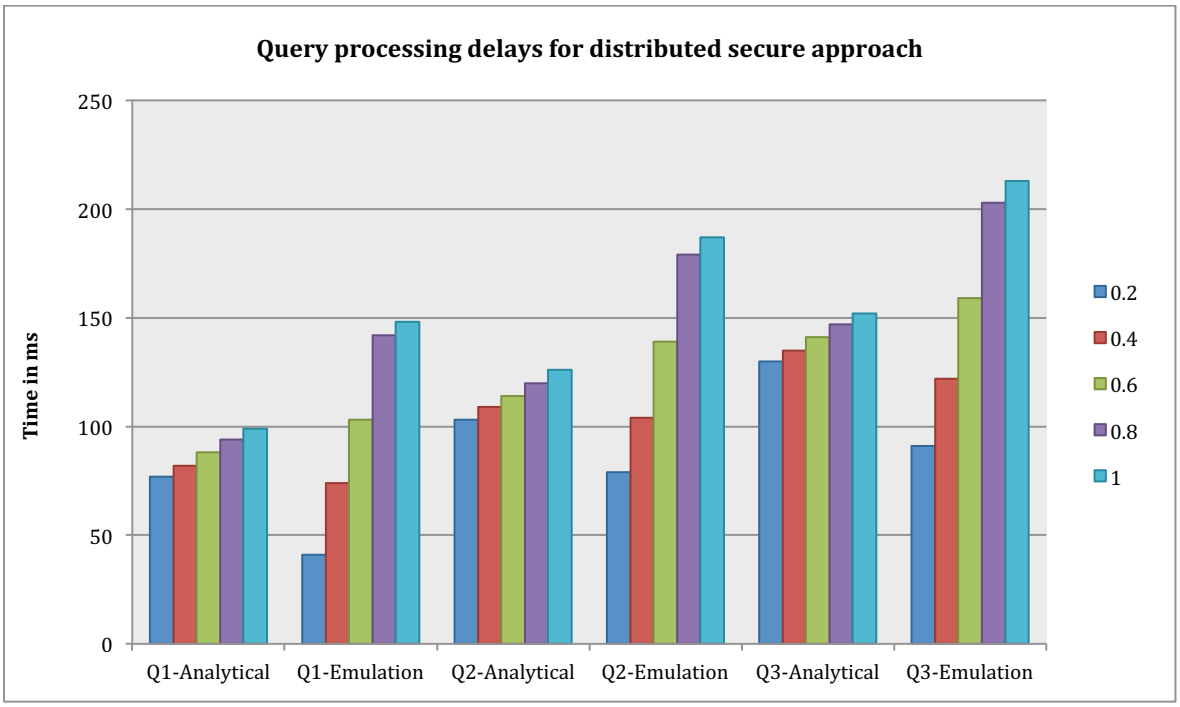


Figure 4- 34: Query processing delays for distributed secure approach

4.1.5.3 Crypto processing delay

In the crypto delays, we have only two approaches to compare with; the secure centralized and the distributed secure. The crypto delays are proportionally affected by the message size. Recall that we only decrypt the final result retrieved from the master cloud so that the size of decryption work depends on the selectivity factor. Thus delays obtained by analytical are the same for both Secure Centralized and Secure Distributed approaches. Same applies for delays derived through emulation. However, delays derived through analytical modeling are higher than those obtained through emulation.

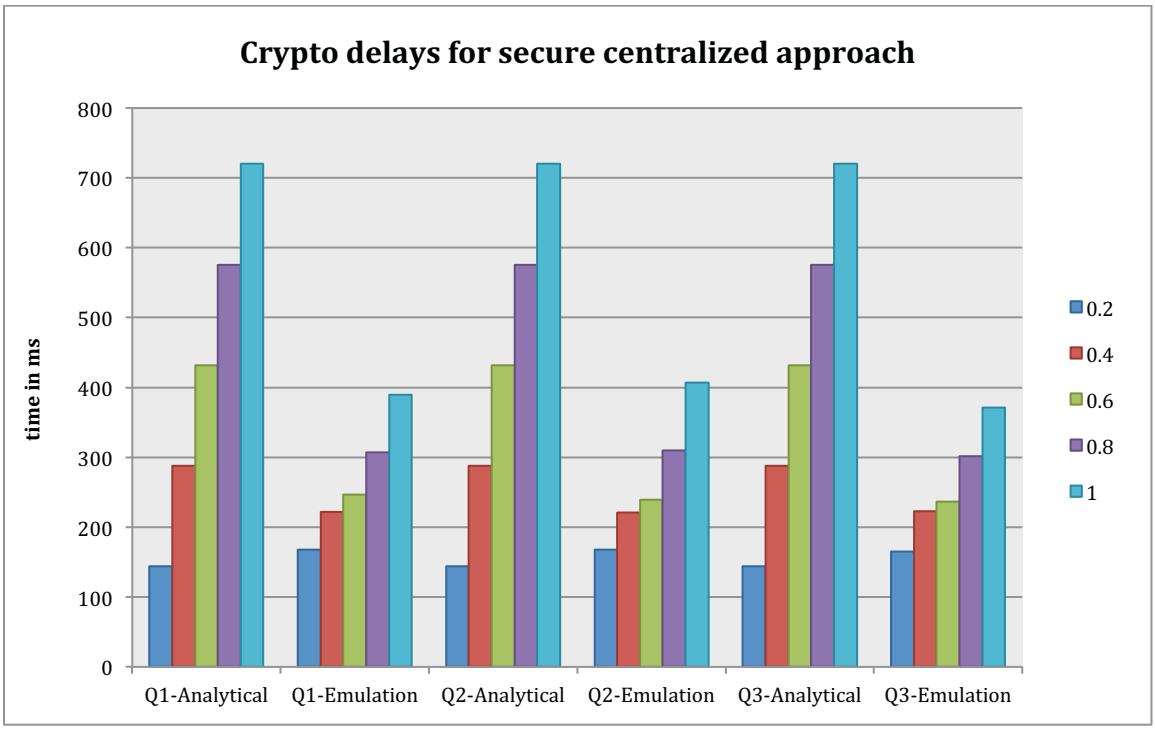


Figure 4- 35: Crypto delays for secure centralized approach

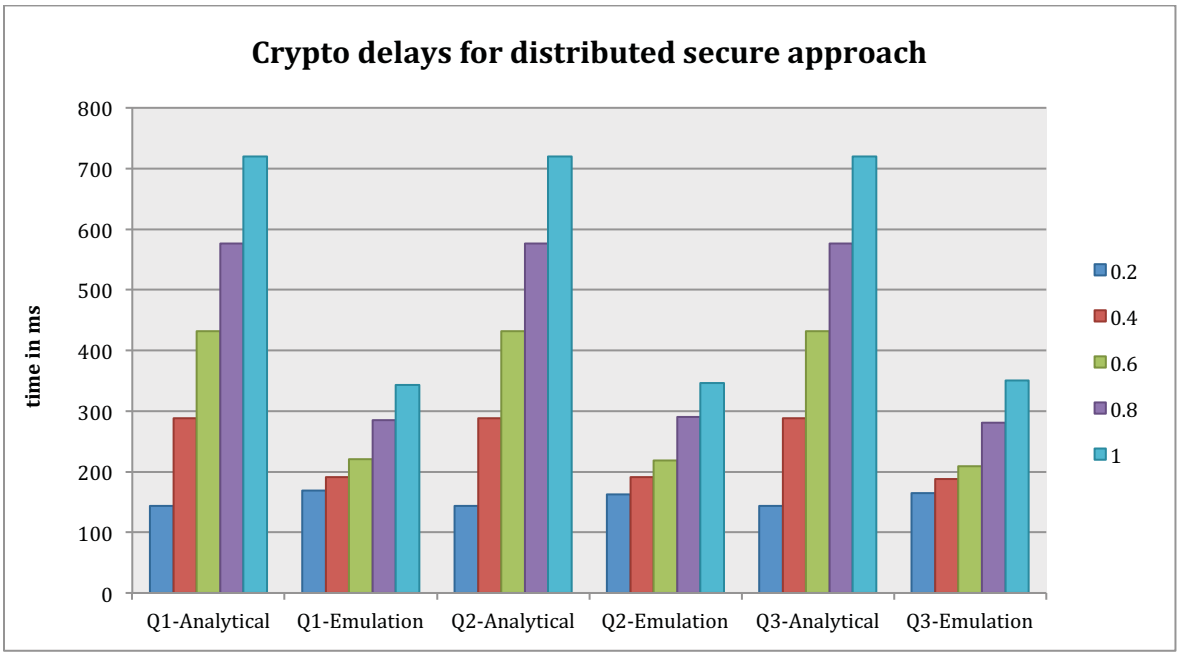


Figure 4- 36: Crypto delays for distributed secure approach

4.1.5.4 Proxy processing delay

Figure 4-37 shows the proxy processing delays for all queries in both models for the distributed secure approach. Recall that the proxy delays include delays to perform the intersection of results returned form slave clouds and overhead associated in transforming the user query to queries on clouds and also processing results received from the clouds. In general, the proxy delays in both models are much less than delay of the other components. The differences in the delays between analytical and emulation models are much less than the differences in the other components.

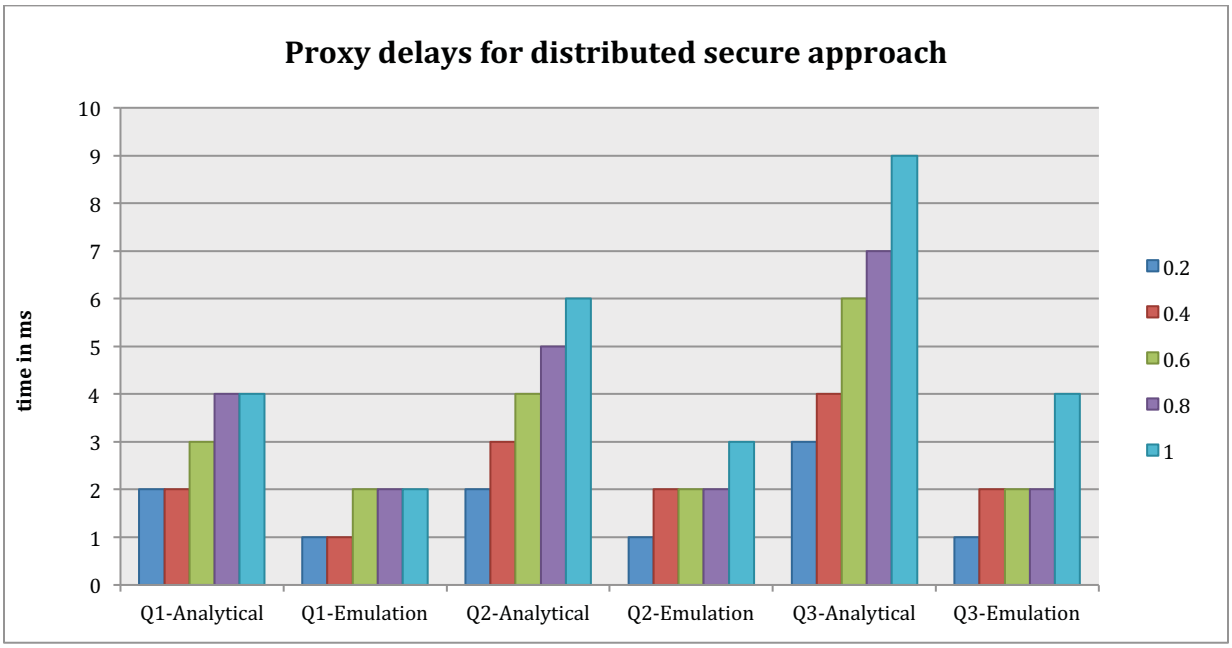


Figure 4- 37: Proxy delays for distributed secure approach

4.2.5.4 Discussion

We observe that the delays derived through analytical modeling, with the exception of crypto delays, are less than those derived through emulation. Recall that in the analytical model, we determine the component delays individually. Thus, the total delay of the analytical model is the sum of the components delays. The analytical model is driven by first calibrating the model's parameters through measurements on average delays in communication, query processing, and decryption, and then using such parameters in analytically predicting delays of the methods under consideration. When making the measurements, we use communication

over Internet and also measure decryption and query processing delays that are performed on infrastructure provided by the cloud. Delays due to emulation are also obtained by using Internet for communication and cloud infrastructure for decryption and query processing, and thus, we can expect much variability in both the analytical and emulation approaches as both depend on when measurements are done. We all accept and experience much variability in delays when communicating over Internet. Delays in cloud infrastructure are also variable as they depend on the load, resources allocated by the cloud provider, and also the cloud's quality of software that measures the load and balances allocation of resources to handle the load.

Thus, although analytical modeling does not predict the actual delays with accuracy, it does provide guidelines on the general trend when variables, such as selectivity or type of issued queries, are varied.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

Cloud computing technology is attractive for storing and managing data. However, concerns over confidentiality with regards to storing sensitive data prevents many official and commercial organizations from moving to the cloud. A number of researchers have attempted to provide a solution to security concerns associated with outsourcing storage. The aim in this thesis research was to prevent untrustworthy providers from obtaining sensitive data. We proposed a combination of the encryption algorithm used in Popa et al.'s (2012) aproach and obfuscation by distributing the data amongst different clouds. The encryption algorithms provide the user with confidentiality and also support query processing of encrypted data, while the distributed technique provides greater security and prevents cloud providers from obtaining meaningful information. The prototype of the proposal was implemented, with the results showing that our scheme provides greater security, however, at the cost of higher delays. Whether such delays are acceptable would need to be ascertained in each specific scenario in which the usage of our proposed method would be considered.

We used two modeling techniques to determine delays of our proposed method and also of methods used for comparison. One method was based on an analytical model while the other method used emulation using a prototype. We calibrated our analytical model's parameters by measurements. When we compared delays derived through analytical modeling to those derived through emulation, we observed that in most cases the analytical model underestimated the delays. Thus, the analytical model can be used to analyze the behavior of the system in terms of the trends on delays but it cannot be used to accurately predict the delays. Of course, the same can be applied about any method that is predicting delays when the load on the system is not taken into account - the load in our case is load that affects delays due to communication over Internet or due to delays of processing on a cloud infrastructure.

5.2 Future Work

This thesis is not the only research attempting to overcome the DBaaS security issue. A number of researchers continue to conduct investigations in the area of cloud computing security, especially in the area of outsourcing data storage. Our scheme is a combination of encryption algorithms and a database distributed system, which aims to prevent sensitive data from being obtained by the cloud computing provider. In both the encryption algorithms and the distributed system, there are potential enhancements that could improve performance and security. The following are some future research works that we believe should be undertaken:

Combining all or some of the encryption algorithms into one algorithm that can support different kinds of queries, so that the number of columns is decreased.

- Using the session key mechanism to update the keys and thereby provide greater security across the clouds.
- Ensuring integrity by making communication between the clouds within a secure channel.
- Exploring whether using horizontal fragmentation coupled with vertical fragmentation may provide greater security and improved performance.
- Using the machine learning technique to cluster fragments across the clouds.

References

Anciaux, N., Benzine, M., Bouganim, L., Pucheral, P., Shasha, D., & Rocquencourt, I. (2007). GhostDB : Querying Visible and Hidden Data Without Leaks.

Bouganim, L., & Pucheral, P. (2002). Chip-Secured Data Access: Confidential Data on Untrusted Servers.

Capitani, S. De, Foresti, S., Samarati, P., & Informatica, D. (2012). Managing and Accessing Data in the Cloud: Privacy Risks and Approaches.

Ciriani, V., Vimercati, S. D. C. Di, Foresti, S., Jajodia, S., Paraboschi, S., & Samarati, P. (2010). Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security*, *13*(3), 1–33. doi:10.1145/1805974.1805978

Daemen, J. (2002). *The design of Rijndael : AES - the advanced encryption standard with 17 tables*. Berlin [u.a.]: Springer.

Desai, A. (2000). New Paradigms for Constructing Symmetric Encryption Schemes Secure Against Chosen-Ciphertext Attack, pages 394–412.

Dijk, M. Van, Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). Fully Homomorphic Encryption over the Integers, 24–43.

Hacıg, H. (2005). Query Optimization in Encrypted Database, 43–55.

Hacıg, H., & Li, C. (2002). Executing SQL over Encrypted Data in the Database-Service-Provider Model, *7*.

Hore, B., Mehrotra, S., Canim, M., & Kantarcioglu, M. (2011). Secure multidimensional range queries over outsourced data. *The VLDB Journal*, *21*(3), 333–358. doi:10.1007/s00778-011-0245-7

Hore, B., Mehrotra, S., & Tsudik, G. (2004). A privacy-preserving index for range queries, 720–731. Retrieved from http://dl.acm.org/citation.cfm?id=1316689.1316752

Liu, D., Wang, S., & Centre, C. I. C. T. (2012). Programmable Order-Preserving Secure Index for Encrypted Database Query. *2012 IEEE Fifth International Conference on Cloud Computing*, 502–509. doi:10.1109/CLOUD.2012.65

Modi, C., Patel, D., Borisaniya, B., Patel, A., & Rajarajan, M. (2012). A survey on security issues and solutions at different layers of Cloud computing. *The Journal of Supercomputing*, *63*(2), 561–592. doi:10.1007/s11227-012-0831-5

Nalinipriya, G., & Aswin Kumar, R. (2013). Extensive medical data storage with prominent symmetric algorithms on cloud - A protected framework. *International Conference on Smart Structures and Systems - Icsss'13*, 171–177. doi:10.1109/ICSSS.2013.6623021

Popa, R. A., Redfield, C. M. S., Zeldovich, N., & Balakrishnan, H. (n.d.). CryptDB : Protecting Confidentiality with Encrypted Query Processing, 85–100.

Sakhi, I. (2012). Database security in the cloud.

Samarati, P., & Society, I. C. (2001). Protecting Respondents ' Identities in Microdata Release, *13*(6), 1010–1027.

Stallings, W. (1999). *Cryptography and network security: Principles and practice*. Upper Saddle River, N.J. : Prentice Hall,.

Stern, P. J., & Eurocrypt, C. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, *1592*.

Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy*. *S&P 2000* (pp. 44–55). IEEE Comput. Soc. doi:10.1109/SECPRI.2000.848445

Weippl, E. R. (2012). Data Confidentiality using Fragmentation in Cloud Computing. Aleksandar Hudic Shareeful Islam, *1*(3).