QATAR UNIVERSITY

COLLEGE OF ENGINEERING

EXPLORING CONFIDENTIALITY AND PRIVACY OF IMAGE IN CLOUD COMPUTING

BY

RANA ADNAN RIHAN

A Thesis Submitted to the Faculty of

the College of Engineering

in Partial Fulfillment

of the Requirements

for the Degree of

Masters of Science in Computing

January 2017

COMMITTEE PAGE

The members of the Committee approve the Thesis of Rana A. Rihan defended on 25/05/2016.

Dr. Khaled Mohammed Khan
Thesis/Dissertation Supervisor

Dr. Mohammad Zulkernine
Committee Member

Dr. Aiman Erbad
Committee Member

Dr. Qutaibah Malluhi
Committee Member

Approved:

Khalifa Al-Khalifa, Dean, College of Engineering

ABSTRACT

RIHAN RANA A., Masters:January:2017, Masters of Science in Computing

Title: Exploring Confidentiality and Privacy of Image in Cloud Computing

Supervisor ofThesis: Khaled M. Khan.

With the increasing popularity of cloud computing, clients are storing their data in cloud servers and are using “software as a service” for computing services. However, clients’ data may be sensitive, critical, and private, and processing such data with cloud servers may result in losing data privacy or compromising data confidentiality. Some cloud servers may be dishonest, while malicious entities may compromise others. In order to protect data privacy and confidentiality, clients need to be able to hide their actual data values and send the obfuscated values to cloud servers.

This thesis deals with the outsourcing of computing to cloud servers, in which clients’ images can be computed and stored. This thesis proposes a technique that obfuscates images before sending them to servers, so these servers can perform computations on images without knowing the actual images. The proposed technique is expected to ensure data privacy and confidentiality. Servers will not be able to identify an individual whose images are stored and manipulated by the server. In addition, our approach employs an obfuscating technique to maintain the confidentiality of images, allowing cloud servers to compute obfuscated data accurately without knowing the actual data value, thus supporting privacy and confidentiality.

The proposed approach is based on the Rabin block cipher technique, which has some weaknesses, however. The main drawback is its decryption technique, which results in four values, and only one of these values represents the actual value of plain data. Another issue is that the blocking technique requires a private key for each block that requires a high-computing effort; requiring one private key for each block of data demands that a great number of keys be stored by the client. As a result, it decreases the robustness of the Rabin block cipher.

This thesis proposes additional techniques to overcome some of the weaknesses of the Rabin block cipher by introducing some new features, such as tokenization, a digit counter, and a set of blocks. The new technique increases the privacy of data and decreases the computational complexity by requiring fewer private keys. The new features have been implemented in image processing in order to demonstrate their applicability. However, in order to apply our approach to images, we must first apply some preprocessing techniques on images to make them applicable to being obfuscated by our proposed obfuscating system.

DEDICATION

This thesis is dedicated to my family.

For their endless love, support, and encouragement throughout the years.

ACKNOWLEDGMENTS

“All the praises and thanks be to Allah, Who has guided us to this, and never could we have found guidance, were it not that Allah had guided us!” I would like to thank all my family and friends for their help and support they offered throughout the entire process, both by keeping me harmonious and helping me put the pieces together. I would like to express my gratitude to my supervisor Dr. Khaled Khan for his guidance and support, as well as his useful comments, remarks, and engagement throughout the learning and working process of this master thesis. Furthermore, I would like to thank Professor Ali Jaoua for his help and support through all of the years I spent in pursuit of my master degree; without his guidance, I would not have reached this stage. I would like to thank all of the doctors who taught me during both my master and bachelor coursework. Last, but not least, I would like to extend my thanks to all of my teachers who taught me during the years of the study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

رَبَّنَآ ءَاتِنَا فِي ٱلدُّنْيَا حَسَنَةً وَفِي ٱلْأٓخِرَةِ حَسَنَةً وَقِنَا عَذَابَ ٱلنَّارِ

Allahumman fa'nee bi-maa 'allam-ta-nee wa 'allim-nee maa yanfa'u-nee war zuq-nee 'ilman yanfa'u-nee

"O, Allah, benefit me with what you have taught me, and teach me that which will benefit me, and grant me knowledge which will benefit me."

CHAPTER 1: INTRODUCTION

Modern society is dealing, using, and connecting with the growing amount of digital data, of which some are sensitive, private, or personal in nature. Because the usage of digital information is growing, the need for outsourcing techniques of various functionalities is increasing as well. Consequently, a new way of outsourcing computation, referred to as cloud computing, has emerged [22]. Cloud computing has become an attractive option for organizations. This outsourcing approach provides highly dynamic computing services in which individuals as well as organizations have the choice to access hardware and software resources as their demands increase or decrease for such resources. Cloud computing offers services that are faster, more flexible, and cheaper than those of conventional IT centers. Virtualization and ultrabroadband connectivity provide this flexibility and scalability to achieve.

Cloud computing, in short, is the outsourcing of IT services to one or more third parties that have rich pools of resources to meet an organization’s computing needs easily and efficiently [34]. These include networking, hardware components, storage, and software applications. Cloud computing facilitates the computing usage to give users on-demand access to computing resources in a “pay as you go” payment model, which is similar to other models that handle utilities such as electricity or water.

Even though cloud computing offers a promising paradigm that can enable businesses to outsource their computations in an agile and cost-efficient manner, the means to guarantee secure outsourcing to cloud computing have not been established. Some cloud services might be untrustworthy or vulnerable to malicious entities. There is a possibility that the outsourced private data of clients could be disclosed to unauthorized entities. Keeping information secure, private, and confidential on a cloud server is challenging; thus, the computation should be done in a privacy-preserving manner that does not reveal information about the sensitive and personal inputs and outputs throughout the computation. The challenge of ensuring secure outsourcing to cloud servers is receiving more attention from the research community, and the principal aim is to fully utilize the benefits of cloud computing while maintaining thorough security for clients' private and sensitive data.

In regard to the security issues related to cloud computing, data encryption seems to be a natural option to pursue. However, this potential recourse has a major problem: It is virtually impossible for cloud servers to compute encrypted data without using effort-consuming techniques, such as a fully homomorphic encryption (FHE) [25]. If a cloud server cannot process a client's data, storing data on the cloud would not seem attractive to clients because the cloud would only provide passive storage.

Using a garbled circuit (GC) [46] is another approach that could secure data confidentiality and privacy. A GC would enable two or more parties to execute functions through their inputs without revealing anything to other parties. This option would provide a protocol for computing any function by using a Boolean circuit representation of the function. An unauthorized entity with access to a GC computation could not learn anything about the input data of the GC or its outputs. However, creating GCs is time consuming. Another serious problem is that they are not reusable.

In this thesis, we attempt to develop an approach that could address some of the problems outlined earlier. The main aim of this research is to develop an efficient technique that could secure outsourced images on cloud servers without requiring additional resources from the cloud servers themselves.

1.1. Problem Statement

In this research, we attempt to propose a secure-outsourcing technique that could allow clients to fully and securely utilize cloud resources and services for their images. The images are computed efficiently and accurately by cloud computing, without the images being revealed to cloud servers. We envision an approach that requires less computational complexities and less additional resources. The approach must ensure confidentiality and privacy of client images.

The success of our approach can be measured by the answers to the following questions:

a) Can a cloud server compute a client's images correctly without knowing the actual images?
b) Can the cloud server derive or extract actual images?
c) Does a client need large amounts of additional resources to make his or her outsourced images secure?
d) Does the cloud server require additional computations or overheads to keep images private and confidential?

Let us provide a specific example of the research problem that we address in the research. A client captures various images of his or her employees and stores them on a cloud server. Each time the client sends an image, the server first checks if the new image has already been stored or not. In order to ensure this, the cloud server compares the input image with other images already stored. In this scenario, the client wants the server to know nothing about the actual images being received; this means that the server needs to store and compare images without knowing the actual pixel values of the images. The comparison task performed by the cloud server should be reasonably accurate. In order to do this, the server does not need additional computing power or resources. We presume that all of the images the client may send will be similar in size, in regard to the number of pixels and resolution. Our research focuses on how to enter this scenario without using expensive techniques, such as public key infrastructure, FHE, or GC.

1.2. Objective and Significance

The main objective of this research is to find answers to the questions defined in the problem statement (Section 1.1). In our approach, we explore and investigate existing cipher and other encryption frameworks to develop, ultimately, an optimal technique. We will experiment with how we can secure images so that cloud servers would be unable to know a client's actual images. Furthermore, we will search for new techniques that can keep clients' images secure when their cloud servers or storage devices are compromised by malicious attackers.

Clients of cloud computing need a lightweight approach that can keep their images private and allow cloud servers to compare images without being revealed to servers. Furthermore, this new approach should demand only a minimum of computational complexity and resources from the client.

The main benefit of this approach is to ensure privacy and confidentiality of clients' images when they use cloud services. Under this new approach, clients will not suffer under a plethora of requirements for computational complexity. In addition, we want to develop a flexible technique that can be applied to different types of data, a technique that will not devour client resources.

1.3. Main Contributions

Our work will propose an optimal approach that can ensure data privacy and confidentiality when cloud computing, and this enhanced security will not need a public key infrastructure, expensive FHEs, or GCs. We concentrate on how to overcome the weaknesses of the existing techniques, and how to optimize the computational complexities of the client and the cloud server, so that our approach requires minimal resources. Our research is based on two strategies:

(i) Exploring how to modify some of the well-established existing techniques, such as the Rabin block cipher, to address our research problems.

(ii) Proposing some techniques and ideas to support the entire approach.

Based on these two strategies, we develop an approach that uses our following specific contributions:

- The first technique is the optimal creation of data blocks, which is an inspired technique from the block-cipher algorithm.
- The second technique is the development of a digit counter, which assigns a digit for each item in the generated blocks of data.
- The third technique is tokenization, which creates a token for each generated block.
- The fourth technique is the identification of sets that group a random number of blocks into one single set.

We also modify the Rabin cryptosystem, which requires two different types of keys: public keys and private keys [37]. In our approach, we explore how to use only private keys instead of both types, without compromising security. We demonstrate the applicability of these techniques by using an image-processing application in a cloud server.

1.4. Thesis Overview

This document is structured as follows:

Chapter 1: Introduction

We outline the general background about the topic, the problem statement, the objective's significance, and the main contributions.

Chapter 2: Literature Review

Here, we critically analyze other related approaches already proposed and accomplished in this field, to assess their strengths and weaknesses.

Chapter 3: Methodology

We describe how our research has been carried out. Techniques and methodologies we explored to address our research problems are included here.

Chapter 4: Data-Obfuscating Technique

We present our main approach for securing data privacy and confidentiality in the context of image processing.

Chapter 5: Implementation and Validation of the Approach

We demonstrate the implemented algorithms and techniques of the proposed framework with running examples and show that the approach can work in practice.

Chapter 6: Discussion

We present our research findings, various experimental results, and limitations of our approach.

Chapter 7: Conclusion and Future Work

We summarize the main contributions of our thesis and outline a list of future work.

CHAPTER 2: LITERATURE REVIEW

Data confidentiality and privacy in cloud computing, in which data are sent to a cloud server in order to be either processed or stored or both, is an important and complex research area. This problem has been addressed by different approaches and from various perspectives. This chapter analyzes some of the proposed approaches in different areas, such as secure outsourcing, data obfuscations, and exponential ciphers.

The proposed solutions, techniques, algorithms, and schemes to this problem can be categorized into two broad areas: secure-outsourcing computation (with an information theoretic approach) and encryption. The next section reviews the work primarily related to secure-outsourcing computation using an information theoretic, and Section 2.2 deals with past research related to encryptions—that is, exponential ciphers.

2.1. Secure-Outsourcing Computation Using an Information Theoretic

Secure-outsourcing computation is the scheme of transferring data from the client to cloud servers for secure computation and data storage. Many solutions, techniques, algorithms, and schemes have been presented and proposed for solving the security problems by using an information theoretic approach. However, all provided solutions and schemes in the literature can be categorized into the four following classes:

1) Outsourcing to single cloud server
2) Outsourcing to multiple noncolluding cloud servers
3) Outsourcing to multiple colluding cloud servers
4) Secure multiparty computation (MPC) scheme.

2.1.1. Outsourcing to a Single Cloud Server

In this scheme, the client uses a single cloud server to outsource his or her computations. This scheme is the simpler scheme among the four described above, because by dealing with one server, the communication protocols of the client will be much simpler than if the client were to deal with multiple servers. Thus, using one server will allow the client to allocate and compute outsourced data more quickly and easily. However, outsourcing to only one server will restrict the client to a limited number of functionalities and to a limited amount of storage space. What is more, if the server breaks down, then the client will lose his or her stored outsourced data. Also, the FHE verification techniques seem to be inherently required, or the client should define a verification operation of the outsourced and computed results. And the verification operation should be done on the client's side.

This scheme was introduced in 1978 by Rivest, Adleman, and Dertouzos [38], who used a singular server with a privacy homomorphism. Their approach implies that clients should use an outsourcing agent to store financial data and to do computations on that same data. In their scheme, financial data or similar forms are stored and computed when they are encrypted, because their approach uses homomorphism encryptions, such as a modular exponential-encrypting technique. The client decrypts the data in order to obtain its original value. In 2001, another approach using a singular server was proposed by Atallah and Pantazopoulos [7]. Their proposed framework was broad, and could be applied to solving many diverse scientific problems. They proposed techniques that could outsource numerous computations, which would, in theory, secure the outsourced data at low costs. Their approach was used to secure their plain data, and it used multiple disguise techniques rather than encryption techniques in order to reduce the required computational complexity of encryption techniques. However, using a single disguise technique is generally less secure than using a single encryption technique, so in [7] they used many disguise techniques, rather than only one, in order to increase their approach. However, their approach may leak private information, because it requires a master key that consists of many subkeys, with a property of discovering one of these subkeys. Note that this approach requires many subkeys because it uses many disguising techniques, and each of these techniques requires a key.

In 2010, Gennaro, Gentry, and Parno [24], as well as Chung, Kalai, and Vadhan [15], proposed approaches for outsourced computation. Gennaro et al.'s [24] scheme combined the proposed GC evaluation [46] with the FHE scheme, which has been proposed [25]. Gennaro et al. [24] proposed that the GC is evaluated only on encrypted inputs. To evaluate a GC on two different inputs was shown to provide no appropriate privacy, because there was a small probability that information would be leaked in each parity, during the exchanges between the GC input and output labels.

According to the approach in [24], the created GC for the required computations is stored in the server. Then the client encrypts his or her input data by using the private key of the FHE and sends the encrypted data to server in order to be evaluated by the stored circuit. After that, by using the homomorphic properties of the encryption scheme, a cloud server computes the function on the received data and sends the result to client. Finally, the client decrypts the received result in order to understand it.

The approach by Gennaro et al. [24] seems to be more efficient than FHE techniques, because their approach does the verification operation only in O(m) operations.

However, their approach might increase its efficiency at the sacrifice of input privacy. There is one main disadvantage: If there are malformed responses from servers, then there is a small probability that a malicious entity already knows k-bit information about the GC, even before the client manages to detect and terminate that malformed response. Therefore, it would be hard for a malicious entity to break the full GC if the amount of k-bit information is sufficiently small. However, this argument cannot be proven.

2.1.2. Outsourcing to Multiple Noncolluding Cloud Servers

In this scheme, the client outsources his or her computations to multiple cloud servers, which do not interact with each other. Generally, the need of multiple servers is for ensuring the correctness of the results [9].

This scheme's main advantage is the reduction in the required computational complexity needed for verifying the computational results of the previously used servers. Also, this scheme can be used to reduce the computational complexity needed for running the homomorphic encryption. On the other hand, the main disadvantage of this scheme is that communicating through these multiple servers overwhelms the client's communication protocols.

In 2008, Benjamin and Atallah [6] presented an example of this scheme. They showed how to fully exploit this approach, in which the expensive cryptographic operations are not required. Their proposed protocols enable a continuous chain of outsourcing data as these data are being outsourced safely. Their scheme allows the client to outsource large matrices to the first server in order to compute the multiplication. Then, to verify the first server computations, the client sends the results of the first server computation to the second server. In this approach, the client does do more than O(n2) work and can discover any corrupted answers by the server.

In 2012, Blanton, Atallah, Frikken, and Malluhi [10] proposed new techniques to enable the servers to carry out the computation obliviously by using only O(rnm) computation and communication complexity. Their solution was designed to work with noncolluding servers that have limited space, such as O(r (m + n)) space. Blanton et al. [10] completely avoided public key cryptography because they used GC evaluation techniques. Therefore, their solution is particularly practical, resulting in fast techniques for the edit-distance and edit-path computation in the privacy-preserving setting.

2.1.3. Outsourcing to Multiple Colluding Cloud Servers

In this protocol, the client outsources his or her computations to multiple cloud servers, and these servers do the work together. This scheme solves the weaknesses of the first scheme: It does not overwhelm the client's communication protocols because it contacts multiple servers that are working together. Therefore, the client sends the outsourced data to the first server as it receives the final result from the last server. Also, this scheme does the verification work by using the servers. However, this scheme's main disadvantage is that it is less secure than the other schemes, especially when the verification operation is done by one of the servers.

In 2010, Roy, Setty, Kilzer, Shmatikov, and Witchel [40] proposed an approach of securing outsourced computations. Their scheme is a MapReduce-based computation [17]. Their approach proposes the Airavat system, which has been designed and implemented to incorporate the mandatory access-control features of SELinux to MapReduce. Airavat was equipped with measures to avoid the leakage of information through the output computations, and the system supports both trusted and untrusted MapReduce computations of sensitive data. It ensures a comprehensive enforcement of data providers' privacy policies because it prevents information leakage through system resources. Airavat runs on SELinux and adds SELinux-like mandatory access control to the MapReduce distributed file system. Furthermore, Airavat enforces differential privacy by applying some modifications to the Java Virtual Machine and the MapReduce framework, in order to prevent leaks through the output of the computation. In [40], Airavat proved to be accurate, but it slowed down the computation because it overwhelmed the system by enforcing differential privacy.

2.1.4. Secure Multiparty Computation

Secure multiparty computation is a protocol that allows parties to compute the same operations or functions to their input data cooperatively; however, each party's inputs are kept private from the others [47].

This model was introduced in 1986 by Yao [47] and is known as the "millionaire problem"—that is, when two millionaires want to know who is richer than the other, but without knowing the actual value of each other's wealth. This scheme allows two parties with individual and private inputs to jointly compute the same functions by entering them using a secure protocol for the same function f. Moreover, the private protocol exposes only the deduced result of function f without revealing any additional information.

Another multi-party model for secure outsourcing was proposed in 2001 by Du and Atallah [19]. One of the proposed models that relates to our work is for storing data securely in external servers after encrypting them, while the other one is for doing operations with encrypted data. Clients need to access and compute their data randomly without revealing themselves to external servers. However, the main drawback in Du and

Atallah's model [19] is that the computed matrices must be the same size, or else they will not be computed or the computation results will be incorrect. This type of secure multiparty protocol was used in [46, 26, 33, 32, 29, 16] and was designed to efficiently compute and operate privately the same jointly used general functions by multi-parties. However, all of these previously proposed solutions have many drawbacks, most of which relate to the security of the multiparty computation protocols. This type of protocol imposes a joint party to compute the comparable burdens.

The problems of securing multi-party computation protocols are defined and summarized in [20], in which the main problem of this scheme is finding a satisfactory vector for different linear systems of equations for each joint party. Another problem is one of matching: Sometimes one of the joint parties wants to know whether its input matches the elements of the entered data set, which has been entered by the other party. In this situation, the main risk is that none of the joint parties should have any knowledge of the others entered data, and the result of this operation may leak private information of both parties. Another problem may occur if the required computation operation requires joining data—that is, classification, clustering, union, and intersection operations are required to be computed with the joint parties' data sets; however, the computing of these operations will leak the private information of the joint parties. Moreover, leakage will occur if the computation operations are only selecting, sorting, or even defining the shortest path.

2.2. Exponential Cipher

An exponential cipher is a cryptosystem based on exponential congruence, which is of the form $a^x \equiv b (mod\ n)$. The exponential cipher encodes plain data by formula $C \equiv A^x (mod\ n)$, in which A is the plain data block, C is the encrypted data block, and n is prime integer number. Exponential cryptosystems are quite difficult to crack than other types of cryptosystems, which has been shown in past research.

Many cryptosystem strategies have been based on exponential ciphers, such as Diffie-Hellman and Pohlig-Hellman's cipher, cipher feedback mode, ElGamal cipher, RSA cipher, and Rabin's block cipher [12]. All these ciphers depend mainly on exponential congruence formulas for encryptions (there are some differences in using this formula among the different ciphers) and to set their factors. The next subsections review some of the main exponential cipher techniques.

2.2.1. Diffie-Hellman key-exchange cryptosystem

In 1976, Diffie and Hellman were the first to invent a public key scheme, which they did for their Diffie-Hellman key-exchange cryptosystem [18]. Their system shows how using a function with a public key can work as a solution for many cryptosystem strategies, including digitalized signatures. Their encryption function is $C \equiv g^x (mod\, p)$, where the public key consists of both g and p, while the exponent part x is the private key.

The Diffie-Hellman cryptosystem has been shown to make it impossible for an eavesdropper to compute the private key when p is a large prime number, but if p is a low prime number, it becomes possible for an eavesdropper to break the private key's seal [1]. Currently, there are no efficient ways to solve the discrete logarithm problems.

2.2.2. RSA Cryptosystem

The RSA cipher is a public key cryptosystem that was presented by Rivest, Adelman, and Shamir in 1978 [39]. Their cryptosystem is a one-way trap-door function. Their encryption function is $C \equiv P^e (mod\, N)$, in which N is a large prime number that is factorable into a product of two large prime numbers p and q. The RSA public key consists of both N and e. In order to determine e, many calculation formulas must be done. To compute the private key, the user needs to compute $d \equiv e^{-1}(mod\ \varphi(N))$, in which the private key consists of d and N, and the decryption function is $P \equiv C^{d}(mod\ N)$.

Rivest, Adelman, and Shamir’s [39] RSA cipher was among the first patent in public key cryptography, and they claimed that their patent included all forms of key cryptography [12].

The RSA cipher’s one-way function has been shown to be most difficult to break, as hard as factorization, but cracking it can be easy in a certain situation. Another decoding algorithm that does not involve or imply factorization N can crack the one-way function [37]. RSA cipher’s main weakness is that the known public key can easily crack the cipher [44]. After comparing it with the Diffie-Hellman cipher, we find that it requires more computational processes than the Diffie-Hellman cipher requires; furthermore, the vulnerability of the RSA cipher is much higher than that of the Diffie-Hellman cipher.

Shortly after presenting the RSA cipher, a modification of the RSA public key approach was presented by Williams in 1980 [44]. In his approach, he describes a modification of the RSA technique, a modification that makes cracking the RSA system as difficult as factoring the form of N, where N = pq, and both p and q are prime numbers of the form $p\equiv3$ $(mod8)$, $q\equiv7$ $(mod\ 8)$. Williams’ decryption method [44] is based on the Chinese remainder theorem, which creates four candidate roots of the encrypted data, and among all of these roots, only one is correct. To solve the decryption problem, Williams uses the Jacobi symbol methodology.

The main weakness in Williams’ [44] approach is that the plain data must be restricted to a special form to make sure that all conditions of the encryption and decryption process are met. In addition, the approach adds more computational complexity to the original RSA cipher; after incorporating Williams' modification, an RSA scheme requires O(n3) for each encryption and decryption process.

In 1988, Kurosawa, Itoh, and Takeuchi presented another modification for the RSA cipher [30]. Their modification is a public key cryptosystem that is very difficult to break, as hard as factoring the large prime of N (N = pq). Their approach suggests that the public key should consist of both N and C, such that (C/p) = (C/q) = -1, and the private key should be the two large prime numbers p and q. For the decryption method, they also use the Chinese reminder theorem, which causes the same problem that occurred for Williams [44]. The approach of Kurosawa et al. [30] uses an extra bit and the Jacobi symbol methodology to solve the four-root decryption problem, so the system can recognize the correct root of the plain data.

Kurosawa et al. [30] reduce the computational complexity so that the encryption process requires O(n2) operations; however, the decryption process still needs O(n3) operations. This approach does not require a special form of plain data to be encrypted, which is a great advantage over Williams' approach.

Takagi [43] proposed another approach that speeds up the RSA cipher. His proposed cryptosystem is based on the RSA cryptosystem in the form $C \equiv M^e (mod\, N)$, where N = pkq. Using this form of N provides resistance to fast-factoring algorithms. Also, his decryption algorithm is three times faster than using the Chinese remainder theorem when decrypting the RSA cipher. In Takagi's approach, the public key is the same as in the normal RSA cipher, while the private key consists of d, p, and q, and his encryption methodology is the same as that of the original RSA cipher. However, Takagi [43] enhanced the decryption process by using his decryption methodology in [42]. He computes the modular multiplication of the encryption exponent and a greatest common divisor to decrypt blocks after the first block. The decryption time of the first block is the foremost one, because the first decrypted block is decrypted by using the formula $P \equiv C^d (mod\, N)$; while for the remaining blocks, his cryptosystem decrypts them by solving the linear equation modulo N. Therefore, even if a message is several times longer than public key N, the encryption of the plain data will be fast without repeatedly using the secret key cryptosystem. The main weakness of Takagi's approaches [42, 43] is that there is a small probability the decryption may fail.

In 1995, Bellare and Rogaway [11] proposed a different methodology for modifying the RSA cryptosystem. Their approach focuses on padding data, which is referred to as optimal asymmetric encryption padding (OAEP), and it enhances the security of the RSA cryptosystem. Their method converts the RSA trapdoor permutation into a chosen secure system for cipher text, within the random oracle model. The first step of this approach is to hash the plain data unit by using the concatenation between both a generator formula and a hash function; then it encrypts the result of that concatenation by using the RSA's encryption method. In order to decrypt the ciphered data, the data are first decrypted by using the normal RSA decryption methodology; then they use the inverse of their hash methodology to compute the original plain data.

Bellare and Rogaway [11] proved that their OAEP methodology increases security of the RSA cipher. In addition, their padding methodology can be applied to any other cryptosystem, not only to the RSA cipher. However, their padding methodology is suitable for short-length data. In addition, their padding methodology demands more computational complexity for encryption and decryption techniques. Despite that the output of their padding decryption technique is unique, the padding decryption has a small probability of failing [6, 11].

2.2.3. Rabin Cryptosystem

In 1979, Rabin [37] presented another public key cryptosystem based on an exponential cipher, and it is actually a type of quadratic cipher. He calls his cryptosystem the Rabin cipher, and its encryption methodology comprises only a quadratic congruence modulo with a large prime number $C \equiv P^2 (mod\ N)$, in which P is the plain data block, C is the ciphered data block, and N is a large composite number that is factorable into a product of two large prime numbers, p and q. The public key of the Rabin cipher consists of only N, which increases its security over the RSA security. Its private key consists of the large prime numbers p and q. The decryption methodology of the Rabin cipher is based on solving the quadratic congruent by using the Chinese remainder theorem, which resulted four candidate roots, and only one of these roots represents the plain data block [37].

The main advantage of the Rabin cipher over the RSA is that the former is more secure than the latter; the Rabin cipher is as hard as factorization when it is being cracked. In addition, the Rabin cipher can be computed much more quickly than the RSA cipher. However, the main disadvantage of the Rabin cipher occurs in its decryption method, because the cryptosystem may fail to recognize the correct plain data block over all of the four roots during decryption. In addition, if public key N has been factored, the Rabin cipher can be cracked easily.

In 2014, Hashim presented research [27] in which he developed the H-Rabin cryptosystem, a modification of the Rabin cryptosystem. Hashim's approach enhances security of the Rabin cryptosystem by increasing the difficulty of factoring the public key, N. Hashim designates N to be a large composite number that can be factored into a product of three large prime numbers (p, q, and r), rather than into only two prime numbers, as happens in the Rabin cipher. Therefore, the private key of the H-Rabin cryptosystem consists of three large prime numbers, and the encryption and decryption methods are the same as those of Rabin's approach. However, the result of decryption consists of eight candidate roots of decrypted plain data blocks, because N is factorable into a product of three large prime numbers [27].

The main advantage of the H-Rabin cryptosystem is that it enhances the security of Rabin cipher; even if adversaries try to factor N, they will hardly consider that N is factorable into a product of three large prime numbers—not merely two. However, H-Rabin's decryption method is the main weakness. First, it demands more computational complexity because it has to compute all of the eight roots. Currently, there is not any feasible process that can be added to the H-Rabin cryptosystem to make it recognize the correct root of the plain data block among all of the eight roots.

In 2001, Boneh presented another padding approach [13] similar to the OAEP technique [11]. Boneh's method is much simpler than OAEP and introduces two simple padding schemes: Simple-OAEP (or SAEP) and SAEP+, and he applied both of his padding techniques to the Rabin and RSA cryptosystems. Both SAEP and SAEP+ schemes are used as preprocessing functions with the Rabin or RSA trapdoor functions. So, in order to encrypt plain text, first it is padded by using an SAEP or SAEP+ schema; then the result of the padding schema is encrypted normally by either the Rabin encryption function or the RSA encryption function. For the decryption, the encrypted message is first decrypted by using the normal decryption technique of either the Rabin or the RSA cipher; then the padded message is decrypted to get the original structure [13].

The main advantage of using either the SAEP or SAEP+ schemes is that they enhance the security of both the Rabin and RSA ciphers. In addition, SAEP and SAEP+ schemes are much simpler and demand less computational complexity than OAEP. However, using any of these padding schemes adds computational complexity to encryption and decryption processes in the Rabin and RSA ciphers. In addition, in order to apply these padding schemes, the size of the plain data is restricted; even when using SAEP+, the size of the plain message has to be smaller than it does within an OAEP scheme. Furthermore, there is a small probability that the padding decryption may fail or give an inaccurate result of the plain data [6, 13].

Another approach that helps recognize the correct decrypted root among the four roots created in the decryption stage was represented in [21]. The authors generated additional information to correctly recognize the proper root, and they can apply both Jacobi symbols and Dedekind sums as the last process in their decryption stage. The approach is based on three schemes, all of which solve the one-to-four mapping function precisely. The first scheme was the simplest one, and it exploits Jacobi symbols by using two extra bits that are generated in the encryption stage and later sent with the encryption message. Subsequently, in the decryption stage after computing the four roots, the system selects only two roots that have the same party bit by comparing them with the first bit that was earlier sent with the encryption message. In regard to these two selected roots, the system selects the root that corresponds to the number equal to the second bit that was sent with the encrypted message. In the second scheme, the extra information is represented by a generated factor that is added to the public key. Therefore, in the decryption stage, the system computes two factors, which depend on that extra factor. After computing the four roots, the system selects only the root that corresponds to the two earlier computed factors. The third scheme proposed [21] is based on using both the Jacobi symbols and Dedekind sums to mimic the computational operations. In the third scheme, extra information is similar to the first scheme, which is represented by two extra bits that were sent with the encrypted message. However, in the last step of the decryption stage, it uses simpler operations to determine the correctly computed root.

The main advantage of these three proposed techniques [21] is that the Rabin cryptosystem can identify the correct computed root among the four roots created in the decryption stage. However, the main disadvantage is that they keep computing all of the four roots and then select the appropriate one by adding two or more operations to the decryption stage. The additional operations in the encryption stage are used to generate extra information. For these reasons, the computational complexity increases for computing needless information, such as that of all four roots. These schemes could decrease the demand for computational complexity by half in the decryption stage if they were to check each generated root before generating the other remaining roots.

Kurosawa, Ogata, Matsuo, and Makishima presented a new approach [31] for increasing security of the Rabin cipher, and their approach combines padding plain data to increase security and using Jacobi symbols in the decryption stage to solve the four-to-one mapping problem. Their approach enhances the security of the public key by using the indistinguishability under adaptive chosen cipher text attacks (IND-CCA) scheme, which is the strongest notion for a public key scheme. Kurosawa et al. [31] improve the IND-

CCA scheme by making its security equivalent to factoring N = pq. The presented scheme adds an extra bit to the public key, in order to use Jacobi symbols in the decryption process, and solves the four-to-one mapping-function problem. In the encryption stage, the first step is to encrypt the plain text by using their proposed padding methodology; then the result of that is ciphered by the normal Rabin's encryption method. The same process occurs in the decryption stage: In the first step, the encrypted message is decrypted by using the normal Rabin cipher decryption method. In the second step, Jacobi symbols are applied to find the correct root. In the final step, the padded message is decrypted to get the original data.

The main advantages of this approach are the solving of the four-to-one mapping-function problem of the Rabin cipher and the enhanced security. However, demand for computational complexity increases when these advantages are pursued. In addition, the final step of their decryption method has a small probability of failing. There are also some restrictions on the message size for padding.

Another approach is reported in [5]. This approach aims to reduce the computational complexity of the Rabin decryption process, and it proposes a fast and efficient algorithm for reducing the running-time complexity of the current decryption operations of a primitive Rabin cryptosystem. It uses the Garner algorithm to solve the Chinese remainder theorem algorithm during the decryption process, cutting down the requirements of certain operations. For example, the multiplication operation requires about half of the numbers usually needed for the Chinese remainder theorem. The result of the decryption process also reduces to two roots rather than four roots. This approach also differs from the Rabin cryptosystem in the decryption method because using the Garner algorithm requires only one modular inversion and voids the need to conduct modular reduction on large moduli. The approach adds a proposition condition in the decryption method in the last step to reduce the decryption outcomes to only two roots.

This approach reduces the time complexity of the decryption process. It reduces the cost of decryption by approximately by 33.8%, especially because the decryption output has also been reduced to only two roots. However, this approach does not have any technique that can recognize the correct root between these two roots.

Asbullah an Ariffin presented the Rabin-p cryptosystem approach [6], which mainly aims to reduce the computational complexity of the decryption process, as well as solving the four-to-one mapping-function problem. Of foremost importance, their decryption method results in a unique, correct root. In addition, Rabin-p increases the security of the Rabin cipher by increasing the difficulty of factoring the public key N. To do so, it proposes that N = p2q, rather than N = pq. According to this approach, the Rabin-p public key is generated by selecting two distinct prime numbers, p and q, and it generates N that equals to p2q. Therefore, the public key consists of N, and the private key consists of only p. However, to encrypt the plain text, it must be restricted to some range of bits depending on N, where the plain text of m size, and N should be a coprime, such that gcd(m, N) = 1. The main difference between the Rabin-p and Rabin ciphers occurs in the decryption process. The Rabin-p decryption method is based on operating single modular exponentiation operations rather than computing the Chinese remainder theorem. The result of the Rabin-p decryption method is two roots rather than four.

The Rabin-p cryptosystem has many advantages. First, Rabin-p reduces the computational complexity of the decryption method. Second, it solves the four-to-one mapping problem efficiently without any need for extra computational processes or extra information. Third, Rabin-p increases the security of the Rabin cryptosystem. On the other hand, the cost of these advantages is that the plain text message must be restricted to some range of bits to make it applicable for being encrypted by Rabin-p.

The proposed Rabin-RZ cryptosystem [48] has an efficient method for solving the four-to-one decryption failure in the Rabin cryptosystem. Furthermore, Rabin-RZ uses a stronger public key than the one used in the Rabin cipher, where its public key N = p2q, which is much harder to be factored than when N = pq. In order to generate the public key, both prime numbers p and q should be distinct and restricted to a defined n-bit strong prime number. Then the public key N is set to p2q, and the private key consists of p, q, and d, where d is equal to pq. In order to encrypt plain data in the Rabin-RZ cryptosystem, the plain data should be restricted to a specified n-bit interval that is much less than the normal Zpq domain in the Rabin cipher. The other processes of the Rabin-RZ encryption are the same as in the Rabin cipher. The main difference between the Rabin-RZ and Rabin cryptosystem occurs in the decryption stage. First, the Rabin-RZ decryption computes the congruence of the ciphered data modulo d, where d = pq. However, in order to recognize the correct root among the resulting four, Rabin-RZ computes another factor based on the ciphered data and the resulting root. Then a condition is applied on the last computed factor, and that condition succeeds by producing only one root, which is the correct root.

The main advantage of the Rabin-RZ cryptosystem is that it solves the four-to-one mapping failure of decryption accurately. It does so without any possibility of failing, without any need for extra information, and without any additional demands for the computational complexity in the decryption process. In addition, Rabin-RZ enhances the security of the Rabin cryptosystem. However, the domain for the plain data is restricted to a specified n-bit interval that is much less than the normal domain of the Rabin cipher.

In our approach, we use the Rabin cipher to be our main encryption methodology. According to our previous analysis, the Rabin cipher is more secure than both the Diffie-Hellman key exchange and the RSA cryptosystems. However, the Rabin cipher cryptosystem has some decryption problems when trying to recognize the root that represents the plain data, but this weakness can be solved by using many techniques (some of which are fast, while others are slow). The Diffie-Hellman approach is not applicable to our own because we are not going to exchange the key with server—that is, we do not incorporate a public key in our system. For this reason, Diffie-Hellman is not applicable in our approach.

CHAPTER 3: METHODOLOGY

The main goal of our work is to provide a security system that addresses the problem statement outlined in Section 1.1. Therefore, we want to propose an approach that can secure the outsourced data in a form that allows the cloud-computing server to do the required computations on the data without knowing their actual values. However, the computation results must be reasonably accurate. Our proposed approach does not allow the cloud servers to extract the actual values of the outsourced data. Our approach must optimize client resources so that the client does not require additional resources to execute our approach. In our work, we can use additional operations in order to verify the server's computational results. Because we want to save and optimize client resources so that clients will not do any operations or works that reach or exceed O(n3) operations, we use a lightweight cipher technique and add to it some obfuscation techniques to strengthen security.

In this chapter, we review and present existing methodologies and techniques that have been previously used to solve the secure-outsourcing problem. We analyze features of these techniques to understand how we could address the research problem defined in this thesis. Our research methodology is primarily based on laboratory experiments using programmatic tools and techniques in an application context, namely, image processing. The next section describes our research methodology framework, with which we carry out our research activities in a systematic way. We also discuss introductory materials of key existing technology that we are going to explore, use, and modify if necessary, in order to formulate a viable solution to the research problem.

3.1. Methodology plan

In order to achieve our research objective and address the research problem delineated in Chapter 1, we plan to use the research methodology framework depicted in Figure 3.1.

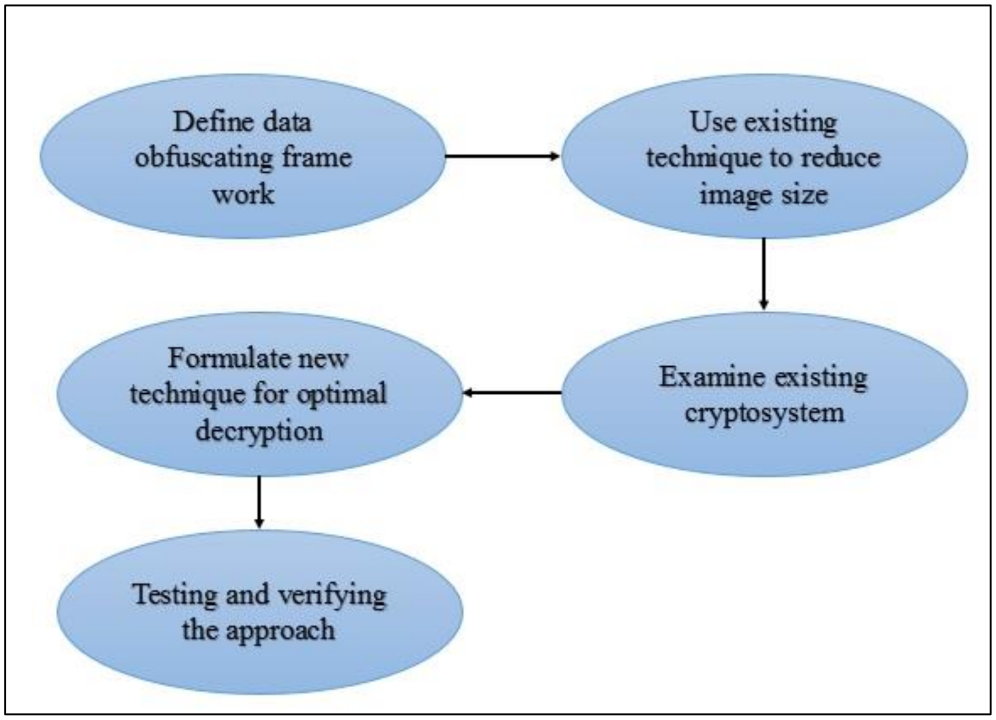

Define data obfuscating frame work
Use existing technique to reduce image size
Examine existing cryptosystem
Formulate new technique for optimal decryption
Testing and verifying the approach

Figure 3.1: Research Methodology Framework

Our first stage of the methodology is to define a new obfuscating framework that can lay out a high-level solution of the problem. This high-level overview of the solution will act as a guiding principle of the proposed solution.

The second stage of our methodology is to examine some of the existing techniques that could allow us to reduce the size of images without losing vital information. Note that our application context in this research is image processing. A single image consists of thousands of pixels, and computing this huge number of pixels by the client is a computational-intensive task. In this stage, we will investigate how to minimize the data volume of the image in the approach using existing techniques.

The third stage of our methodology is to use one or more existing cryptosystems to optimize the obfuscation of images. We are going to experiment how we can encrypt our data without using a public key. To do so, we plan to examine the Rabin cipher.

In the fourth stage, we explore how to devise an optimal decrypting technique that could help clients to decrypt data easily.

Finally, we will test and verify our proposed solutions in terms of feasibility of the approach and the accuracy of the results.

3.2. Obfuscation methodology

As we have mentioned earlier in this chapter, the main aim of this work is to ensure the security of outsourced data, and to make it suitable for further computations by cloud servers without decryption, so that the server can compute the outsourced data correctly and accurately. Yet we do not want to use any technique that requires huge overheads, such as fully homomorphism encrypting. Therefore, we use a lightweight cipher technique, the Rabin cipher, because its difficulty in being cracked is as challenging as integer factorization [28]. The Rabin cipher is partially a homomorphic encrypting system—that is, it is homomorphic in some operations, such as multiplication operations, but it is not homomorphic in other operations, such as additional operations [45]. This cipher technique can be modified to suit our needs for developing an approach that does not allow any leakage of the client's private information. By encrypting the client's outsourced data by using the Rabin cipher technique, we can enable the cloud server to compute the encrypted outsourced data without any need for decryption.

3.2.1. Rabin Cipher

The main idea of Rabin cryptosystem is based on computing quadratic congruence modulo a composite number. Where solving the quadratic congruence is simple when the factorization of the composite number is known, however, it will be very complex when the factorization of the composite number is unknown.

The encryption formula of Rabin cryptosystem is

$$C \equiv P^2 (mod\, n) \qquad (1)$$

Where C is the cipher data, P is the plain data, n is the product of two distinct large primes, say p and q, and both are congruent to 3 modulo 4. As the public key will be n, and the private key is p and q.

The decrypted methodology of the Rabin cipher is based on creating a system of linear congruencies, which consists of congruence formulas (2) and (3):

$$C_p \equiv P^{\frac{p+1}{4}} (mod\, p) \qquad (2)$$

$$C_q \equiv P^{\frac{q+1}{4}} (mod\, q) \qquad (3)$$

Then we use the Chinese remainder theorem and the quadratic congruence propositions in order to solve the created system of linear congruencies. Consider the following quadratic congruence propositions used to solve the system of linear congruencies.

Proposition:

Suppose integers $a_1, a_2, a_3, \dots , a_n$ are relatively prime pairwise. Then $(a_1, a_2, a_3, \dots , a_n)|c$ if and only if $a_1|c, a_2|c, a_3|c, \dots , a_n|c$.

Proposition:

Let $a \equiv b\ (mod\, m_1), a \equiv b\ (mod\, m_2), a \equiv b\ (mod\, m_3), \dots , a \equiv b\ (mod\, m_n)$, where

$a_1,\ a_2,\ a_3, \dots, a_n$ are relatively prime pairwise. Then

$$a \equiv b\ (mod\ m_1, m_2, m_3, \dots, m_n).$$

Consider the Chinese remainder theorem, where it is used with these propositions for solving the system of linear congruencies.

3.2.1.1. The Chinese Remainder Theorem

Let $n_1, n_2,\ \dots., n_r$ be positive integers, such that $gcd\,(n_i, n_j)$ = 1 for i ≠ j. Then the system of linear congruencies is

$$x\ \equiv\ c_1\ (mod\ n_1); x\ \equiv\ c_2\ (mod\ n_2); \dots.; x\ \equiv\ c_r(mod\ n_r),, a$$

and it has a simultaneous solution, which is a unique modulo $n_1 n_2 \dots n_r$.

Normally, the Chinese remainder theorem is solved by using Gauss's algorithm, which is provided below.

3.2.1.2. Gauss's Algorithm

Let $N = n_1 n_2 \dots n_r$ then

$$x\ \equiv\ c_1 N_1 d_1 +\ c_2 N_2 d_2 + \cdots +\ c_r N_r d_r\ \ (mod\ N)\ ,$$

Where $N_i = {}^{N}\!/_{n_i}\ , and\ d_i\ \equiv N_i^{-1}\ (mod\ n_i).$

Therefore, decryption is normally done by using Gauss's algorithm for solving the Chinese remainder theorem. Where both congruency formulas (2) and (3) are combined to obtain solutions for P. Hence, the solutions of P will be gotten by the formula

$$P\ \equiv\ \pm\big(zqq_p{}' \pm wpp_q{}'\big)(mod\ n)$$

Where $z\ =\ C^{(p+1)/4}, w =\ C^{(q+1)/4}, q_p{}'$ is an inverse of q modulo p, and $P_q{}'$ is an inverse of p modulo q, as the $P_q{}'$ is calculated by the using the extended Euclidean algorithm.

Consider Gauss's algorithm 3.1 for solving Chinese remainder theorem.

Algorithm 3.1: Gauss's Algorithm

Input: integers *p*, *q*, and ciphered data *C*

Output: integers X_1, X_2, X_3, X_4

Steps:

```
1- $C_p \leftarrow P^{\frac{p+1}{4}} (mod\, p)$
2- $C_q \leftarrow P^{\frac{q+1}{4}} (mod\, q)$
3 - $M_1 \leftarrow q^{-1} (mod\, p)$
4- $M_2 \leftarrow p^{-1} (mod\, q)$
5- $X_1 \leftarrow C_p M_1 q + C_q M_2 p\ (mod\, pq)$
6- $X_2 \leftarrow C_p M_1 q - C_q M_2 p\ (mod\, pq)$
7- $X_3 \leftarrow -(C_p M_1 q) + C_q M_2 p\ (mod\, pq)$
8- $X_4 \leftarrow -(C_p M_1 q) - C_q M_2 p\ (mod\, pq)$
9- return $(X_1, X_2, X_3, X_4)$
```

However, using Gauss's algorithm for solving Chinese remainder theorem requires more computational complexities, because it requires computing two multiplicative inverses (i.e., step 1 and step 2). It also requires many numbers of the multiplicative, such as the ones used in steps 1, 2, 3, and 4.

Another algorithm is used for solving the Chinese remainder theorem: Garner's algorithm. The Garner algorithm is generally used for solving the Chinese remainder theorem, where Garner's algorithm sufficiently speeds up the Chinese remainder theorem computation.

3.2.1.3. Garner Algorithm Definition

Given the positive moduli $m_i \in Z\ (0 \leq i \ \leq n)$, which are pairwise relatively prime and given corresponding residues $u_i \in \ Z_{m_i}\ (0 \ \leq i \ \leq n)$, then to compute the unique $u \ \in \ Z_m\ (where\ m = \ \prod_{i=0}^{n} m_i)$ satisfies the system of congruence

$$u \ \equiv \ u_i\ (mod\ m_i), 0 \leq i \leq n$$

The key to Garner's algorithm is to express the solution $u \in \ Z_m$ in the mixed radix representation

$$u = \ v_0 + \ v_1(m_0) + v_2(m_0 m_1) + \cdots + v_n(\prod_{i=0}^{n-1} m_i)$$

Where $v_k \in \ Z_{m_k}\ for\ k = 0, 1, 2, \ldots, n$.

Therefore, the first step in the decryption process is forming a system of linear congruencies, which consists of two congruence formulas, (2) and (3). Then the Garner's algorithm is applied to solving the Chinese remainder theorem, which in turn solves the system of linear congruencies, (2) and (3).

Consequently, consider decryption algorithm 3.2, which is the Garner's algorithm for solving the Chinese remainder theorem, and is formed as follows:

Algorithm 3.2: Garner's algorithm

Input: integers p, q

Output: integers $X_1, X_2, X_3,\ X_4$

Steps:

```
1- $C_p \leftarrow P^{\frac{p+1}{4}} \ (mod\, p)$

2- $C_q \leftarrow P^{\frac{q+1}{4}} \ (mod\ q)$

3- $j \leftarrow p^{-1} \ (mod\ q)$

4- $h_1 \leftarrow (C_q - C_p)j \ (mod\ q)$

5- $h_2 \leftarrow (-C_q - C_p)j \ (mod\ q)$

6- $X_1 \leftarrow C_p + h_1 p$

7- $X_2 \leftarrow C_p + h_2 p$

8- $X_3 \leftarrow pq - X_2$

9- $X_4 \leftarrow pq - X_1$

10 - return $(X_1, X_2, X_3, X_4)$
```

As has been noticed, Garner's algorithm is almost more efficient than Gauss's algorithm in regard to solving the Chinese remainder theorem. Because Garner's algorithm only requires computing one modular inverse operation (i.e., step 3 in algorithm 3.2), it cuts the computational efforts in half. In addition, Garner's algorithm reduces by half the number of multiplications needed for Gauss's algorithm. For these reasons, the Garner algorithm has significant computational advantages when solving the Chinese remainder theorem.

Consider that the hardness of Rabin ciphers increase when the composite number N is large, and it will be very simple to crack Rabin ciphers when the composite number N is small. Therefore, Rabin ciphers are highly secure when the plain data is represented as a large integer number, while the amount of its security is reduced when the plain data is a small integer number. Thus, to solve this problem, almost all of the researchers that incorporate Rabin ciphers complement it with a blocking cipher technique, in order to increase the security of encrypting small integer numbers, and to make the Rabin cipher applicable for all integer numbers.

3.2.2. Block Cipher Definition

A block cipher is a parameterized, deterministic function that maps n-bit plaintext blocks into n-bit ciphertext blocks. The value n is called the block length. The cipher is essentially a simple substitution cipher with character set = {0, 1}n [8].

Although any size for blocks is acceptable, some aspects must be considered for selecting the size, in order to reach the maximum amount of security in the system. For example, the size of a block cannot be very small because small blocks are discovered more easily. On the other hand, block size should not be too large because the cipher will begin to operate inefficiently when the plain data is receiving extra padding before being encrypted. The need for larger amounts of padding makes the system inefficient. In addition, if the padding is always done with the same bits, the padding may reduce the degree of security at times [8].

By using the block cipher, the restriction of the Rabin cipher's security for encrypting small integer numbers is solved. However, the main drawback of using the Rabin cipher occurs in its decryption algorithm, which results in four candidate blocks of the original plain text for each decrypted block cipher. The second weakness of the Rabin cryptosystem is that if it has a public key that can be factorized, then that will cause a breaking in the Rabin system. In addition, using a fixed size for blocks can provide hints to potential attackers if they discover some of the plain text blocks corresponding to some previously sent cipher text blocks. Also, using different keys to encrypt each block consumes more of a client's resources. The client needs to save all of these private keys. In addition, using only one key to encrypt all blocks poses a certain danger: If that key is discovered, then the whole system will be compromised.

Thus, one of our main objectives in this thesis is to address the weaknesses of the Rabin cipher's decryption techniques, to increase the difficulty of its integer factorizing. We also aim to provide techniques that could provide some flexibility in choosing block size, so that blocks can be encrypted and decrypted without causing any loss of original plain data. We also intend to discover some new methodology to minimize unnecessary consumption of client recourses.

3.3. Testing and Evaluating Methodology

In order to evaluate our proposed approach, all of the weaknesses mentioned earlier must be addressed and overcome. Our proposed approach will be more useful if we can address all of these weaknesses. Our approach is based on techniques that address the following challenges:

1. Computing the outsourced encrypted data accurately by the cloud server, without revealing the data's original value
2. Increase the security of data outsourcing
3. Address the Rabin cipher's decryption weaknesses
4. Optimize client resources

We are aiming to develop an approach by which clients can obfuscate their data and send them to cloud servers, in order to be computed and stored. Then server can compute the obfuscated, outsourced data without knowing their original values, and can return the results to the client, along with the obfuscated data that the client earlier sent to the server. The client then can de-obfuscate the data to find if the server computed the correct data.

We apply our techniques on image processing—namely, the comparison of images by the server—to find the similarity of the images. In our framework (presented in Chapter 4), an image is represented by an integer matrix, because the Rabin cipher is only applicable to integer numbers. The cloud server compares an image with the stored obfuscated images. The cloud server returns the comparison result to the client, whether it has found a similar image or not. The final step of the client is to de-obfuscate the received obfuscated image and to compare it with the saved one, in order to ensure the correctness of the input image to the server. In Chapter 6, we are going to do some experiments to test and verify our framework. These tests are intended to confirm the accuracy of our cloud-computing results and the required time for implementing our framework.

In this chapter we have presented and analyzed the techniques and methodologies that we are going to use to achieve our aim of proposing a secure outsourcing system. Our approach is applied on the following scenario: The client obfuscates his or her image by using the Rabin cipher, and the server computes and stores client data without having any knowledge of the original data. The server compares the input of the obfuscated image with the stored obfuscated images on the cloud. The server then sends comparison results, in terms of close similarity it has already found with the input of the obfuscated image. After receiving the obfuscated image from the server, the client has the choice to de-obfuscate the ciphered image by using the Garner algorithm. Because the Rabin cipher is only applicable with integers, we map our plain data to integer numbers.

In the next chapter, we introduce our approach in detail. In addition, we show and describe our contribution to solving the weaknesses of the previously established techniques, and we address the listed challenges as well.

CHAPTER 4: DATA-OBFUSCATING TECHNIQUE

In this chapter, the main approach for data obfuscation suitable for processing by cloud servers is presented. One main objective of our approach is to obfuscate data in such a way that the cloud server or untrusted computers can process them without decrypting, most importantly without knowing the actual values of the outsourced data. The untrusted server must be able to compute the obfuscated data. The approach is flexible and applicable to any form or representation of data; it can be an integer matrix, double matrix, string matrix, and so forth. We use an image for the experiments of our research, which is a matrix of pixels as input to our obfuscating process. Consider that this approach can be used for applications that depend on equality and inequality operations, such as image and text comparison.

The proposed framework consists of three main components: preprocessing, obfuscating, and de-obfuscating data. Preprocessing is the first component that transforms raw data, such as images, into plain data types, such as integers. The main purpose of preprocessing is to prepare data for the second component of the framework—that is, the obfuscating of data. We have proposed some new techniques and modified some existing ones in order to support an effective and efficient obfuscating of data for untrusted cloud servers. Our proposed framework is expected to reduce computational complexity and to ensure data privacy and security. The third component is the de-obfuscating technique, which is only needed when the users want to de-obfuscate their obfuscated data.

Our specific contribution to this research is the proposed new techniques that include creating blocks, developing digit counts, tokenization, and identifying sets of blocks. All these are developed to optimize the process and secure the data. Creating blocks is an inspired technique from the world of block ciphers. In addition, developing a digit counter is a technique that creates a digit for each content piece of the generated blocks. The tokenization technique creates a token for each generated block. The identifying-sets technique groups a random number of blocks in one set. Finally, we modify the Rabin cryptosystem, which is an asymmetric system that requires two different types of keys: the public and the private keys [37]. In our approach, we use only one type of keys that is the private key. The details of these new techniques and updated existing techniques are discussed in the latter sections. Figure 4.1 shows a high-level architecture of the framework. It has four major components:

- Preprocessing
- Obfuscating
- De-obfuscating.
- Verification.

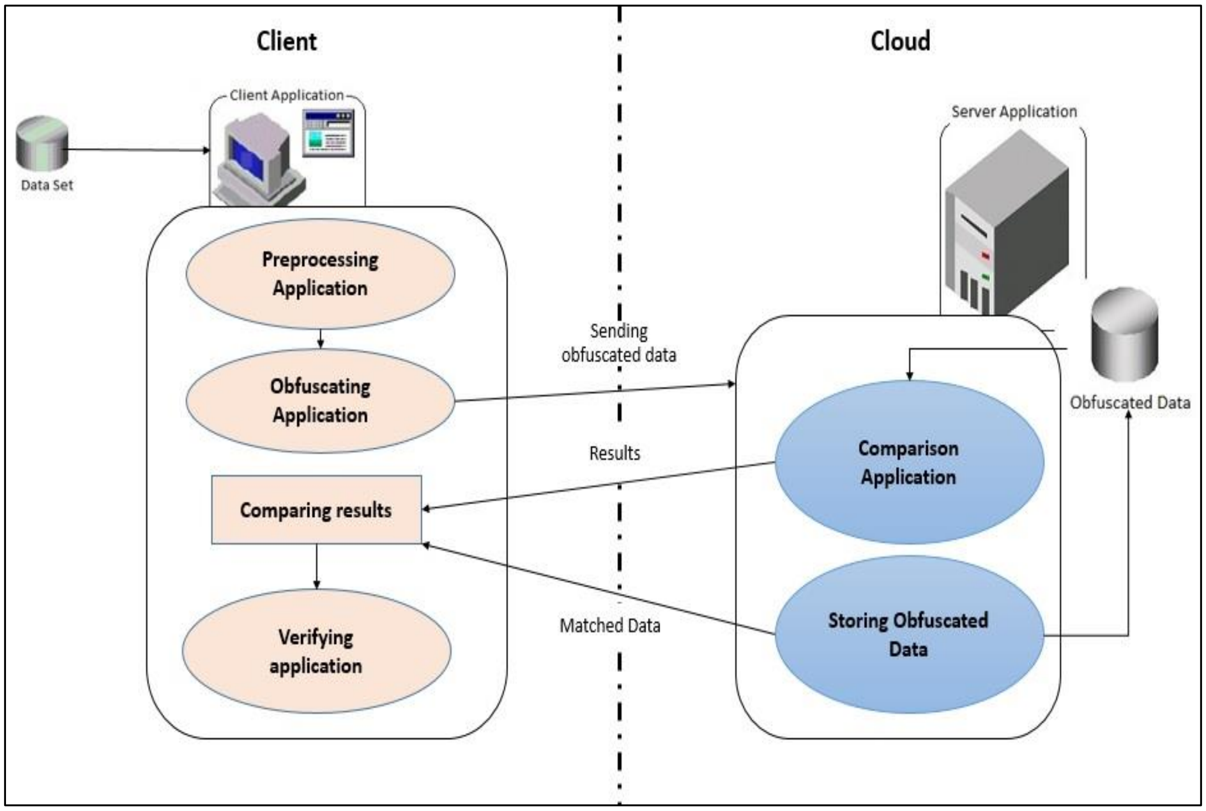


Figure 4.1: High-Level Architecture of the Data-obfuscating Process

4.1. The Proposed Framework

According to our framework, the client prepares his or her data into a matrix of plain integer values. The matrix is then transformed into obfuscated data and sent it to the server to be compared with other obfuscated matrices already stored in the cloud server. The server then sends back the comparison result to the client, which can be either “Exist” if the server finds a matched matrix, or “Not Exist” if the server does not find any matching matrix. If the server result is “Exist” then it also sends the matched obfuscated matrix to the client. Then the client de-obfuscates the received matrix and compares it with the corresponding integer matrix it saved earlier to verify the server result.

4.2. Preprocessing

The main purpose of the preprocessing is to process plain data and prepare them so that the obfuscating component can modify data easily without having any problems. This actually transforms data from one format to another depending on the data type needed by the next process, namely, obfuscating. This component is used and can be modified depending on the data type, so it differs from one type of data to another one.

In our framework, we use an image as input data into the framework. The image is represented with a matrix of thousands of pixels. Our framework can only obfuscate the integer data type; therefore, it has two tasks: reduce the size of the matrix of the image and convert pixels of the image into integers. We first reduce the image size to improve efficiency and speed up the obfuscating process. Once the matrix is reduced into an optimal size, we transform this reduced image of pixels into a matrix of integers. The size reduction and conversion of the matrix values into integers serve two purposes: (1) Obfuscating becomes easier and faster because of the reduced size and integer values; (2) because data size is reduced, the cloud server can perform the comparison in relatively less time.

In order to achieve these goals, the preprocessing component consists of four different techniques, as depicted in Figure 4.2:

- Reducing image size
- Converting colored images into gray images
- Extracting pixel values

- Representing double into integers.

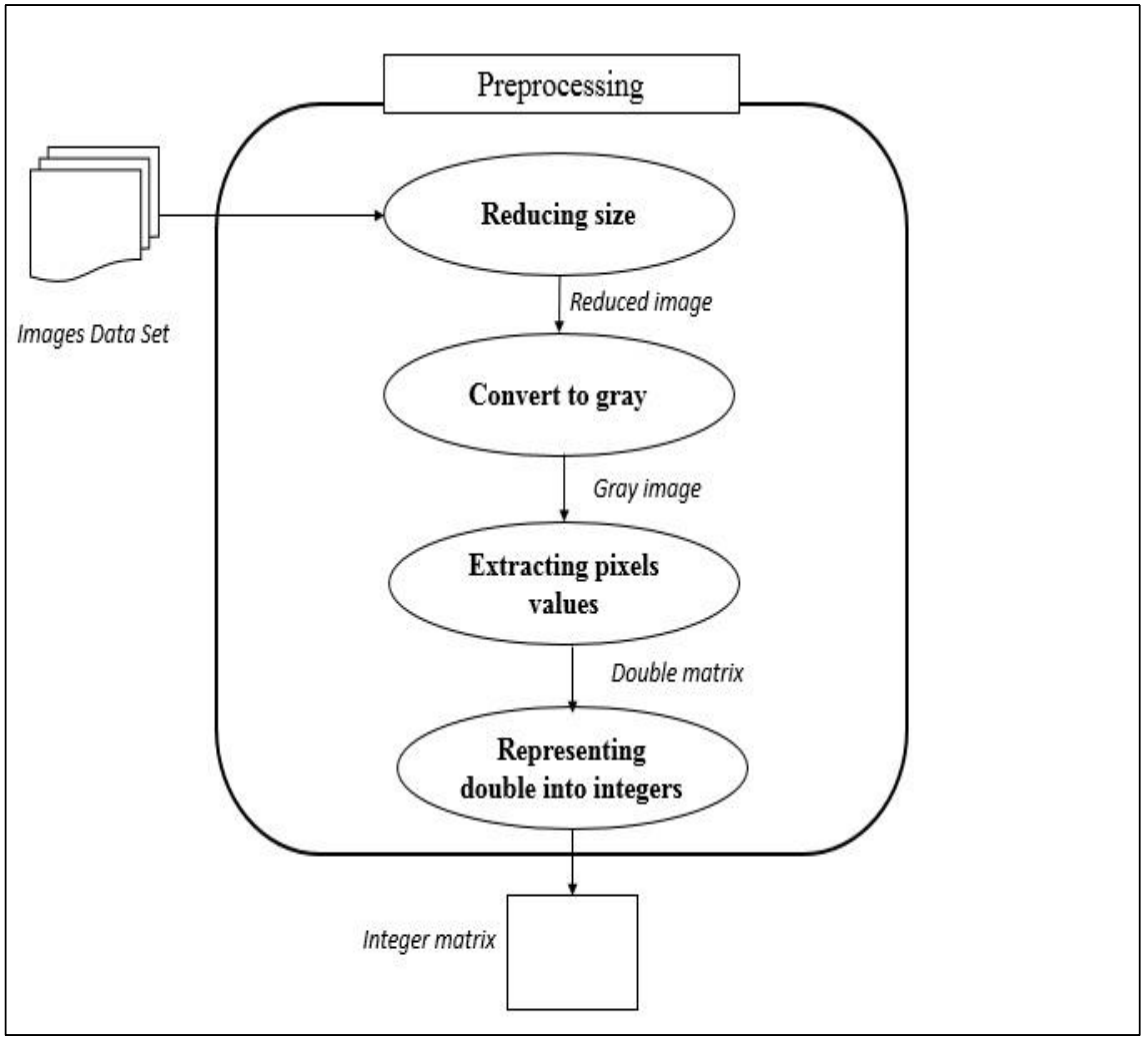


Figure 4.2: Preprocessing Component

The details of each of these four techniques are provided in the following subsections.

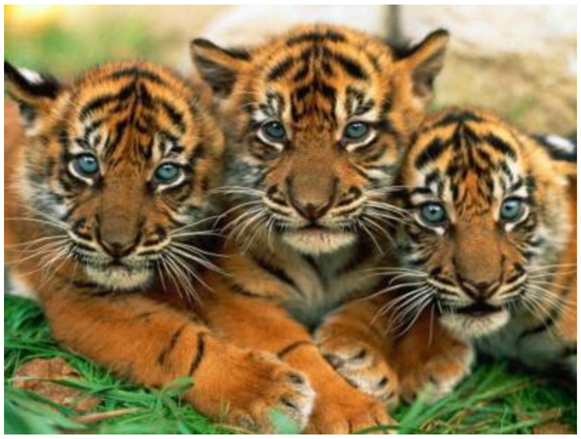
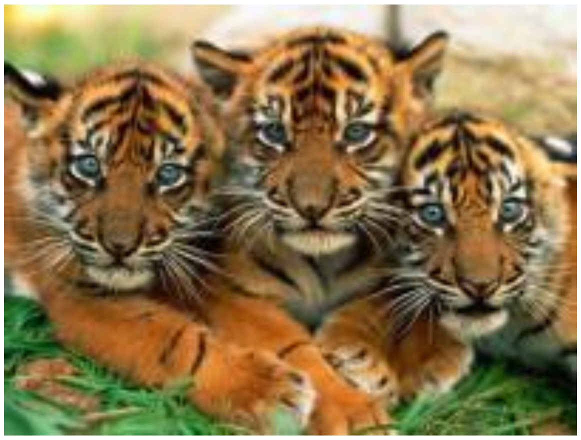
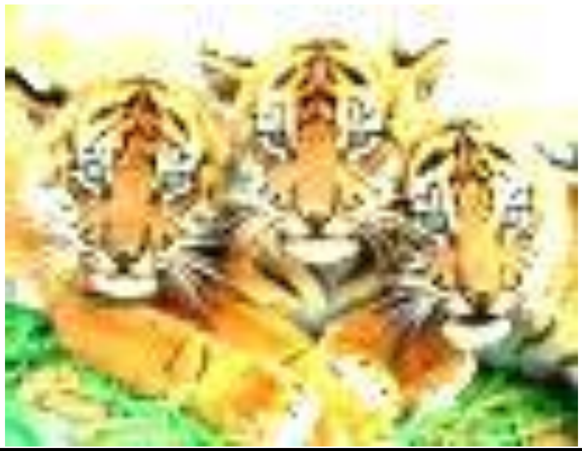
4.2.1. Reducing Image Size

Because the images each consist of thousands of pixels, we need to reduce them to a size that can be represented with less numbers of pixels, yet keep these images meaningful, accurate, and clear. The main benefit of this step is to reduce the client's computational complexity by minimizing the size of the image matrix. Subsequently, rather than computing a matrix of size 1000 * 600 pixels, the client ends up computing a matrix of 500 * 300 pixels or 250 * 150 pixels for the same image, which saves a significant amount of computational complexities.

The image size is reduced into such a tolerable size that the information kept in the reduced size is still meaningful and does not lose much information for processing by the server. Reducing image size to an optimal level does not affect the comparison, but it significantly improves the obfuscating and comparison processes, because these processes will compute smaller matrices rather than larger ones, which saves computational complexity. Example of reducing an image size is shown in the Table 4.1.

Table 4.1: Reducing Image Size: An example

Image Dimensions:	Image
800 * 600 pixels (original dimensions)	

Reduced to 401 * 301 pixels	
Further reduction of size but still tolerable. 202 * 152 pixels	
More reduction, but lost some information 102 * 77 pixels	

More reduction but lost a lot of information. 52 * 40 pixels	

As shown in Table 4.1, reducing size significantly results in loss of information of the image after a certain size threshold. For this reason, there must be a tolerable reducing rate of the image so that it will not lose significant information. For instance, in the example of reducing image size shown in Table 4.1, the most tolerable size for an image is 202 * 152 pixels, which allows it to keep all of its information. Note that in our framework, it is the user who determines the most tolerable size of image.

4.2.2 Converting a colored image to gray

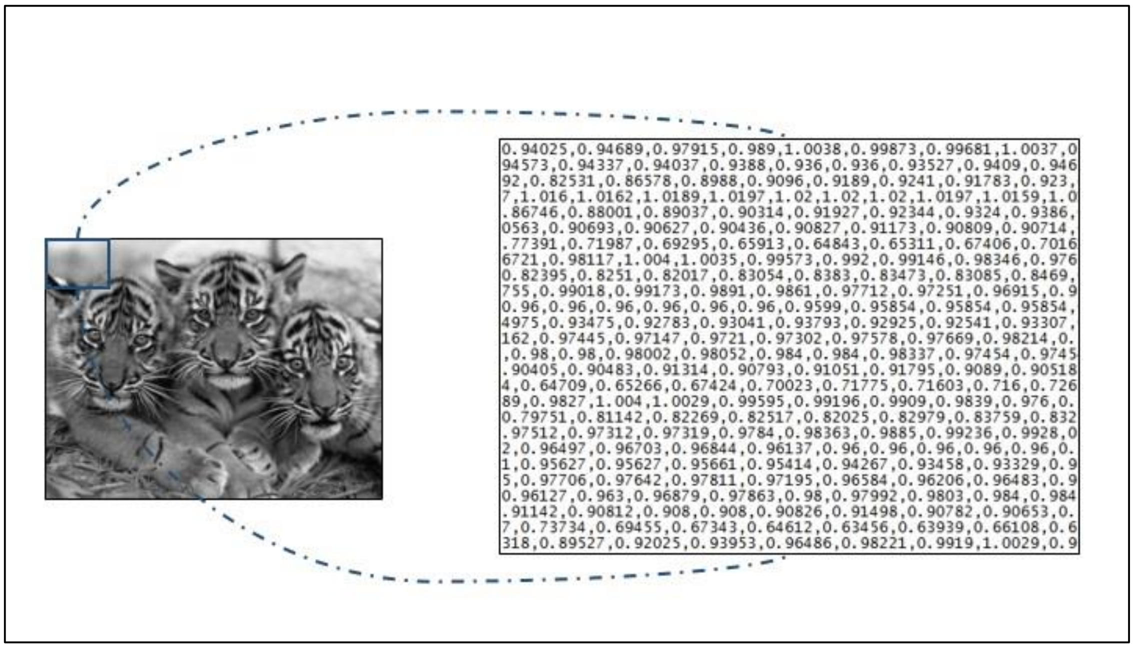
In order to simplify the next processing, we need to convert a color image to gray to reduce the dimension, which significantly improves the obfuscating process, due to a smaller volume of data being manipulated. Note that the colored image is represented in a matrix of three dimensions. The color increases the computational complexity of data obfuscation and comparison in later stages. Therefore, we need to convert the colored image to gray one, because the gray image is represented in a two-dimensional matrix, which is much simpler to compute than a three-dimensional matrix. The conversion from color to gray does not affect the comparison because the gray image does not lose vital information needed for the comparison of images. An example of converting colored image to gray image is shown in Table 4.2 below.

Table 4.2: An example of converting colored image to gray image

Colored image:	Gray image:

4.2.3. Extracting pixels values

Because the data obfuscating and comparison processes can only be applied on integer values, we need to derive the pixels of images that are double numbers to convert them to integer numbers later. Note that each pixel is represented originally by an integer value that ranged from 0 to 251, but because we reduce the image size, we cause each pixel to be represented with a double value. In this step, we extract the original double representation of each pixel. Consequently, the final result of this step is a matrix of double values keeping the same size and dimensions of the image's reduced size. This transformation significantly reduces the data volume of the original matrix of pixels of the color image. Figure 4.3 shows a sample of extracting pixel values.



```
0.94025,0.94689,0.97915,0.989,1.0038,0.99873,0.99681,1.0037,0
94573,0.94337,0.94037,0.9388,0.936,0.936,0.93527,0.9409,0.946
92,0.82531,0.86578,0.8988,0.9096,0.9189,0.9241,0.91783,0.923,
7,1.016,1.0162,1.0189,1.0197,1.02,1.02,1.02,1.0197,1.0159,1.0
.86746,0.88001,0.89037,0.90314,0.91927,0.92344,0.9324,0.9386,
0563,0.90693,0.90627,0.90436,0.90827,0.91173,0.90809,0.90714,
.77391,0.71987,0.69295,0.65913,0.64843,0.65311,0.67406,0.7016
6721,0.98117,1.004,1.0035,0.99573,0.992,0.99146,0.98346,0.976
0.82395,0.8251,0.82017,0.83054,0.8383,0.83473,0.83085,0.8469,
755,0.99018,0.99173,0.9891,0.9861,0.97712,0.97251,0.96915,0.9
0.96,0.96,0.96,0.96,0.96,0.96,0.9599,0.95854,0.95854,0.95854,
4975,0.93475,0.92783,0.93041,0.93793,0.92925,0.92541,0.93307,
162,0.97445,0.97147,0.9721,0.97302,0.97578,0.97669,0.98214,0.
,0.98,0.98,0.98002,0.98052,0.984,0.984,0.98337,0.97454,0.9745
.90405,0.90483,0.91314,0.90793,0.91051,0.91795,0.9089,0.90518
4,0.64709,0.65266,0.67424,0.70023,0.71775,0.71603,0.716,0.726
89,0.9827,1.004,1.0029,0.99595,0.99196,0.9909,0.9839,0.976,0.
0.79751,0.81142,0.82269,0.82517,0.82025,0.82979,0.83759,0.832
.97512,0.97312,0.97319,0.9784,0.98363,0.9885,0.99236,0.9928,0
2,0.96497,0.96703,0.96844,0.96137,0.96,0.96,0.96,0.96,0.96,0.
1,0.95627,0.95627,0.95661,0.95414,0.94267,0.93458,0.93329,0.9
5,0.97706,0.97642,0.97811,0.97195,0.96584,0.96206,0.96483,0.9
0.96127,0.963,0.96879,0.97863,0.98,0.97992,0.9803,0.984,0.984
.91142,0.90812,0.908,0.908,0.90826,0.91498,0.90782,0.90653,0.
7,0.73734,0.69455,0.67343,0.64612,0.63456,0.63939,0.66108,0.6
318,0.89527,0.92025,0.93953,0.96486,0.98221,0.9919,1.0029,0.9
```

Figure 4.3: Extracting pixels value

4.2.4. Transforming decimals to integer values

Our obfuscating technique can only be applied to integer values, so the final resulting matrix of double values from the previous stage must be converted into an integer matrix. And because the pixel value is accurately lying in the range from 0 to 251, then the integer value must be in the range from 0 to 251. In order to achieve this, we apply the following formula to each value of the resulting double matrix:

$$\lfloor d * 251 \rfloor \qquad (1)$$

Where `d` is the decimal value of the matrix. The final result of this step is an integer matrix of the same size and dimensions of the double matrix. An example of converting a double to an integer is: let `d = 0.94208`, then by applying (1)

$$\lfloor 0.94208 * 251 \rfloor = \lfloor 236.462 \rfloor = 236.$$

By the end of this step, the data is modified and processed by the next data-obfuscating technique.

4.3. Obfuscating

The heart of our approach is the data obfuscating that must be uncompromising in regard to ensuring data confidentiality and privacy. The input to the obfuscating process is an integer matrix resulting from the preprocessing stage. The output of this process is the obfuscated version of the integer matrix called an obfuscated matrix. Note that the obfuscated matrix is a matrix of an object, and each of its object elements consists of two double numbers, where the first number is called encrypt, while the second number is called token. The encrypt number consists of two numbers: the result of encrypting block that lies in the integer part, and the hashed digit counter that lies in the decimal part of the encrypt number. For example, the objects of the encrypted matrix are (113.6, 18.3), then the encrypt number is 113.6. The result of the encrypting block is 113 and the hashed digit counter is 6, where the token is 18.3. In the next section, a brief explanation of the final resulting matrix is presented.

Our obfuscating process uses the Rabin cryptosystem to encrypt data as in [37, 13, 21, 27, 31, 6, 48, 5]. The main idea behind the Rabin cryptosystem is based on computing a composite number for the quadratic congruence modulo. Solving the quadratic congruence is simple when the factorization of the composite number is known, but doing so is very complex when the factorization of the composite number is unknown. Therefore, the Rabin cryptosystem is an asymmetric system that requires two different types of keys: a public key (which is the composite number) and a private key (which is the factorization of the composite number). The public key is needed to encrypt plain data, and the private key is needed to decrypt the encrypted data. Note that the public key is a distributed key to every user so that they can encrypt their data, while the private key is not distributed to any users, and it must be unknown and hidden from other users, except the user who owns it. However, in our approach we use only the private key for both processes of obfuscating and de-obfuscating, and the public key is not needed. The main advantage of excluding the public key is that hackers have no idea about the composite number, so they will not have any knowledge about its factorization. The obfuscating approach without using the public key increases the hardness and security. Another advantage is that we do not need to pay much attention to how hardness is factoring the composite number as reported in [27, 31, 4, 30]. Note that if the composite number is factorized, then there is a high chance that the obfuscated data can be compromised.

In our approach, the obfuscating technique is not entirely based on the Rabin cryptosystem, because it has some drawbacks. Consequently, in order to boost and support the Rabin crypto system, we propose some new contributions that address the drawbacks of the Rabin cryptosystem and improve the confidentiality and the privacy of data. In addition, our new contributions decrease the computational complexity of the data-obfuscating process.

The proposed obfuscation process is based on six different techniques:

1. Creating blocks
2. Developing digit counts
3. Tokenization
4. Identifying sets
5. Creating private keys

6. Obfuscating

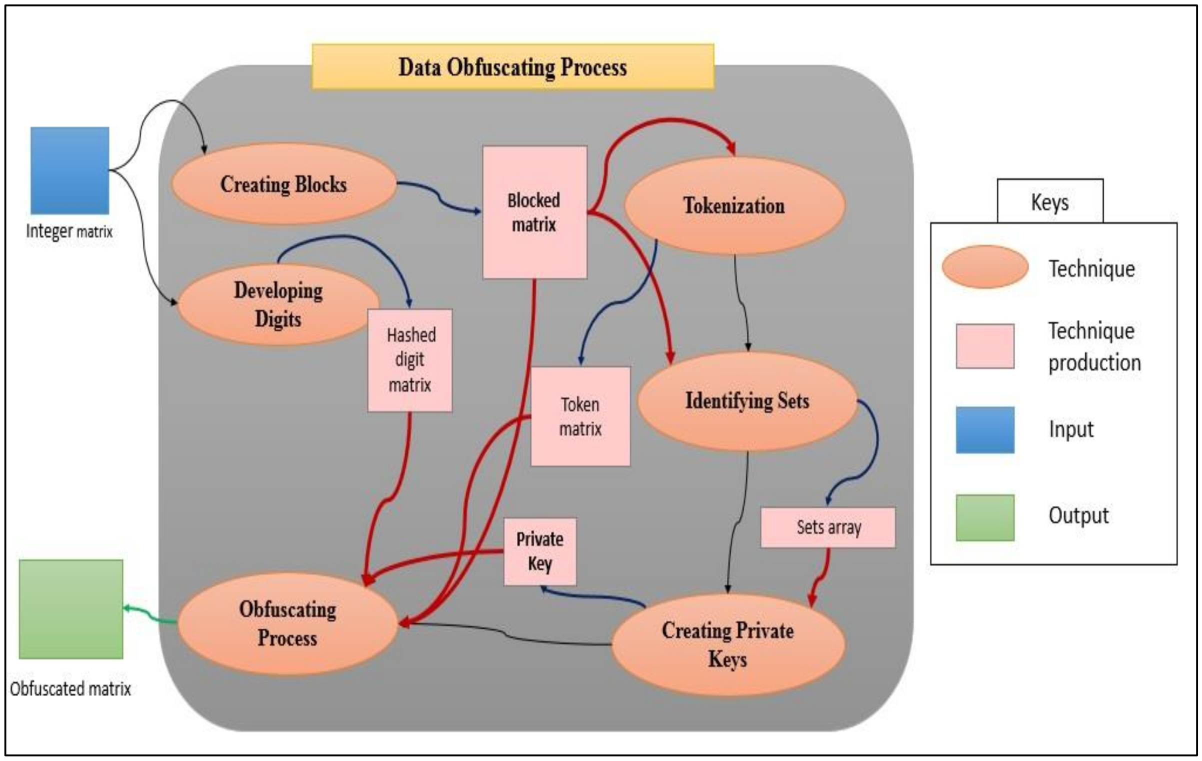


Figure 4.4: Depicts the Details and Purpose of Data-Obfuscating Techniques

The data-obfuscating process is as follows:

1- The input integer matrix is processed by two techniques: creating blocks (which produces the blocked matrix) and developing digits (which produces the hashed digit matrix).

2- The blocked matrix is processed by the tokenization technique, which produces the token matrix.

3- The blocked matrix is grouped into sets; the identifying-sets technique produces a sets array.

4- The private key is created by the sets array.

5- The blocked matrix is obfuscated by the obfuscating technique that uses the hashed digit matrix, token matrix, and private key.

The details and purpose of each of these six techniques are described in the following subsections.

4.3.1 Creating blocks

The block cipher is a cryptosystem that combines the plain data to create a fixed length of bits, called blocks [8]. In our approach, the technique for creating blocks has been inspired from the block cipher field. For our contribution, instead of providing a fixed length of bits in each block, we propose a length of the block without a fixed size, meaning that our approach supports a flexible length of blocks, as opposed to using the fixed block sizes in block cipher. The created blocks are not of fixed size, yet they are created from combining two columns of content of the matrix, but all of this content is not of the same size.

Creating blocks of data is a technique used to combine the content of two columns of matrix together. As a result, each block contains two integer numbers combined horizontally, and both of them are represented as if they were only one number. The main purpose of our blocking technique is to reduce the size of actual data in the matrix by half without losing vital information, in order to minimize the computational complexity of our approach. The size of the resulting block matrix is equal to

$$\left(\frac{Column}{2}\right) * row \qquad (2)$$

Where the column is the number of column in the matrix, 2 is the size of each block, and row is the number of rows in the matrix. Consider an example of creating blocks. Let M designate a matrix of size 6 * 6 = 36 as follows:

$$M = \begin{bmatrix} 169 & 11 & 1 & 77 & 117 & 46 \\ 21 & 37 & 200 & 120 & 240 & 60 \\ 143 & 235 & 41 & 59 & 12 & 170 \\ 74 & 118 & 147 & 228 & 207 & 148 \\ 75 & 178 & 103 & 214 & 146 & 63 \\ 225 & 147 & 57 & 85 & 115 & 0 \end{bmatrix} \Rightarrow BM \begin{bmatrix} 16911 & 177 & 11746 \\ 2137 & 200120 & 24060 \\ 143235 & 4159 & 12170 \\ 74118 & 147228 & 207148 \\ 75178 & 103214 & 14663 \\ 222147 & 5785 & 1150 \end{bmatrix}$$

Where BM is the resulting block matrix, and its size is = (6/2) * 6 = 18. The size of the original matrix is reduced to half by using this blocking technique.

4.3.2 Developing digit counts

The digit count technique is applied during the creating of the block matrix. It is one of our contributions in this framework. We use two techniques: digit counter, and a digit hash map counter. Both of these techniques are briefly described in the next subsection in detail. The main objective of this technique is to save information about the matrix values, so that later when the obfuscated matrix is de-obfuscated, the resulting plain data will not lose any of its original form. If the block is decomposed into different numbers than the others that have formed it, then it will lose in the final information. For example, if the two numbers that form a block are 116 and 71, then the block is equal to 11671, so in order to break it the resulting two numbers must be 116 and 71—not 11 and 671—because this would give wrong data that would cause a loss of the de-obfuscated information.

Digit counter is a technique that counts the number of digits for each integer number of the matrix before it is combined in the block matrix. On the other hand, the digit hash map counter is a hash map that contains all possible numbers of digits in each content of the matrix, which is used to set and identify the digit factor for the forthcoming encrypted matrix. Note that the number of entries in the digit hash map counter depends on both the number of content in each block and the maximum number of digits of the content. For example, the size of the block in our approach is equal to 2, where the maximum number of digits for each content is equal to 3 digits because the integer number in the matrix is in the range 0 to 251. The number of entries in the digit hash map counter is defined by the formula

$$(\text{maximum number of digits})^{\text{size of block}} \qquad (3)$$

Therefore, by applying (3), the number of the entries of the digit hash map counter in our approach is 32, that is, a total of 9 entries. The digit hash map counter of our approach is represented in Table 4.3.

Table 4.3: Digit hash map counter

Key	Digit counter	Description
0.1	11	Each piece of content in the block has one digit.
0.2	12	The first content of the block has 1 digit, while the other content has 2 digits.
0.3	13	The first content of the block has 1 digit, while the other content has 3 digits.
0.4	21	The first content of the block has 2 digits, while the other content has 1 digit.
0.5	22	Each piece of content in the block has two digits.
0.6	23	The first content of the block has 2 digits, while the other content has 3 digits.
0.7	31	The first content of the block has 3 digits, while the other content has 1 digit.
0.8	32	The first content of the block has 3 digits, while the other content has 2 digits.
0.9	33	Each piece content in the block has three digits.

Consider the example presented in Section 4.3.1 (creating blocks). The digit counter is defined by counting the digits of each block element, and after hashing the digit counter, the digit hash matrix, DH, is defined as follows:

$$Digit\ block\ matrix\ DBM\ = \begin{bmatrix} 32 & 12 & 32 \\ 22 & 33 & 32 \\ 33 & 23 & 23 \\ 23 & 33 & 33 \\ 23 & 33 & 32 \\ 33 & 22 & 31 \end{bmatrix} \xRightarrow{After\ hashing\ it} DH = \begin{bmatrix} 0.8 & 0.2 & 0.8 \\ 0.5 & 0.9 & 0.8 \\ 0.9 & 0.5 & 0.6 \\ 0.6 & 0.9 & 0.9 \\ 0.6 & 0.9 & 0.8 \\ 0.9 & 0.5 & 0.7 \end{bmatrix}$$

The digit block matrix DBM is a matrix of integer elements because it is of the same size of block matrix BM (see Section 4.2.1). The DBM content has been defined by counting the digits of plain matrix contents. For example M[0][0] = 169, and M[0][1] = 11; then DBM [0][0] = 32 so the DH[0][0] = 0.8, because M[0][0] content consists of an integer number that has three digits, where M[0][1] content consists of an integer number that has two digits. Because the block size is two, the content of DBM matrix is for only two elements of matrix M.

4.3.3. Tokenization

One of the main drawbacks of the Rabin cryptosystem is that it is not a one-to-one function; whereas, the result of its decryption technique is four corrects roots, while one of these four resulting roots represents the correct value of plain data. In order to define the correct root in our approach, we need a different technique. We propose the following to address Rabin’s decryption problem with the purpose of de-obfuscating the obfuscated matrix accurately and without having any problem or confusion. Recall that we need the de-obfuscating stage to verify the result sent by the server.

Our proposed idea is to first compute a token for each integer number in the block matrix. The token is computed by adding the content of the digits of integer numbers of the block matrix BM element, and by counting the total number of these digits. A token is a decimal number; its integer part is the result of adding the digits of elements of the block matrix BM, while its decimal part is the total number of digits.

For each block, there is a token; then we need to have a matrix of the same size as the block matrix to save each block token. Consequently, in order to save space complexity and to add more flexibility to our obfuscating approach, we concatenate the token part as content of the obfuscated matrix, which is ultimately sent to the cloud server. By doing this, the only information that the client needs to save is the private keys that were used to obfuscate the data. The explanation of the obfuscated matrix and its content is provided in a later section. The usage of tokens is briefly described in the de-obfuscating section.

Consider the previous example cited in Section 4.3.1 about creating blocks.

$$BM = \begin{bmatrix} 16911 & 177 & 11746 \\ 2137 & 200120 & 24060 \\ 143235 & 4159 & 12170 \\ 74118 & 147228 & 207148 \\ 75178 & 103214 & 14663 \\ 222147 & 5785 & 1150 \end{bmatrix}$$

An example of defining the token is the following:

The first block of BM is 16911 $\Longrightarrow$ token integer part = 1+6+9+1+1 = 18; the total number of digits = 5 $\Longrightarrow$ token = 18+ 0.5 = 18.5.

The second block is 177 $\Longrightarrow$ token integer part = 1 + 7 + 7 = 15; the total number of digits = 3 $\Longrightarrow$ token = 15+ 0.3= 15.3.

By computing the same operation for each block, the final resulting token matrix (TM) of BM is,

$$token\ matrix(TM) = \begin{bmatrix} 18.5 & 15.3 & 19.5 \\ 13.4 & 5.6 & 12.5 \\ 18.6 & 19.4 & 11.5 \\ 21.5 & 24.6 & 22.6 \\ 28.5 & 11.6 & 20.5 \\ 18.6 & 25.4 & 7.4 \end{bmatrix}$$

4.3.4. Identifying sets

Identifying sets are one of the contributions of our approach. The main advantage of using sets is to decrease the size of total private keys, so rather than having a private key for each block, we will have a private key for each set, which consists of several blocks. We increase the security rate by having more than one private key for obfuscating and de-obfuscating all of the blocked matrix; thus, if hackers discover an element of a private key, they cannot de-obfuscate all of the obfuscated data. Furthermore, the random size of sets fixes the limitation of the fixed size of the block, because all blocks must be of the same size. So this technique increases the protection of the final obfuscated data and decreases the computational complexity of the obfuscating process by reducing the total number of generated and stored private keys.

This technique is based on dividing the blocked matrix into sets, each with a different random size. The minimum size of a set is a single value, and the maximum size is half of the total size of the block matrix. However, the range of the set size can be modified depending on the user needs, so if the user wants to have more sets, the maximum bound of the set size can be decreased to one-third or one-fourth of the total size of the block.

The number of total sets correlates with the data-security implications. Security is usually higher if more sets are created, but the overhead costs of processing would be higher as well. Subsequently, the total number of sets is directly proportional to security implications. The number of sets is also directly proportional to the computational complexity. The increasing number of sets may require higher computational efforts because each set would have its own private key used to obfuscate the set, but the security would be better.

We use a set array in order to capture the total number of sets. The length of the set array is equal to the total number of sets, while its content is the size of the set. For example, if a block matrix consists of five sets, then the set array has a length of five, and each value represents the size of each set.

Consider the previous example cited in Section 4.3.1 about creating blocks

$$BM = \begin{bmatrix} 16911 & 177 & 11746 \\ 2137 & 200120 & 24060 \\ 143235 & 4159 & 12170 \\ 74118 & 147228 & 207148 \\ 75178 & 103214 & 14663 \\ 222147 & 5785 & 1150 \end{bmatrix}$$

The range of set size is defined using the following equation:

$$[1\,,\text{size of BM}/2] \implies \left[1, \frac{18}{2}\right] \implies [1, 9]$$

The smallest set will contain only one block, while the largest set will contain nine blocks. We then assign random numbers to set sizes, so that the total summation of them is equal to the size of BM. Assigning the random numbers can be done as shown below:

S[0] size = 2, S[1] size = 8, S[3] size = 8, $\because \sum S[i]size = 18 \implies$

$we\ will\ have\ only\ 3\ sets,$

where S[i] represents the set number.

The content of each set will be as follows:

S[0] = {16911, 177}

S[1] = {11746, 2137, 200120, 24060, 143235, 4159, 12170, 74118}

S[3] = {147228, 207148, 75178, 103214, 14663, 222147, 5785, 1150}

4.3.5. Creating Private Keys

One of the main objectives of creating sets is to specify the number of private keys, which is equal to the number of sets. For example, if we have three sets, then we will have three private keys—one key for each set. An array of private keys stores all keys, and its length is equal to the length of set array. The private key array consists of three main factors: PK (p, q, s). The first two factors, p and q, are the factorization of the composite number (as mentioned previously in the obfuscating section earlier), and the third factor, s, is the size of the set that uses that private key.

Because the main obfuscating process is based on the Rabin crypto system, in order to obfuscate data we need a composite number that consists of two large prime numbers, p and q. These two prime numbers must be private. Otherwise, disclosing these will result in breaking of the Rabin cryptosystem. The third factor of the private key is necessary to identify the sets and to know which set should be obfuscated or de-obfuscated by these two prime numbers. Therefore, for each set, we will generate random large prime numbers. Note that in order to identify these two large prime numbers' range, we must first consider what is the largest integer number of the data. For example, in our approach, the largest integer number from the integer matrix is 251, which means that the largest block is 251251. Consequently, in order to define these two prime numbers, p and q, their range must start form larger number than 251. So in order to evaluate the quadratic congruence of each block, the composite number formed from p and q must be larger than 251251. The composite number should not be too large so that when the block is encrypted by the quadratic formula, it will not change its actual value. For this reason, we discover a range of prime numbers for p and q, and the range should achieve the previously mentioned conditions. This range goes from 700 to 1500. For example, if both prime numbers are 700, then the composite number is 700 * 700 = 490000, which is larger than 251251. If both prime numbers are 1500, then the composite number is 1500 * 1500 = 2250000, so it can obfuscate blocks with small integer values.

Consider the previous example discussed in Section 4.3.4 (identifying sets), in which the form of the set was

S[0] size = 2, S[1] size = 8, S[3] size = 8

That means we have to randomly generate two different prime numbers for p and q, for each set, both in the range of [700, 1500]. Let the result of this calculation be as follows:

for S[0] $\Longrightarrow$ p = 643, q = 691, where N = p * q = 444313

for S[1] $\Longrightarrow$ p = 563, q = 683, where N = p * q = 384529

for S[2] $\Longrightarrow$ p = 631, q = 743, where N = p * q = 468833

Where N is the composite number.

Finally, the resulted array of private key is as follows:

$$PK = \begin{bmatrix} 643, 691, 2 \\ 563, 683, 8 \\ 631, 743, 8 \end{bmatrix}$$

4.3.6. Obfuscating process

Because our obfuscating process is based on the Rabin cryptosystem to encrypt data, the formula of the Rabin cryptosystem is

$$C \equiv P^2 (mod\ N) \qquad (4)$$

In this formula, C is the cipher data, P is the plain data, N is the composite number that is produced from multiplying the two distinct large primes, p and q. Note that the strength of the Rabin cryptosystem depends on the hardness of the factorization of the composite number.

In our approach, obfuscating plain data is not the only part of the process; rather, after encrypting each block of the blocked matrix, we add a hashed digit counter from the digit hashed counter matrix to the encrypted data. In addition, we concatenate each encrypted block with its related token. The final result of all of these operations is a matrix of obfuscated data, which is in the form of enc(C.dh, t); here C is the encrypted data, dh is the digit counter data, and t is the token of that obfuscated data.

An example of the obfuscating process and the obfuscated matrix is shown as follows.

Consider the previous example presented in Section 4.3.1 (the creating block), where the PK for BM matrices are

$$BM = \begin{bmatrix} 16911 & 177 & 11746 \\ 2137 & 200120 & 24060 \\ 143235 & 4159 & 12170 \\ 74118 & 147228 & 207148 \\ 75178 & 103214 & 14663 \\ 222147 & 5785 & 1150 \end{bmatrix}$$

$$PK = \begin{bmatrix} 643, 691, 2 \\ 563, 683, 8 \\ 631, 743, 8 \end{bmatrix}$$

Now we will encrypt BM matrix depending on *PK*, such that the first two blocks are encrypted by making p = 643, and q = 691 ⟹ N = 444313

$$\Longrightarrow C = (16911)^2 \ (mod\ 444313) = 288662$$

And the same operation is applied to the next block.

Then the next eight blocks starting from BM[0][2] = 11746 to BM[3][0] = 74118 are encrypted by making p = 563, and q = 683 ⟹ N = 384529

$$\Longrightarrow C = (11746)^2 \ (mod\ 384529) = 307134$$

And the same operation is applied to the next remaining seven blocks.

The next eight blocks starting from BM[3][1] = 147228 to BM[5][2] = 1150, are encrypted by making p = 631, and q = 743 ⟹ N = 468833

$$\Longrightarrow C = (147228\,)^2 \ (mod\ 468833) = 59062$$

And the same operation is applied to the next remaining seven blocks.

Then the final resulting encrypted matrix encBM is

$$encBM = \begin{bmatrix} 288662 & 31329 & 307134 \\ 336950 & 88108 & 167455 \\ 220937 & 378005 & 65235 \\ 96630 & 59062 & 353579 \\ 418702 & 306370 & 278055 \\ 396862 & 179082 & 384834 \end{bmatrix}$$

By adding the DH matrix to the encBM matrix, we get the following obfuscated BM matrix:

$$DH + encBM = \begin{bmatrix} 0.8 & 0.2 & 0.8 \\ 0.5 & 0.9 & 0.8 \\ 0.9 & 0.5 & 0.6 \\ 0.6 & 0.9 & 0.9 \\ 0.6 & 0.9 & 0.8 \\ 0.9 & 0.5 & 0.7 \end{bmatrix} + \begin{bmatrix} 288662 & 31329 & 307134 \\ 336950 & 88108 & 167455 \\ 220937 & 378005 & 65235 \\ 96630 & 59062 & 353579 \\ 418702 & 306370 & 278055 \\ 396862 & 179082 & 384834 \end{bmatrix}$$

$$= \begin{bmatrix} 288662.8 & 31329.2 & 307134.8 \\ 336950.5 & 88108.9 & 167455.8 \\ 220937.9 & 378005.5 & 65235.6 \\ 96630.6 & 59062.9 & 353579.9 \\ 418702.6 & 306370.9 & 278055.8 \\ 396862.9 & 179082.5 & 384834.7 \end{bmatrix}$$

The final step is to concatenate the encBM matrix with the generated token matrix (which has already been described earlier in the tokenization Section 4.3.3 earlier in this chapter) to create the final obfuscated matrix. The final form of encBM matrix is

$$encBM = \begin{bmatrix} 288662.8, 18.5 & 31329.2, 15.3 & 307134.8, 19.5 \\ 336950.5, 13.4 & 88108.9, 5.6 & 167455.8, 12.5 \\ 220937.9, 18.6 & 378005.5, 19.4 & 65235.6, 11.5 \\ 96630.6, 21.5 & 59062.9, 24.6 & 353579.9, 22.6 \\ 418702.6, 28.5 & 306370.9, 11.6 & 278055.8, 20.5 \\ 396862.9, 18.6 & 179082.5, 25.4 & 384834.7, 7.4 \end{bmatrix}$$

This is the final obfuscated matrix that is sent to the cloud server for further processing, such as comparison with similar data or storage depending on the user's needs.

4.4. De-obfuscating

The de-obfuscating process is only needed when the client wants to verify the computational results of the server. The verification technique is done only when the server claims that the print image of the input finger was matched with one of the stored and obfuscated fingerprint images. In that case, the server returns a copy of the obfuscated matrix of the matched image, which allows the client to de-obfuscate it and compare it with the original integer matrix of the input fingerprint image stored by the client.

As we explained earlier, our approach encrypts integer numbers by using a quadratic congruence formula, which can be decrypted by using the Chinese remainder theorem. One of the algorithms used to solve the Chinese remainder theorem is the Garner algorithm [5]. In our approach, we use the Garner algorithm to decrypt encrypted data because it is the simplest algorithm that can solve the Chinese remainder theorem with less computational complexity, and it reduces by half the total multiplications needed for Gauss's algorithm, which has been normally used to solve the Chinese remainder theorem.

In order to correctly de-obfuscate the data matrix, we use every factor in both the private keys and obfuscated matrix, because every factor has a role to play in the de-obfuscating process. Note that a private key object is built with three factors, and these factors are the large prime numbers p and q, and the set size factor. The set size factor allows us to know which set-to-be de-obfuscates with the specific p and q numbers. Each value of the obfuscated matrix is an obfuscated object that consists of two double numbers. The first double number represents both the encrypted integer number that is replaced in the integer part and the digit counter, which is the double part. The second double number represents the token factor. Recall that the form of objects of obfuscated matrix encBM is enc(C.dh, t), where C is the cipher data, dh is the digit counter data, and t is the token of that obfuscated data.

An example of de-obfuscation is presented with the previous data, where PK contains private keys, and encBM is the obfuscated matrix:

$$PK = \begin{bmatrix} 643, 691, 2 \\ 563, 683, 8 \\ 631, 743, 8 \end{bmatrix}, \quad encBM = \begin{bmatrix} 288662.8, 18.5 & 31329.2, 15.3 & 307134.8, 19.5 \\ 336950.5, 13.4 & 88108.9, 5.6 & 167455.8, 12.5 \\ 220937.9, 18.6 & 378005.5, 19.4 & 65235.6, 11.5 \\ 96630.6, 21.5 & 59062.9, 24.6 & 353579.9, 22.6 \\ 418702.6, 28.5 & 306370.9, 11.6 & 278055.8, 20.5 \\ 396862.9, 18.6 & 179082.5, 25.4 & 384834.7, 7.4 \end{bmatrix}$$

The first step of the de-obfuscating process is to extract the integer number from the first double number in the encBM matrix:

The obfuscated factor of encBM[0][0] = 288662.8 ➔ the integer part of the factor encBM[0][0] = 288662.

Then the resulted integer number is decrypted by using PK[0], since encBM[0][0] belongs to the first set. The set size factor in PK[0] = 2, which means that the first two elements of the encBM matrix are decrypted using PK[0].

By using the Garner algorithm and making $p = 643$ and $q = 691$, the results of de-obfuscation are these following four roots:

Sol = {16911, 81211, 363102, 427402}. In order to identify the correct plain data of our obfuscated matrix, the token part is used, where encBM[0][0] token = 18.5.

In order to do so, we will define the token for each defined root such as:

16911 token = 18.5, because 1+6+9+1+1 = 18, and it consists of 5 digits

81211 token = 13.5, because 8+1+2+1+1 = 13, and it consists of 5 digits

363102 token = 15.6, because 3+6+3+1+0+2 = 15, and it consists of 6 digits

427402 token = 19.6, because 4+2+7+4+0+2 = 19, and it consists of 6 digits

- ⇨ The correct plain data root is 16911 because its token is equal to the token part of encBM[0][0]

The final step in de-obfuscating is to break the resulting block, which can be done by using the double part of the obfuscated number of encBM[0][0], which was 0.8. Consider the digit hash map counter in the developing digit-counts technique already discussed in Section 4.3.2. The value 0.8 in encBM[0][0] was the hashed key of the digit counter 32. This means that the first part of the block consists of a three-digit integer number, while the second part of the block consists of a two-digit integer number. As a result, the 16911 blocked number is decomposed into 169 and 11. So the first two contents of the de-obfuscated matrix are dec[0][0] = 169 and dec[0][1] = 11.

Then by repeating each of these steps for each encBM element, we get the following resulted dec matrix:

$$dec = \begin{bmatrix} 169 & 11 & 1 & 77 & 117 & 46 \\ 21 & 37 & 200 & 120 & 240 & 60 \\ 143 & 235 & 41 & 59 & 12 & 170 \\ 74 & 118 & 147 & 228 & 207 & 148 \\ 75 & 178 & 103 & 214 & 146 & 63 \\ 225 & 147 & 57 & 85 & 115 & 0 \end{bmatrix}$$

Note that dec matrix is exactly equal to the plain data M matrix that was shown as an example in the earlier section for creating blocks. The elements of the M matrix are:

$$M = \begin{bmatrix} 169 & 11 & 1 & 77 & 117 & 46 \\ 21 & 37 & 200 & 120 & 240 & 60 \\ 143 & 235 & 41 & 59 & 12 & 170 \\ 74 & 118 & 147 & 228 & 207 & 148 \\ 75 & 178 & 103 & 214 & 146 & 63 \\ 225 & 147 & 57 & 85 & 115 & 0 \end{bmatrix}$$

The main goals of this approach is to secure the client’s outsourced data without using public key infrastructure and to allow the cloud server to do the computations on the obfuscated data. The presented techniques in this chapter show how to achieve this goal. Some of the existing techniques have drawbacks, so we show how we modified some of them to overcome those drawbacks. The new techniques that we have introduced are the blocking technique, digit counter technique, creating sets of blocks technique, and tokenization technique. In the next chapter, we present the implementation details of these processes and techniques of our framework.

CHAPTER 5: IMPLEMENTATION AND VALIDATION OF THE APPROACH

In this chapter, we develop and use various algorithms and techniques to implement the proposed framework. The approach is flexible and can be applied to any form or representation of numerical data. In this thesis, we apply our approach to images to study its applicability. This chapter describes how we first process the images to make them suitable for being obfuscated by the obfuscating process. We also demonstrate how the images are obfuscated using our techniques discussed earlier. After obfuscating the images, they are sent to the server that processes images by storing and comparing the input images with the stored obfuscated images. The server finally returns the comparison result of obfuscated images along with a copy of the obfuscated image that was found matched with the input image.

The proposed application presented in this chapter is an image-comparison application, which is a secure fingerprint checking system. This application captures the user's fingerprint, obfuscates it, and sends it to the cloud server. The server stores all users' obfuscated fingerprints if not already stored. For this purpose, it first compares the input fingerprint with the already stored obfuscated fingerprints. The comparison process starts when a user enters his or her fingerprint into the system to check it if it matches with any image stored in the server. The server checks if a similar fingerprint exists by comparing it with the stored fingerprints in the server. After checking and comparing the new image with the stored ones, the server returns the result to the client. The result is either "Exist" if the fingerprint matches with one of the stored ones, otherwise it returns "Not Exist."

However, if the user is new, the first step is to compare his or her fingerprint with the stored fingerprints; if no match is found, the server stores the new fingerprint, so it can verify the fingerprint when the user enters his or her fingerprint the next time.

This application can be used for checking the authenticity of a user accessing data, systems, activities, and so forth. It can also record employees' attendance by using image recognition. This application is very useful in many areas.

In order to implement the secure fingerprint comparison and storing system, the client must have image-processing software, in order to prepare and process images for making them applicable to be obfuscated. We use Java and Matlab for implementing our framework, because they offer many efficient functionalities that can optimize the client's computational complexity and resources.

The following sections present the algorithms and techniques for implementing secure fingerprint comparison and storing processes.

5.1. Application Usage

A secure fingerprint checking system consists of two main use-cases: fingerprint checking and storing obfuscated fingerprint image.

The first use-case is the storing of the obfuscated fingerprint; the admin of the system collects the needed fingerprints form all of the users who are going to use the system, and obfuscates them by using the secure fingerprint checking system, in which the obfuscating process is done on the client's side. Then all of the selected obfuscated fingerprint images are sent to the server in order to be stored on the cloud.

The second use-case is fingerprint checking. The user enters his or her fingerprint into the system. The system obfuscates it and sends it to the server, which checks whether the user is authorized or not. A user is considered authorized if his or her fingerprint is already stored in the server; an unauthorized user does not have his or her fingerprint stored in the system. The server checks if a similar fingerprint exists by comparing it with the stored fingerprints in the server. After checking and comparing the new image with those stored, the server returns the result to client. The result is either “Exist” if the fingerprint matches one of the stored ones; otherwise the server returns “Not Exist.” If the result is the former, then the server returns the stored obfuscated fingerprint, which matches the received one, to the client. After receiving the server’s comparison result, if the result is “Exist” the client de-obfuscates the received fingerprint image.

Consider that this system has two main threats. The first is if the server claims that the entered fingerprint exists when it actually does not. Therefore, in order to verify the server’s result, we added a final verification process that is executed by the client.

The second threat is the potential for malicious entities to compromise the server; for example, a malicious entity changes the stored and obfuscated data of the fingerprint. As a result of this hypothetical attack, the server results would be only “Not Exist!”—even for the authorized users. To prevent this threat, we allow the server to encrypt its stored data by a public key.

Consider that this application can run on smart phones, tablets, laptops, and desktops. Therefore, optimizing clients’ resources is important in our application; we want to incorporate our application into all types of devices used by clients.

5.2. Implementation of preprocessing

As already mentioned earlier, this component’s main purpose is to prepare plain data to be ready and applicable for being processed in the next stage. Because our plain data is images, image processing is needed for implementing the data-preparation process. Matlab offers a good number of image-processing methods. Therefore, we implement our preprocessing in Matlab. The next stage, which is obfuscating fingerprints, has been implemented in Java; the final stages of the data processing and the preparing process also have been implemented in Java.

The preprocessing is used to reduce the image's size so that the size of the final resulting matrix of the preprocessing is much less than the original size of the fingerprint image. Then the preprocessing converts each fingerprint image to a two-dimensional double matrix. Note that a fingerprint image is usually represented in a three-dimensional double matrix. Assume all entered fingerprint images are of the same size, because all fingerprints are recorded by the same fingerprint recorder. This assumption is important to note in this application. This process can handle any types of image, such as jpeg, png, bmp, and gif.

The preprocessing has four stages, and each implements its own processes. Algorithm 5.1 shows how to perform the data-preparation process.

```
Algorithm 5.1 Preprocessing of data
Input: fingerprint image
Output: 2D integer matrix M[1..m][1..n]
Steps:
    1. Select fingerprint image
    2. Reduce image size by using single-level discrete 2-D
       wavelet transform (dwt2) method.
    3. Convert reduced image from three-dimensional image to
       two-dimensional image by using convert RGB image or
       colormap to grayscale (rgb2gray) method
    4. Save 2d image into text file.
    5. Convert 2d double matrix to integer matrix.
```

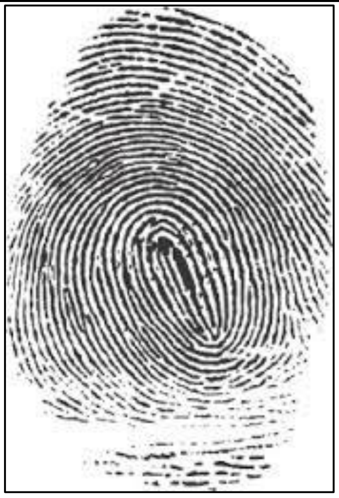
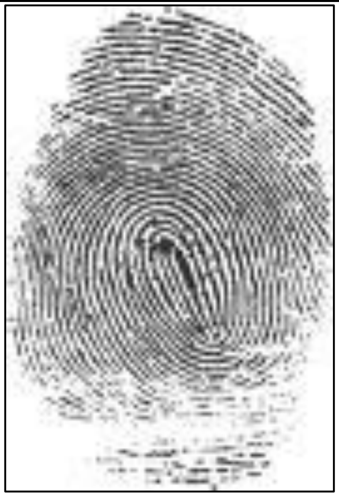
This algorithm has been implemented with a combination of Matlab and Java applications. The four steps in Algorithm 5.1 have been implemented using Matlab because it requires only O(mn) time to generate a 2d matrix from a fingerprint image. Matlab has been used to implement four operations because it offers a good number of high-level image-processing operations. The single-level, discrete, 2-D wavelet-transform (dwt2) method reduces the image size to half of its original size by one single operation. In addition, the conversion of the RGB image or colormap to grayscale (rgb2gray) method converts the three-dimensional image to a two-dimensional image by using the operation "rgb2gray." In addition, to convert image pixels to their original double values, Matlab does so when the image is saved as a text file, and it requires no other operations to complete the task. However, converting a two-dimensional double matrix to an integer-matrix process has been generated by a Java program that requires O(mn) time, where m is the number of rows and n is the number of columns of the double matrix. The values of m and n remain the same in the final resulting integer matrix. Then the required time complexity for implementing this component is O(mn). The following section describes this algorithm in details.

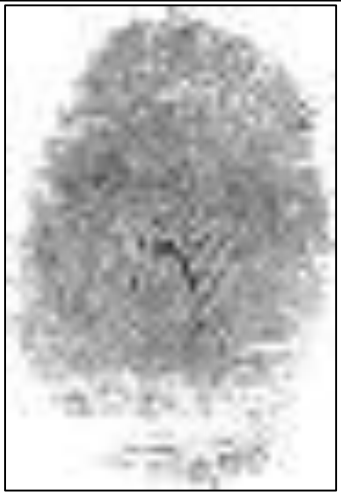
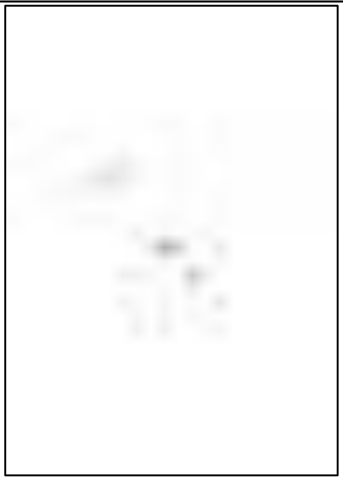
5.2.1. Technique for reducing image size

A fingerprint image of dimensions 255 * 172 pixels, where the width is 172 pixels and the height is 255 pixels, is captured. First, the preprocessing component reduces its size by using the dwt2 method. In order to determine a suitable reduction limit of an image of dimensions 255 * 172 pixels, an experiment was performed. Note that during the process of reducing any image size, we must maintain its accuracy and visibility so that it will not lose any vital information, because the dwt2 method has an inverse relation with both accuracy and visibility of images.

The experiment reduced a fingerprint image by using several cycles of running dwt2 method. The main goal of this method is to reach the most-possible minimum size of the image with dimensions 255 * 172 pixels, without losing vital information of the image. The outcomes of that experiment is shown in table 5.1:

Table 5.1: Image Reducing Experiment Results.

Image Dimensions:	Image	Notes:
255 * 172 pixels		Original fingerprint image before any reductions.
129 * 87 pixels		The resulting image of the first cycle of reducing image size by dwt2 method.

66 * 45 pixels		The resulting image of the second cycle of reducing image size by dwt2 method.
34 * 24 pixels		The resulting image of the third cycle of reducing image size by dwt2 method.

We conclude from this experiment that the best optimal reduction result is achieved by the first cycle of reducing the fingerprint image dimensions. The dimensions of 129 * 87 pixels are half of the original fingerprint image dimensions. This is the most optimal size for image comparison. Although the reduced size from the second cycle looks reasonable, to get a more optimal size, we only reduce the fingerprint image by one cycle.

5.2.2. Technique for converting a color image to gray

In this stage, the color image of the fingerprint is transformed into a gray image. This is done by converting the RGB image or colormap to grayscale using rgb2gray method. Doing so will not only result in a gray image, but this gray image is also automatically transformed from three dimensions to two. This transformation will not have any negative impact on the comparison process because the loss of information is minimal. Note that a fingerprint image contains only two colors, white and black. Therefore, losing the color will not have any impact on the comparison process performed by the cloud server later.

An example of the resulting reduced image, from the previous stage, converted to a gray image is shown in Table 5.2:

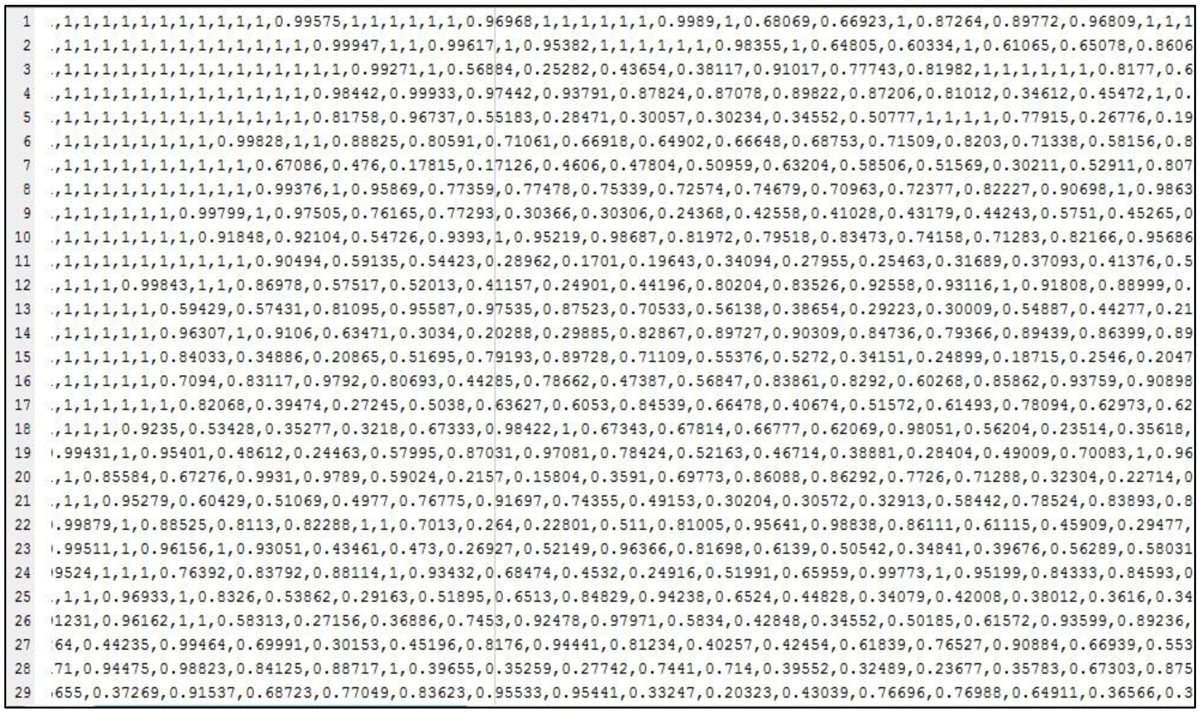
Table 5.2: Image Converted to a Gray Image

Colored image:	Gray image:
Size = 129 * 87 * 3	Size = 129 * 87

From the images in Table 5.2, we conclude that converting the image to gray does not affect much of its size or appearance. The only factor that matters is the image-dimensional factor, which transforms it from a three-dimensional image into two dimensions. Note that the colored image's dimensions are 129 * 87 * 3—that is, the image has three dimensions. However, the size of the gray image is 129 * 87, which means that it has two dimensions.

5.2.3. Extracting double value form pixel

In this stage of the implementation, the gray image from the previous stage is extracted to a double matrix that has two dimensions. This can be done simply in Matlab by saving the image as a text file. The resulting text file includes the two-dimensional matrix representing the image, in which each pixel in the original image is represented by an integer number that ranges from 0 to 251. However, because we reduce the image size, the reduction has an impact on the actual value of the pixel. We transform it from an integer to a double value. We provide a screenshot of transforming the image into a double matrix, as an example. However, only one segment of the double matrix is shown in Figure 5.1, because the actual one too large to be shown in one figure.



```
.,1,1,1,1,1,1,1,1,1,1,1,0.99575,1,1,1,1,1,1,0.96968,1,1,1,1,1,0.9989,1,0.68069,0.66923,1,0.87264,0.89772,0.96809,1,1,1
.,1,1,1,1,1,1,1,1,1,1,1,1,1,0.99947,1,1,0.99617,1,0.95382,1,1,1,1,1,1,0.98355,1,0.64805,0.60334,1,0.61065,0.65078,0.8606
.,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0.99271,1,0.56884,0.25282,0.43654,0.38117,0.91017,0.77743,0.81982,1,1,1,1,1,1,0.8177,0.6
.,1,1,1,1,1,1,1,1,1,1,1,1,1,0.98442,0.99933,0.97442,0.93791,0.87824,0.87078,0.89822,0.87206,0.81012,0.34612,0.45472,1,0.
.,1,1,1,1,1,1,1,1,1,1,1,1,1,0.81758,0.96737,0.55183,0.28471,0.30057,0.30234,0.34552,0.50777,1,1,1,1,0.77915,0.26776,0.19
.,1,1,1,1,1,1,1,1,0.99828,1,1,0.88825,0.80591,0.71061,0.66918,0.64902,0.66648,0.68753,0.71509,0.8203,0.71338,0.58156,0.8
.,1,1,1,1,1,1,1,1,1,1,0.67086,0.476,0.17815,0.17126,0.4606,0.47804,0.50959,0.63204,0.58506,0.51569,0.30211,0.52911,0.807
.,1,1,1,1,1,1,1,1,1,1,0.99376,1,0.95869,0.77359,0.77478,0.75339,0.72574,0.74679,0.70963,0.72377,0.82227,0.90698,1,0.9863
.,1,1,1,1,1,1,0.99799,1,0.97505,0.76165,0.77293,0.30366,0.30306,0.24368,0.42558,0.41028,0.43179,0.44243,0.5751,0.45265,0
.,1,1,1,1,1,1,1,0.91848,0.92104,0.54726,0.9393,1,0.95219,0.98687,0.81972,0.79518,0.83473,0.74158,0.71283,0.82166,0.95686
.,1,1,1,1,1,1,1,1,1,1,0.90494,0.59135,0.54423,0.28962,0.1701,0.19643,0.34094,0.27955,0.25463,0.31689,0.37093,0.41376,0.5
.,1,1,1,0.99843,1,1,0.86978,0.57517,0.52013,0.41157,0.24901,0.44196,0.80204,0.83526,0.92558,0.93116,1,0.91808,0.88999,0.
.,1,1,1,1,1,0.59429,0.57431,0.81095,0.95587,0.97535,0.87523,0.70533,0.56138,0.38654,0.29223,0.30009,0.54887,0.44277,0.21
.,1,1,1,1,1,0.96307,1,0.9106,0.63471,0.3034,0.20288,0.29885,0.82867,0.89727,0.90309,0.84736,0.79366,0.89439,0.86399,0.89
.,1,1,1,1,1,0.84033,0.34886,0.20865,0.51695,0.79193,0.89728,0.71109,0.55376,0.5272,0.34151,0.24899,0.18715,0.2546,0.2047
.,1,1,1,1,1,0.7094,0.83117,0.9792,0.80693,0.44285,0.78662,0.47387,0.56847,0.83861,0.8292,0.60268,0.85862,0.93759,0.90898
.,1,1,1,1,1,1,0.82068,0.39474,0.27245,0.5038,0.63627,0.6053,0.84539,0.66478,0.40674,0.51572,0.61493,0.78094,0.62973,0.62
.,1,1,1,0.9235,0.53428,0.35277,0.3218,0.67333,0.98422,1,0.67343,0.67814,0.66777,0.62069,0.98051,0.56204,0.23514,0.35618,
.99431,1,0.95401,0.48612,0.24463,0.57995,0.87031,0.97081,0.78424,0.52163,0.46714,0.38881,0.28404,0.49009,0.70083,1,0.96
.,1,0.85584,0.67276,0.9931,0.9789,0.59024,0.2157,0.15804,0.3591,0.69773,0.86088,0.86292,0.7726,0.71288,0.32304,0.22714,0
.,1,1,0.95279,0.60429,0.51069,0.4977,0.76775,0.91697,0.74355,0.49153,0.30204,0.30572,0.32913,0.58442,0.78524,0.83893,0.8
.99879,1,0.88525,0.8113,0.82288,1,1,0.7013,0.264,0.22801,0.511,0.81005,0.95641,0.98838,0.86111,0.61115,0.45909,0.29477,
.99511,1,0.96156,1,0.93051,0.43461,0.473,0.26927,0.52149,0.96366,0.81698,0.6139,0.50542,0.34841,0.39676,0.56289,0.58031
9524,1,1,1,0.76392,0.83792,0.88114,1,0.93432,0.68474,0.4532,0.24916,0.51991,0.65959,0.99773,1,0.95199,0.84333,0.84593,0
.,1,1,0.96933,1,0.8326,0.53862,0.29163,0.51895,0.6513,0.84829,0.94238,0.6524,0.44828,0.34079,0.42008,0.38012,0.3616,0.34
1231,0.96162,1,1,0.58313,0.27156,0.36886,0.7453,0.92478,0.97971,0.5834,0.42848,0.34552,0.50185,0.61572,0.93599,0.89236,
64,0.44235,0.99464,0.69991,0.30153,0.45196,0.8176,0.94441,0.81234,0.40257,0.42454,0.61839,0.76527,0.90884,0.66939,0.553
.71,0.94475,0.98823,0.84125,0.88717,1,0.39655,0.35259,0.27742,0.7441,0.714,0.39552,0.32489,0.23677,0.35783,0.67303,0.875
655,0.37269,0.91537,0.68723,0.77049,0.83623,0.95533,0.95441,0.33247,0.20323,0.43039,0.76696,0.76988,0.64911,0.36566,0.3
```

Figure 5.1: A segment of a 2D double matrix.

5.2.4. Method to transform double to integer

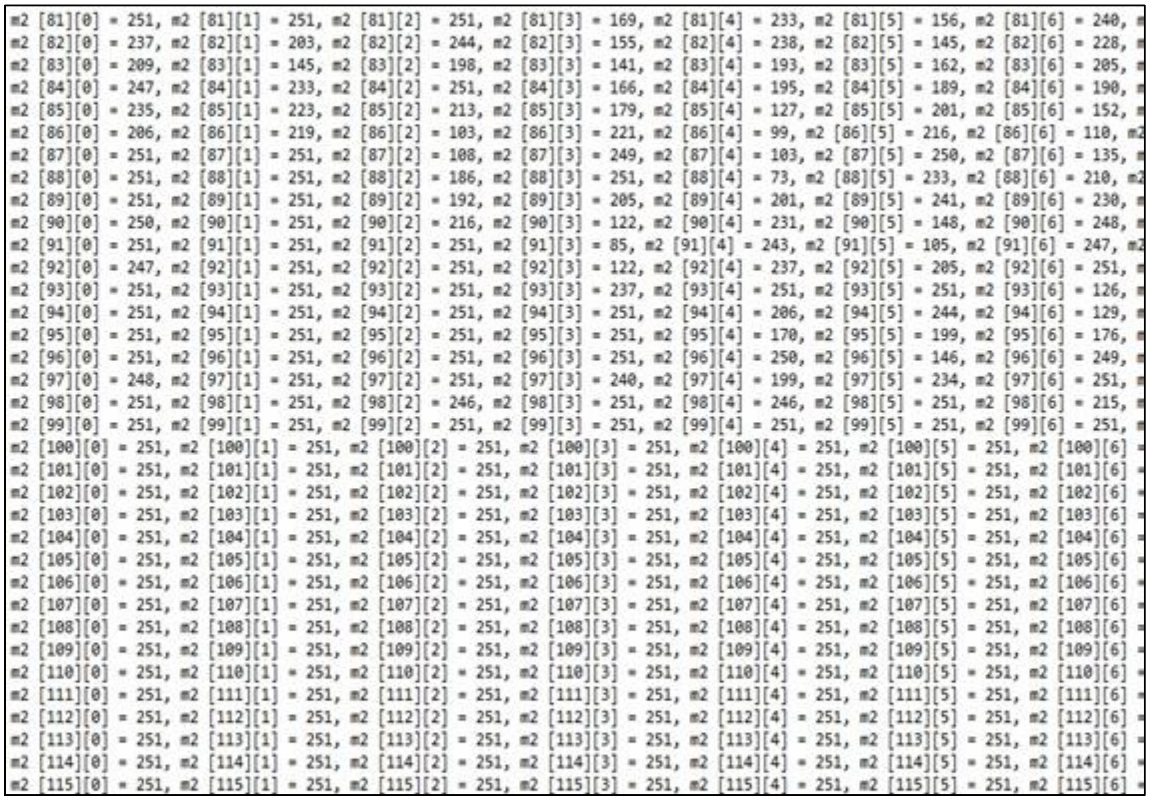
This stage is implemented in Java. By the end of this stage, data are ready for the obfuscation, so that they will not require any additional processes. Therefore, to simplify both the preprocessing and the obfuscating processes, this transformation has been implemented in Java. Note that the obfuscating process has also been implemented in Java.

The main objective of this transformation is to convert the previously saved double matrix to integer values ranging from 0 to 251. The purpose of converting the double matrix into an integer matrix of the same size and dimensions of the double matrix is that our approach obfuscates integer values. Consider Algorithm 5.2 for this transformation:

```
Algorithm 5.2 Convert Double Matrix to Integer Matrix
Input: text file contain double matrix D[1..m][1..n]
Output: integer matrix M[1..m][1..n]
Steps:
     1. Open text file.
     2. Row ← number of rows
     3. Column ← number of column.
     4. Create both double matrix d [1..row][1..column]
     5. Create integer matrix M[1..row][1..column].
     6. Load double matrix from text file into created double
        matrix d[1..row][1..column].
     7. For r ← 0 to row -1
     8.    For c ← 0 to column-1
     9.        M[r][c] =
     10.   End for
     11. End for
```

This algorithm is mainly dependent on the number of rows and columns of double matrix, where m is the number of row and n is the number of column of the double matrix. The time complexity of this algorithm is O(mn).

A screenshot has been taken as an example of transforming the double matrix into integers. However, only a segment of the resulting integer matrix is shown in Figure 5.2, because the actual one too large to be shown in one figure.



```
m2 [81][0] = 251, m2 [81][1] = 251, m2 [81][2] = 251, m2 [81][3] = 169, m2 [81][4] = 233, m2 [81][5] = 156, m2 [81][6] = 240, m
m2 [82][0] = 237, m2 [82][1] = 203, m2 [82][2] = 244, m2 [82][3] = 155, m2 [82][4] = 238, m2 [82][5] = 145, m2 [82][6] = 228, m
m2 [83][0] = 209, m2 [83][1] = 145, m2 [83][2] = 198, m2 [83][3] = 141, m2 [83][4] = 193, m2 [83][5] = 162, m2 [83][6] = 205, m
m2 [84][0] = 247, m2 [84][1] = 233, m2 [84][2] = 251, m2 [84][3] = 166, m2 [84][4] = 195, m2 [84][5] = 189, m2 [84][6] = 190, m
m2 [85][0] = 235, m2 [85][1] = 223, m2 [85][2] = 213, m2 [85][3] = 179, m2 [85][4] = 127, m2 [85][5] = 201, m2 [85][6] = 152, m
m2 [86][0] = 206, m2 [86][1] = 219, m2 [86][2] = 103, m2 [86][3] = 221, m2 [86][4] = 99, m2 [86][5] = 216, m2 [86][6] = 110, m2
m2 [87][0] = 251, m2 [87][1] = 251, m2 [87][2] = 108, m2 [87][3] = 249, m2 [87][4] = 103, m2 [87][5] = 250, m2 [87][6] = 135, m
m2 [88][0] = 251, m2 [88][1] = 251, m2 [88][2] = 186, m2 [88][3] = 251, m2 [88][4] = 73, m2 [88][5] = 233, m2 [88][6] = 210, m2
m2 [89][0] = 251, m2 [89][1] = 251, m2 [89][2] = 192, m2 [89][3] = 205, m2 [89][4] = 201, m2 [89][5] = 241, m2 [89][6] = 230, m
m2 [90][0] = 250, m2 [90][1] = 251, m2 [90][2] = 216, m2 [90][3] = 122, m2 [90][4] = 231, m2 [90][5] = 148, m2 [90][6] = 248, m
m2 [91][0] = 251, m2 [91][1] = 251, m2 [91][2] = 251, m2 [91][3] = 85, m2 [91][4] = 243, m2 [91][5] = 105, m2 [91][6] = 247, m2
m2 [92][0] = 247, m2 [92][1] = 251, m2 [92][2] = 251, m2 [92][3] = 122, m2 [92][4] = 237, m2 [92][5] = 205, m2 [92][6] = 251, m
m2 [93][0] = 251, m2 [93][1] = 251, m2 [93][2] = 251, m2 [93][3] = 237, m2 [93][4] = 251, m2 [93][5] = 251, m2 [93][6] = 126, m
m2 [94][0] = 251, m2 [94][1] = 251, m2 [94][2] = 251, m2 [94][3] = 251, m2 [94][4] = 206, m2 [94][5] = 244, m2 [94][6] = 129, m
m2 [95][0] = 251, m2 [95][1] = 251, m2 [95][2] = 251, m2 [95][3] = 251, m2 [95][4] = 170, m2 [95][5] = 199, m2 [95][6] = 176, m
m2 [96][0] = 251, m2 [96][1] = 251, m2 [96][2] = 251, m2 [96][3] = 251, m2 [96][4] = 250, m2 [96][5] = 146, m2 [96][6] = 249, m
m2 [97][0] = 248, m2 [97][1] = 251, m2 [97][2] = 251, m2 [97][3] = 240, m2 [97][4] = 199, m2 [97][5] = 234, m2 [97][6] = 251, m
m2 [98][0] = 251, m2 [98][1] = 251, m2 [98][2] = 246, m2 [98][3] = 251, m2 [98][4] = 246, m2 [98][5] = 251, m2 [98][6] = 215, m
m2 [99][0] = 251, m2 [99][1] = 251, m2 [99][2] = 251, m2 [99][3] = 251, m2 [99][4] = 251, m2 [99][5] = 251, m2 [99][6] = 251, m
m2 [100][0] = 251, m2 [100][1] = 251, m2 [100][2] = 251, m2 [100][3] = 251, m2 [100][4] = 251, m2 [100][5] = 251, m2 [100][6] =
m2 [101][0] = 251, m2 [101][1] = 251, m2 [101][2] = 251, m2 [101][3] = 251, m2 [101][4] = 251, m2 [101][5] = 251, m2 [101][6] =
m2 [102][0] = 251, m2 [102][1] = 251, m2 [102][2] = 251, m2 [102][3] = 251, m2 [102][4] = 251, m2 [102][5] = 251, m2 [102][6] =
m2 [103][0] = 251, m2 [103][1] = 251, m2 [103][2] = 251, m2 [103][3] = 251, m2 [103][4] = 251, m2 [103][5] = 251, m2 [103][6] =
m2 [104][0] = 251, m2 [104][1] = 251, m2 [104][2] = 251, m2 [104][3] = 251, m2 [104][4] = 251, m2 [104][5] = 251, m2 [104][6] =
m2 [105][0] = 251, m2 [105][1] = 251, m2 [105][2] = 251, m2 [105][3] = 251, m2 [105][4] = 251, m2 [105][5] = 251, m2 [105][6] =
m2 [106][0] = 251, m2 [106][1] = 251, m2 [106][2] = 251, m2 [106][3] = 251, m2 [106][4] = 251, m2 [106][5] = 251, m2 [106][6] =
m2 [107][0] = 251, m2 [107][1] = 251, m2 [107][2] = 251, m2 [107][3] = 251, m2 [107][4] = 251, m2 [107][5] = 251, m2 [107][6] =
m2 [108][0] = 251, m2 [108][1] = 251, m2 [108][2] = 251, m2 [108][3] = 251, m2 [108][4] = 251, m2 [108][5] = 251, m2 [108][6] =
m2 [109][0] = 251, m2 [109][1] = 251, m2 [109][2] = 251, m2 [109][3] = 251, m2 [109][4] = 251, m2 [109][5] = 251, m2 [109][6] =
m2 [110][0] = 251, m2 [110][1] = 251, m2 [110][2] = 251, m2 [110][3] = 251, m2 [110][4] = 251, m2 [110][5] = 251, m2 [110][6] =
m2 [111][0] = 251, m2 [111][1] = 251, m2 [111][2] = 251, m2 [111][3] = 251, m2 [111][4] = 251, m2 [111][5] = 251, m2 [111][6] =
m2 [112][0] = 251, m2 [112][1] = 251, m2 [112][2] = 251, m2 [112][3] = 251, m2 [112][4] = 251, m2 [112][5] = 251, m2 [112][6] =
m2 [113][0] = 251, m2 [113][1] = 251, m2 [113][2] = 251, m2 [113][3] = 251, m2 [113][4] = 251, m2 [113][5] = 251, m2 [113][6] =
m2 [114][0] = 251, m2 [114][1] = 251, m2 [114][2] = 251, m2 [114][3] = 251, m2 [114][4] = 251, m2 [114][5] = 251, m2 [114][6] =
m2 [115][0] = 251, m2 [115][1] = 251, m2 [115][2] = 251, m2 [115][3] = 251, m2 [115][4] = 251, m2 [115][5] = 251, m2 [115][6] =
```

Figure 5.2: A segment of the 2D integer matrix.

At the end of this transformation, the final resulting integer matrix is fed to the obfuscating process, which starts running automatically once it gets the data.

5.3. Obfuscating-Process Implementation

In this section, the algorithms and techniques for implementing the obfuscating process are presented and described in detail. The main purposes of this process are to obfuscate fingerprint images and send them to the server for comparison with the stored images with the same size. In order to simplify the implementing processes, the computing environment must optimize and compute the obfuscation stage efficiently, without consuming excessive amounts of the client's resources. This process has been implemented in Java. Our Java implementation is expected to save both time and space complexity.

This process obfuscates the fingerprint image that has been previously processed and represented as a two-dimensional integer matrix. Recall the proposed obfuscating process in Chapter 4, which consists of six different techniques, each of them has it specific tasks to perform. However, all of these six techniques have been fully implemented. Figure 5.3 shows the abstraction of six techniques into two bigger processes—namely, blocking stage and obfuscating stage.

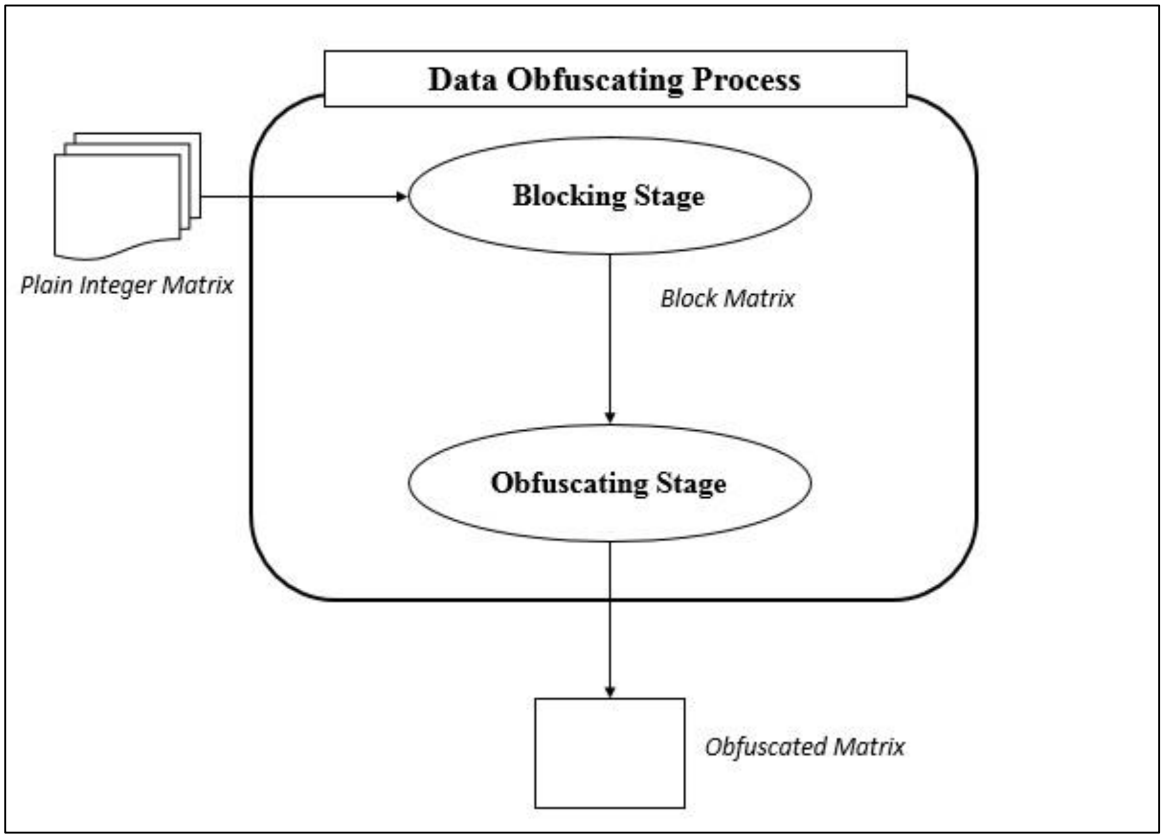


Figure 5.3: Data-Obfuscating Process.

Algorithm 5.3 depicts the main steps involved in the obfuscating process:

```
Algorithm 5.3 Obfuscating process
Input: integer matrix M [1..m][1..n].
Output: obfuscated matrix encBM[1..m][1..n/2], prepared ciphered
        matrix  DencBM[1..m][1..n/2] of double elements.
Steps:
     1. create block matrix BM[1..m][1..n/2].
     2. if there exist private key
     3.       then encrypt block matrix by the exist private
              key.
     4. else
     5.       create private key.
     6. obfuscate block matrix.
     7. save obfuscated matrix encBM[1..m][1..n/2].
     8. send both encrypted matrix to server.
```

This algorithm handles two techniques for obfuscating: *creating block matrix* and *obfuscate block matrix*. The following section describes these two, and shows how these work in details.

5.3.1. Blocking stage

The main aim of this stage is to create a blocking matrix from the integer matrix to minimize the encryption process. Each block in the block matrix has the size of two contents of the integer matrix that are concatenated horizontally. This technique reduces the number of columns to half in the block matrix; thus, the obfuscating process is required to use fewer numbers of private keys.

Another goal of this stage is to generate a digit counter that is a hashed digit matrix

(Chapter 4, Section 4.3.2, discusses the digit counter and hashed digit matrix). When each block is created, its digits counter is recorded, hashed, and saved into a hashed digit matrix. So in this stage, two components (blocks and the digit counter) are generated to save time complexity.

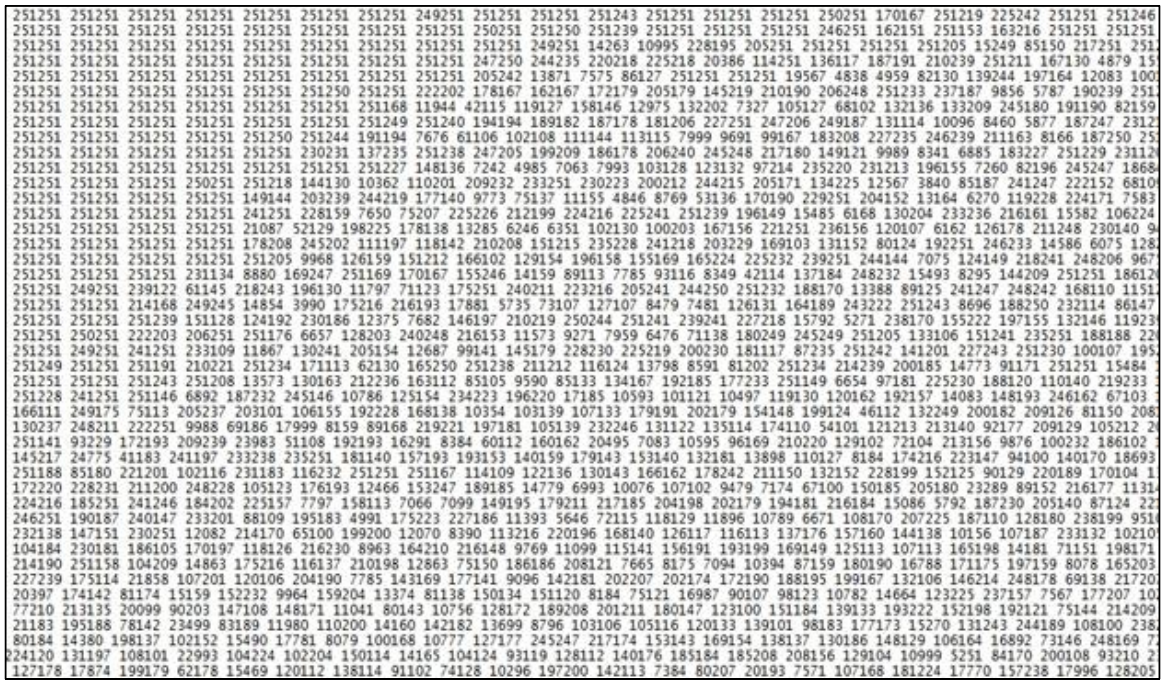
Consider Algorithm 5.4, which shows how blocks are generated using a digit counter and hashed digit matrix:

```
Algorithm 5.4 Blocking Stage
Input: integer matrix M[1..m][1..n]
Output: block matrix BM[1..m][1..n/2], and hashed digit matrix
       DHBM[1..m][1..n/2].
Steps:
      1. create integer matrix BM[1..m][1..n/2].
      2. create double matrix DHBM [1..m][1..n/2].
      3. c ← 0
      4. for i ← 0 to m-1
      5.    for j ← 0 to (n/2)-1
      6.       digit counter ← the number of the M[i][j*2]
               digits and concatenating it with the total number
               of digits of M[i][(j*2)+1].
      7.       DHBM[i][c] ← hash digit counter
      8.       BM[i][c]← concatenating both M[i][ j*2] with
               M[i][( j*2)+1].
      9.       increment c.
      10.   end for.
      11.   c ← 0.
      12.  end for.
```

This algorithm is mainly dependent on the number of rows and columns, where m is the number of row and n is the number of column of M matrix (the input integer matrix

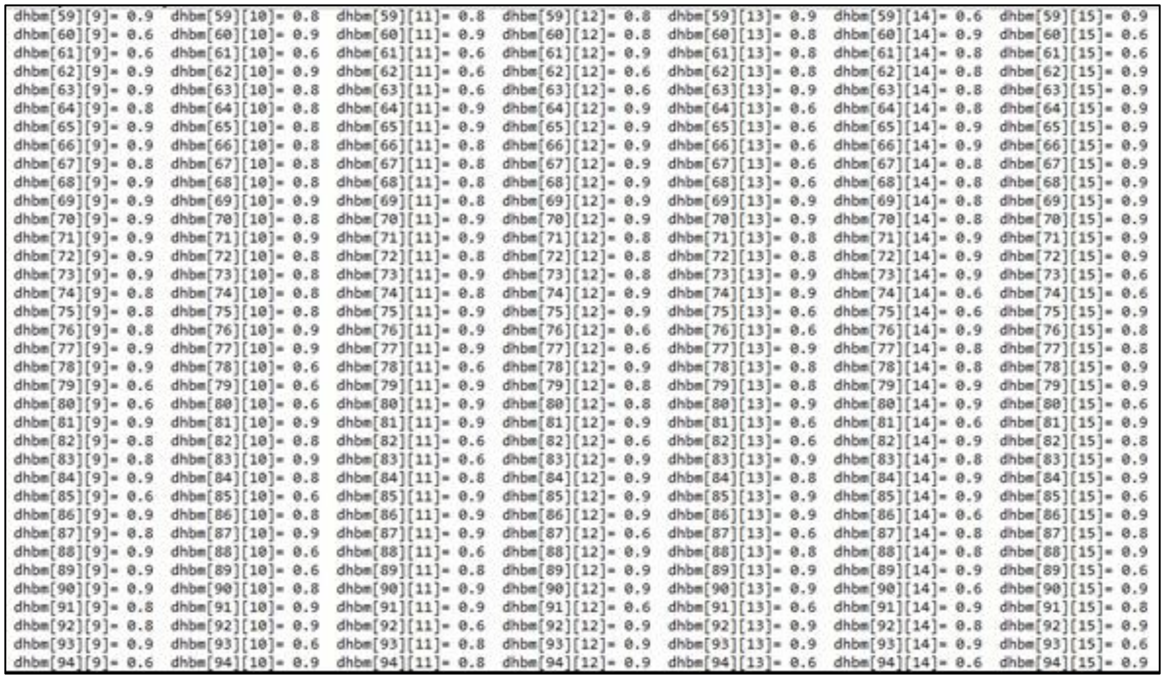
M), where M is the resulted integer matrix from the preprocessing. The complexity of this algorithm is O(m(n/2)).

A screenshot has been taken as an example of the blocking-stage result. However, only a segment of the BM (block) matrix is shown in Figure 5.4, because the actual one is too large to be shown in one figure. Only a part of the DHBM (hashed digit) matrix is shown in Figure 5.5.



```
251251 251251 251251 251251 251251 251251 251251 249251 251251 251251 251243 251251 251251 251251 250251 170167 251219 225242 251251 251246
251251 251251 251251 251251 251251 251251 251251 251251 250251 251250 251239 251251 251251 251251 246251 162151 251153 163216 251251 251251
251251 251251 251251 251251 251251 251251 251251 251251 251251 249251 14263 10995 228195 205251 251251 251251 251205 15249 85150 217251 251
251251 251251 251251 251251 251251 251251 251251 251251 247250 244235 220218 225218 20386 114251 136117 187191 210239 251211 167130 4879 15
251251 251251 251251 251251 251251 251251 251251 251251 205242 13871 7575 86127 251251 251251 19567 4838 4959 82130 139244 197164 12083 100
251251 251251 251251 251251 251251 251250 251251 222202 178167 162167 172179 205179 145219 210190 206248 251233 237187 9856 5787 190239 251
251251 251251 251251 251251 251251 251251 251168 11944 42115 119127 158146 12975 132202 7327 105127 68102 132136 133209 245180 191190 82159
251251 251251 251251 251251 251251 251251 251249 251240 194194 189182 187178 181206 227251 247206 249187 131114 10096 8460 5877 187247 2312
251251 251251 251251 251251 251250 251244 191194 7676 61106 102108 111144 113115 7999 9691 99167 183208 227235 246239 211163 8166 187250 25
251251 251251 251251 251251 251251 230231 137235 251238 247205 199209 186178 206240 245248 217180 149121 9989 8341 6885 183227 251229 23112
251251 251251 251251 251251 251251 251251 251227 148136 7242 4985 7063 7993 103128 123132 97214 235220 231213 196155 7260 82196 245247 1868
251251 251251 251251 250251 251218 144130 10362 110201 209232 233251 230223 200212 244215 205171 134225 12567 3840 85187 241247 222152 6810
251251 251251 251251 251251 149144 203239 244219 177140 9773 75137 11155 4846 8769 53136 170190 229251 204152 13164 6270 119228 224171 7583
251251 251251 251251 251251 241251 228159 7650 75207 225226 212199 224216 225241 251239 196149 15485 6168 130204 233236 216161 15582 106224
251251 251251 251251 251251 21087 52129 198225 178138 13285 6246 6351 102130 100203 167156 221251 236156 120107 6162 126178 211248 230140 9
251251 251251 251251 251251 178208 245202 111197 118142 210208 151215 235228 241218 203229 169103 131152 80124 192251 246233 14586 6075 128
251251 251251 251251 251251 251205 9968 126159 151212 166102 129154 196158 155169 165224 225232 239251 244144 7075 124149 218241 248206 967
251251 251251 251251 231134 8880 169247 251169 170167 155246 14159 89113 7785 93116 8349 42114 137184 248232 15493 8295 144209 251251 18612
251251 249251 239122 61145 218243 196130 11797 71123 175251 240211 223216 205241 244250 251232 188170 13388 89125 241247 248242 168110 1151
251251 251251 214168 249245 14854 3990 175216 216193 17881 5735 73107 127107 8479 7481 126131 164189 243222 251243 8696 188250 232114 86147
251251 251251 251239 151128 124192 230186 12375 7682 146197 210219 250244 251241 239241 227218 15792 5271 238170 155222 197155 132146 11923
251251 250251 222203 206251 251176 6657 128203 240248 216153 11573 9271 7959 6476 71138 180249 245249 251205 133106 151241 235251 188188 22
251251 249251 241251 233109 11867 130241 205154 12687 99141 145179 228230 225219 200230 181117 87235 251242 141201 227243 251230 100107 195
251249 251251 251191 210221 251234 171113 62130 165250 251238 211212 116124 13798 8591 81202 251234 214239 200185 14773 91171 251251 15484
251251 251251 251243 251208 13573 130163 212236 163112 85105 9590 85133 134167 192185 177233 251149 6654 97181 225230 188120 110140 219233
251228 241251 251146 6892 187232 245146 10786 125154 234223 196220 17185 10593 101121 10497 119130 120162 192157 14083 148193 246162 67103
166111 249175 75113 205237 203101 106155 192228 168138 10354 103139 107133 179191 202179 154148 199124 46112 132249 200182 209126 81150 208
130237 248211 222251 9988 69186 17999 8159 89168 219221 197181 105139 232246 131122 135114 174110 54101 121213 213140 92177 209129 105212 2
251141 93229 172193 209239 23983 51108 192193 16291 8384 60112 160162 20495 7083 10595 96169 210220 129102 72104 213156 9876 100232 186102
145217 24775 41183 241197 233238 235251 181140 157193 193153 140159 179143 153140 132181 13898 110127 8184 174216 223147 94100 140170 18693
251188 85180 221201 102116 231183 116232 251251 251167 114109 122136 130143 166162 178242 211150 132152 228199 152125 90129 220189 170104 1
172220 228231 211200 248228 105123 176193 12466 153247 189185 14779 6993 10076 107102 9479 7174 67100 150185 205180 23289 89152 216177 1131
224216 185251 241246 184202 225157 7797 158113 7066 7099 149195 179211 217185 204198 202179 194181 216184 15086 5792 187230 205140 87124 22
246251 190187 240147 233201 88109 195183 4991 175223 227186 11393 5646 72115 118129 11896 10789 6671 108170 207225 187110 128180 238199 951
232138 147151 230251 12082 214170 65100 199200 12070 8390 113216 220196 168140 126117 116113 137176 157160 144138 10156 107187 233132 10210
104184 230181 186105 170197 118126 216230 8963 164210 216148 9769 11099 115141 156191 193199 169149 125113 107113 165198 14181 71151 198171
214190 251158 104209 14863 175216 116137 210198 12863 75150 186186 208121 7665 8175 7094 10394 87159 180190 16788 171175 197159 8078 165203
227239 175114 21858 107201 120106 204190 7785 143169 177141 9096 142181 202207 202174 172190 188195 199167 132106 146214 248178 69138 21720
20397 174142 81174 15159 152232 9964 159204 13374 81138 150134 151120 8184 75121 16987 90107 98123 10782 14664 123225 237157 7567 177207 10
77210 213135 20099 90203 147108 148171 11041 80143 10756 128172 189208 201211 180147 123100 151184 139133 193222 152198 192121 75144 214209
21183 195188 78142 23499 83189 11980 110200 14160 142182 13699 8796 103106 105116 120133 139101 98183 177173 15270 131243 244189 108100 238
80184 14380 198137 102152 15490 17781 8079 100168 10777 127177 245247 217174 153143 169154 138137 130186 148129 106164 16892 73146 248169 7
224120 131197 108101 22993 104224 102204 150114 14165 104124 93119 128112 140176 185184 185208 208156 129104 10999 5251 84170 200108 93210 2
127178 17874 199179 62178 15469 120112 138114 91102 74128 10296 197200 142113 7384 80207 20193 7571 107168 181224 17770 157238 17996 128205
```

Figure 5.4: A segment of the block matrix (BM)



```
dhbm[59][9]= 0.9  dhbm[59][10]= 0.8  dhbm[59][11]= 0.8  dhbm[59][12]= 0.8  dhbm[59][13]= 0.9  dhbm[59][14]= 0.6  dhbm[59][15]= 0.9
dhbm[60][9]= 0.6  dhbm[60][10]= 0.9  dhbm[60][11]= 0.9  dhbm[60][12]= 0.8  dhbm[60][13]= 0.8  dhbm[60][14]= 0.9  dhbm[60][15]= 0.6
dhbm[61][9]= 0.6  dhbm[61][10]= 0.6  dhbm[61][11]= 0.6  dhbm[61][12]= 0.9  dhbm[61][13]= 0.8  dhbm[61][14]= 0.8  dhbm[61][15]= 0.6
dhbm[62][9]= 0.9  dhbm[62][10]= 0.9  dhbm[62][11]= 0.6  dhbm[62][12]= 0.6  dhbm[62][13]= 0.8  dhbm[62][14]= 0.8  dhbm[62][15]= 0.9
dhbm[63][9]= 0.9  dhbm[63][10]= 0.8  dhbm[63][11]= 0.6  dhbm[63][12]= 0.6  dhbm[63][13]= 0.9  dhbm[63][14]= 0.8  dhbm[63][15]= 0.9
dhbm[64][9]= 0.8  dhbm[64][10]= 0.8  dhbm[64][11]= 0.9  dhbm[64][12]= 0.9  dhbm[64][13]= 0.6  dhbm[64][14]= 0.8  dhbm[64][15]= 0.9
dhbm[65][9]= 0.9  dhbm[65][10]= 0.8  dhbm[65][11]= 0.9  dhbm[65][12]= 0.9  dhbm[65][13]= 0.6  dhbm[65][14]= 0.9  dhbm[65][15]= 0.9
dhbm[66][9]= 0.9  dhbm[66][10]= 0.8  dhbm[66][11]= 0.8  dhbm[66][12]= 0.9  dhbm[66][13]= 0.6  dhbm[66][14]= 0.9  dhbm[66][15]= 0.9
dhbm[67][9]= 0.8  dhbm[67][10]= 0.8  dhbm[67][11]= 0.8  dhbm[67][12]= 0.9  dhbm[67][13]= 0.6  dhbm[67][14]= 0.8  dhbm[67][15]= 0.9
dhbm[68][9]= 0.9  dhbm[68][10]= 0.8  dhbm[68][11]= 0.8  dhbm[68][12]= 0.9  dhbm[68][13]= 0.6  dhbm[68][14]= 0.8  dhbm[68][15]= 0.9
dhbm[69][9]= 0.9  dhbm[69][10]= 0.9  dhbm[69][11]= 0.8  dhbm[69][12]= 0.9  dhbm[69][13]= 0.9  dhbm[69][14]= 0.8  dhbm[69][15]= 0.9
dhbm[70][9]= 0.9  dhbm[70][10]= 0.8  dhbm[70][11]= 0.9  dhbm[70][12]= 0.9  dhbm[70][13]= 0.9  dhbm[70][14]= 0.8  dhbm[70][15]= 0.9
dhbm[71][9]= 0.9  dhbm[71][10]= 0.9  dhbm[71][11]= 0.9  dhbm[71][12]= 0.8  dhbm[71][13]= 0.8  dhbm[71][14]= 0.9  dhbm[71][15]= 0.9
dhbm[72][9]= 0.9  dhbm[72][10]= 0.8  dhbm[72][11]= 0.8  dhbm[72][12]= 0.8  dhbm[72][13]= 0.8  dhbm[72][14]= 0.9  dhbm[72][15]= 0.9
dhbm[73][9]= 0.9  dhbm[73][10]= 0.8  dhbm[73][11]= 0.9  dhbm[73][12]= 0.8  dhbm[73][13]= 0.9  dhbm[73][14]= 0.9  dhbm[73][15]= 0.6
dhbm[74][9]= 0.8  dhbm[74][10]= 0.8  dhbm[74][11]= 0.8  dhbm[74][12]= 0.9  dhbm[74][13]= 0.9  dhbm[74][14]= 0.6  dhbm[74][15]= 0.6
dhbm[75][9]= 0.8  dhbm[75][10]= 0.8  dhbm[75][11]= 0.9  dhbm[75][12]= 0.9  dhbm[75][13]= 0.6  dhbm[75][14]= 0.6  dhbm[75][15]= 0.9
dhbm[76][9]= 0.8  dhbm[76][10]= 0.9  dhbm[76][11]= 0.9  dhbm[76][12]= 0.6  dhbm[76][13]= 0.6  dhbm[76][14]= 0.9  dhbm[76][15]= 0.8
dhbm[77][9]= 0.9  dhbm[77][10]= 0.9  dhbm[77][11]= 0.9  dhbm[77][12]= 0.6  dhbm[77][13]= 0.9  dhbm[77][14]= 0.8  dhbm[77][15]= 0.8
dhbm[78][9]= 0.9  dhbm[78][10]= 0.6  dhbm[78][11]= 0.6  dhbm[78][12]= 0.9  dhbm[78][13]= 0.8  dhbm[78][14]= 0.8  dhbm[78][15]= 0.9
dhbm[79][9]= 0.6  dhbm[79][10]= 0.6  dhbm[79][11]= 0.9  dhbm[79][12]= 0.8  dhbm[79][13]= 0.8  dhbm[79][14]= 0.9  dhbm[79][15]= 0.9
dhbm[80][9]= 0.6  dhbm[80][10]= 0.6  dhbm[80][11]= 0.9  dhbm[80][12]= 0.8  dhbm[80][13]= 0.9  dhbm[80][14]= 0.9  dhbm[80][15]= 0.6
dhbm[81][9]= 0.9  dhbm[81][10]= 0.9  dhbm[81][11]= 0.9  dhbm[81][12]= 0.9  dhbm[81][13]= 0.6  dhbm[81][14]= 0.6  dhbm[81][15]= 0.9
dhbm[82][9]= 0.8  dhbm[82][10]= 0.8  dhbm[82][11]= 0.6  dhbm[82][12]= 0.6  dhbm[82][13]= 0.6  dhbm[82][14]= 0.9  dhbm[82][15]= 0.8
dhbm[83][9]= 0.8  dhbm[83][10]= 0.9  dhbm[83][11]= 0.6  dhbm[83][12]= 0.9  dhbm[83][13]= 0.9  dhbm[83][14]= 0.8  dhbm[83][15]= 0.9
dhbm[84][9]= 0.9  dhbm[84][10]= 0.8  dhbm[84][11]= 0.8  dhbm[84][12]= 0.9  dhbm[84][13]= 0.8  dhbm[84][14]= 0.9  dhbm[84][15]= 0.9
dhbm[85][9]= 0.6  dhbm[85][10]= 0.6  dhbm[85][11]= 0.9  dhbm[85][12]= 0.9  dhbm[85][13]= 0.9  dhbm[85][14]= 0.9  dhbm[85][15]= 0.6
dhbm[86][9]= 0.9  dhbm[86][10]= 0.8  dhbm[86][11]= 0.9  dhbm[86][12]= 0.9  dhbm[86][13]= 0.9  dhbm[86][14]= 0.6  dhbm[86][15]= 0.9
dhbm[87][9]= 0.8  dhbm[87][10]= 0.9  dhbm[87][11]= 0.9  dhbm[87][12]= 0.6  dhbm[87][13]= 0.6  dhbm[87][14]= 0.8  dhbm[87][15]= 0.8
dhbm[88][9]= 0.9  dhbm[88][10]= 0.6  dhbm[88][11]= 0.6  dhbm[88][12]= 0.9  dhbm[88][13]= 0.8  dhbm[88][14]= 0.8  dhbm[88][15]= 0.9
dhbm[89][9]= 0.9  dhbm[89][10]= 0.6  dhbm[89][11]= 0.8  dhbm[89][12]= 0.9  dhbm[89][13]= 0.9  dhbm[89][14]= 0.9  dhbm[89][15]= 0.6
dhbm[90][9]= 0.9  dhbm[90][10]= 0.8  dhbm[90][11]= 0.9  dhbm[90][12]= 0.9  dhbm[90][13]= 0.9  dhbm[90][14]= 0.6  dhbm[90][15]= 0.9
dhbm[91][9]= 0.8  dhbm[91][10]= 0.9  dhbm[91][11]= 0.9  dhbm[91][12]= 0.6  dhbm[91][13]= 0.6  dhbm[91][14]= 0.9  dhbm[91][15]= 0.8
dhbm[92][9]= 0.8  dhbm[92][10]= 0.9  dhbm[92][11]= 0.6  dhbm[92][12]= 0.9  dhbm[92][13]= 0.9  dhbm[92][14]= 0.8  dhbm[92][15]= 0.9
dhbm[93][9]= 0.9  dhbm[93][10]= 0.6  dhbm[93][11]= 0.8  dhbm[93][12]= 0.9  dhbm[93][13]= 0.9  dhbm[93][14]= 0.9  dhbm[93][15]= 0.6
dhbm[94][9]= 0.6  dhbm[94][10]= 0.9  dhbm[94][11]= 0.8  dhbm[94][12]= 0.9  dhbm[94][13]= 0.6  dhbm[94][14]= 0.6  dhbm[94][15]= 0.9
```

Figure 5.5: A segment of the DHBM matrix.

5.3.2. Obfuscating stage

This is the main stage of the obfuscating process. This stage uses four techniques of the obfuscating process—tokenization, generating sets, creating a private key, and obfuscating. The new private key is not created unless one does not already exist. Similarly, no new sets are created unless needed. Note that creating a private key is related to creating sets. If one is to be generated, then the other one needs to be created as well. If there exists no private key, new sets and the corresponding private keys need are created. Note that all fingerprints of a client are obfuscated using the same set of private keys. Recall that a private key consists of an array of private keys because a matrix is divided into various sizes of sets and each set has its own private key. Consider Algorithm 5.5, which shows how obfuscating is done when no private key exists:

Algorithm 5.5 Obfuscating Stage Case A

Input: block matrix BM[1..m][1..n], and hashed digit matrix DHBM[1..m][1..n].

Output: obfuscated matrix encBM[1..m][1..n] of obfuscated objects, and private key PK[1..x] of private key objects.

Steps:

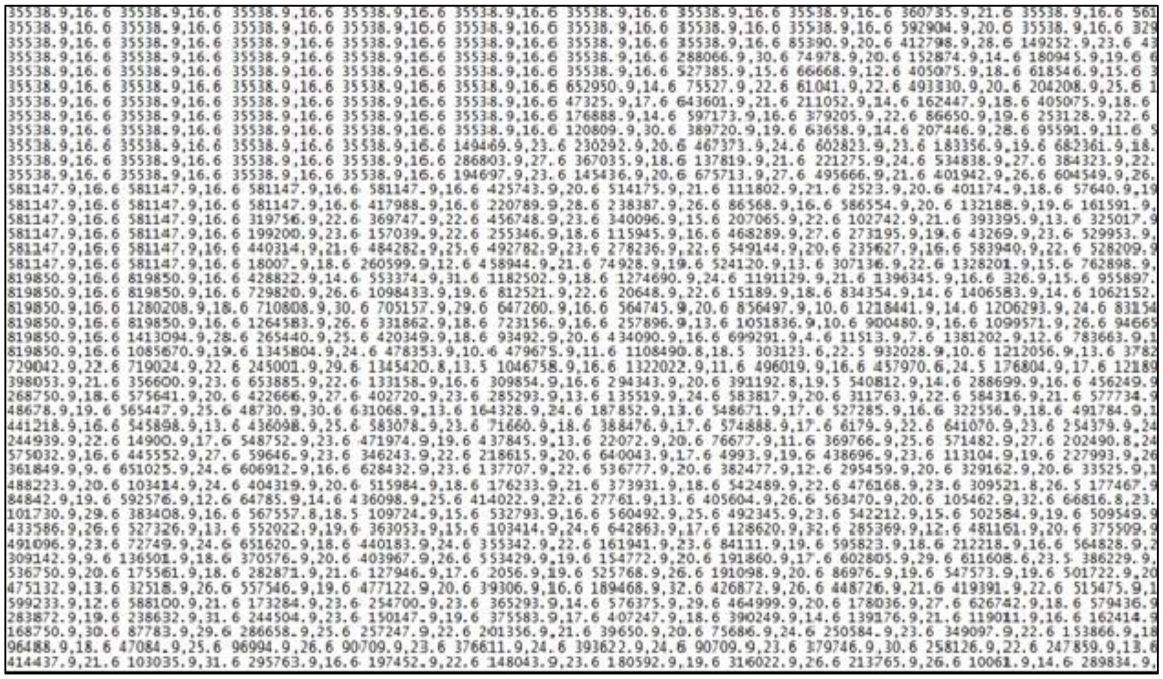
```
1. Generate sets by creating sets array sets[1..x].
2. Create encBM[1..m][1..n] matrix of obfuscated object.
3. Create private key PK[1..x] array of private key object.
4. Counter ← 0.
5. Sets counter ← 0.
6. For i ←0 to m
7.      For j ← to n
8.         if counter = 0 and sets counter = 0 then
9.                 Counter = sets[0].
10.                Generate p & q.
11.                PK[sets counter] ← new private key
                   object has p, q and sets[sets counter].
12.        End if
13.        token ← generate token for BM[i][j].
14.           enc ← encrypt BM[i][j]
15.           denc ← enc + DHBM[i][j]
16.           encBM[i][j] ← new obfuscated object has denc
              and token.
17.           decrement counter
18.           if counter = 0 then
19.                increment sets counter
20.                counter ← sets [sets counter]
21.                generate new p & q.
22.                PK[sets counter] ← new private key
                   object has p, q and sets[sets counter].
```

```
            end if
        end for
    end for
```

This algorithm depends mainly on the number of rows and columns of the BM matrix; it also depends on the length of the generated set array. Thus if m is the number of rows and n is the number of columns and x is the lengths of the generated sets array, then time complexity of this algorithm is O(x) + O(mn).

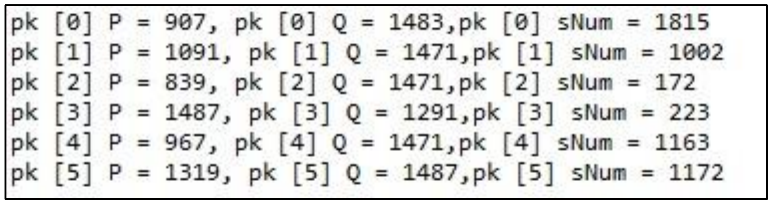
This algorithm implements four techniques—generating sets, creating a private key, tokenization, and obfuscating. Executing all of these techniques together saves space complexity and time complexity. Consider that required time complexity for the tokenization technique alone is O(mn).The time complexity for the technique of creating a private key array alone requires O(x). Thus, the total amount of saved time complexity is O(mn) + O(x), where m is the row of the BM matrix, n is the column of BM, and x is the length of the sets array. The total amount of generating space complexity is O(mn), because the size of token matrix is equal to the size of the BM matrix.

A screenshot has been taken as an example of the obfuscating stage. However, only part of the encBM matrix is shown in Figure 5.6, because the actual one is too large to be shown in one figure. Figure 5.7 shows the created private key.



```
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 360735.9,21.6 35538.9,16.6 561
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 592904.9,20.6 35538.9,16.6 329
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 85390.9,20.6 412798.9,28.6 149252.9,23.6 43
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 288066.9,30.6 74978.9,20.6 152874.9,14.6 180945.9,19.6 6
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 527385.9,15.6 66668.9,12.6 405075.9,18.6 618546.9,15.6 3
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 652950.9,14.6 75527.9,22.6 61041.9,22.6 493330.9,20.6 204208.9,25.6 1
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 47325.9,17.6 643601.9,21.6 211052.9,14.6 162447.9,18.6 405075.9,18.6
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 176888.9,14.6 597173.9,16.6 379205.9,22.6 86650.9,19.6 253128.9,22.6
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 120809.9,30.6 389720.9,19.6 63658.9,14.6 207446.9,28.6 95591.9,11.6 9
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 149469.9,23.6 230292.9,20.6 467373.9,24.6 602823.9,23.6 183356.9,19.6 682361.9,18.
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 286803.9,27.6 367035.9,18.6 137819.9,21.6 221275.9,24.6 534838.9,27.6 384323.9,22.
35538.9,16.6 35538.9,16.6 35538.9,16.6 35538.9,16.6 194697.9,23.6 145436.9,20.6 675713.9,27.6 495666.9,21.6 401942.9,26.6 604549.9,26.
581147.9,16.6 581147.9,16.6 581147.9,16.6 581147.9,16.6 425743.9,20.6 514175.9,21.6 111802.9,21.6 2523.9,20.6 401174.9,18.6 57640.9,19
581147.9,16.6 581147.9,16.6 581147.9,16.6 417988.9,16.6 220789.9,28.6 238387.9,26.6 86568.9,16.6 586554.9,20.6 132188.9,19.6 161591.9,
581147.9,16.6 581147.9,16.6 319756.9,22.6 369747.9,22.6 456748.9,23.6 340096.9,15.6 207065.9,22.6 102742.9,21.6 393395.9,13.6 325017.9
581147.9,16.6 581147.9,16.6 199200.9,23.6 157039.9,22.6 255346.9,18.6 115945.9,16.6 468289.9,27.6 273195.9,19.6 43269.9,23.6 529953.9,
581147.9,16.6 581147.9,16.6 440314.9,21.6 484282.9,25.6 492782.9,23.6 278236.9,22.6 549144.9,20.6 235627.9,16.6 583940.9,22.6 528209.9
581147.9,16.6 581147.9,16.6 18007.9,18.6 260599.9,12.6 458944.9,21.6 74928.9,19.6 524120.9,13.6 307136.9,22.6 1328201.9,15.6 762898.9,
819850.9,16.6 819850.9,16.6 428822.9,14.6 553374.9,31.6 1182502.9,18.6 1274690.9,24.6 1191129.9,21.6 1396345.9,16.6 326.9,15.6 955897.
819850.9,16.6 819850.9,16.6 729820.9,26.6 1098433.9,19.6 812521.9,22.6 20648.9,22.6 15189.9,18.6 834354.9,14.6 1406583.9,14.6 1062152.
819850.9,16.6 1280208.9,18.6 710808.9,30.6 705157.9,29.6 647260.9,16.6 564745.9,20.6 856497.9,10.6 1218441.9,14.6 1206293.9,24.6 83154
819850.9,16.6 819850.9,16.6 1264583.9,26.6 331862.9,18.6 723156.9,16.6 257896.9,13.6 1051836.9,10.6 900480.9,16.6 1099571.9,26.6 94665
819850.9,16.6 1413094.9,28.6 265440.9,25.6 420349.9,18.6 93492.9,20.6 434090.9,16.6 699291.9,4.6 11513.9,7.6 1381202.9,12.6 783663.9,1
819850.9,16.6 1085670.9,19.6 1345804.9,24.6 478353.9,10.6 479675.9,11.6 1108490.8,18.5 303123.6,22.5 932028.9,10.6 1212056.9,13.6 3782
729042.9,22.6 719024.9,22.6 245001.9,29.6 1345420.8,13.5 1046758.9,16.6 1322022.9,11.6 496019.9,16.6 457970.6,24.5 176804.9,17.6 12189
398053.9,21.6 356600.9,23.6 653885.9,22.6 133158.9,16.6 309854.9,16.6 294343.9,20.6 391192.8,19.5 540812.9,14.6 288699.9,16.6 456249.9
268750.9,18.6 575641.9,20.6 422666.9,27.6 402720.9,23.6 285293.9,13.6 135519.9,24.6 583817.9,20.6 311763.9,22.6 584316.9,21.6 577734.9
48678.9,19.6 565447.9,25.6 48730.9,30.6 631068.9,13.6 164328.9,24.6 187852.9,13.6 548671.9,17.6 527285.9,16.6 322556.9,18.6 491784.9,1
441218.9,16.6 545898.9,13.6 436098.9,25.6 583078.9,23.6 71660.9,18.6 388476.9,17.6 574888.9,17.6 6179.9,22.6 641070.9,23.6 254379.9,24
244939.9,22.6 14900.9,17.6 548752.9,23.6 471974.9,19.6 437845.9,13.6 22072.9,20.6 76677.9,11.6 369766.9,25.6 571482.9,27.6 202490.8,24
575032.9,16.6 445552.9,27.6 59646.9,23.6 346243.9,22.6 218615.9,20.6 640043.9,17.6 4993.9,19.6 438696.9,23.6 113104.9,19.6 227993.9,26
361849.9,9.6 651025.9,24.6 606912.9,16.6 628432.9,23.6 137707.9,22.6 536777.9,20.6 382477.9,12.6 295459.9,20.6 329162.9,20.6 33525.9,1
488223.9,20.6 103414.9,24.6 404319.9,20.6 515984.9,18.6 176233.9,21.6 373931.9,18.6 542489.9,22.6 476168.9,23.6 309521.8,26.5 177467.9
84842.9,19.6 592576.9,12.6 64785.9,14.6 436098.9,25.6 414022.9,22.6 27761.9,13.6 405604.9,26.6 563470.9,20.6 105462.9,32.6 66816.8,23.
101730.9,29.6 383408.9,16.6 567557.8,18.5 109724.9,15.6 532793.9,16.6 560492.9,25.6 492345.9,23.6 542212.9,15.6 502584.9,19.6 509549.9
433586.9,26.6 527326.9,13.6 552022.9,19.6 363053.9,15.6 103414.9,24.6 642863.9,17.6 128620.9,32.6 285369.9,12.6 481161.9,20.6 375509.9
491096.9,23.6 72749.9,24.6 651620.9,18.6 440183.9,24.6 355342.9,22.6 161941.9,23.6 84111.9,19.6 595823.9,18.6 212218.9,16.6 564828.9,2
309142.9,9.6 136501.9,18.6 370576.9,20.6 403967.9,26.6 553429.9,19.6 154772.9,20.6 191860.9,17.6 602805.9,29.6 611608.6,23.5 386229.9,
536750.9,20.6 175561.9,18.6 282871.9,21.6 127946.9,17.6 2056.9,19.6 525768.9,26.6 191098.9,20.6 86976.9,19.6 547573.9,19.6 501722.9,20
475132.9,13.6 32518.9,26.6 557546.9,19.6 477122.9,20.6 39306.9,16.6 189468.9,32.6 426872.9,26.6 448726.9,21.6 419391.9,22.6 515475.9,1
599233.9,12.6 588100.9,21.6 173284.9,23.6 254700.9,23.6 365293.9,14.6 576375.9,29.6 464999.9,20.6 178036.9,27.6 626742.9,18.6 579436.9
283872.9,19.6 238632.9,31.6 244504.9,23.6 150147.9,19.6 375583.9,17.6 407247.9,18.6 390249.9,14.6 139176.9,21.6 119011.9,16.6 162414.9
168750.9,30.6 87783.9,29.6 286658.9,25.6 257247.9,22.6 201356.9,21.6 39650.9,20.6 75686.9,24.6 250584.9,23.6 349097.9,22.6 153866.9,18
96488.9,18.6 47084.9,25.6 96994.9,26.6 90709.9,23.6 376611.9,24.6 393622.9,24.6 90709.9,23.6 379746.9,30.6 258126.9,22.6 247859.9,13.6
414437.9,21.6 103035.9,31.6 295763.9,16.6 197452.9,22.6 148043.9,23.6 180592.9,19.6 316022.9,26.6 213765.9,26.6 10061.9,14.6 289834.9,
```

Figure 5.6: A segment of the 2D encBM matrix



```
pk [0] P = 907, pk [0] Q = 1483,pk [0] sNum = 1815
pk [1] P = 1091, pk [1] Q = 1471,pk [1] sNum = 1002
pk [2] P = 839, pk [2] Q = 1471,pk [2] sNum = 172
pk [3] P = 1487, pk [3] Q = 1291,pk [3] sNum = 223
pk [4] P = 967, pk [4] Q = 1471,pk [4] sNum = 1163
pk [5] P = 1319, pk [5] Q = 1487,pk [5] sNum = 1172
```

Figure 5.7: Private key array

The created private key array in Figure 5.7 has a length of six, which is used to encrypt the block matrix BM[129][84] of size 129 * 84. The generating-sets process places 1815 blocks of BM matrix in the first set, 1002 blocks in the second set, 172 blocks in the third set, 223 blocks in the fourth set, 1163 blocks in the fifth set, and 1172 blocks in the final set. Each set of blocks has been encrypted by its own prime factors p and q.

Consider Algorithm 5.6, which shows how the private key is generated, if there exists a private key:

```
Algorithm 5.6 Obfuscating Stage Case B
Input: block matrix BM[1..m][1..n], and hashed digit matrix
DHBM[1..m][1..n].
Output: obfuscated matrix encBM[1..m][1..n] of obfuscated
object.
Steps:
    1. Create encBM[1..m][1..n] matrix of obfuscate object.
    2. ns ← set size content of PK[0].
    3. ipk ← 0.
    4. for i ←0 to m
    5.     for j ← to n
    6.        token ← generate token for BM[i][j].
    7.        enc ← encrypt BM[i][j]
    8.        denc ← enc + DHBM[i][j]
    9.        encBM[i][j] ← new obfuscated object has denc
       and token.
    10.          decrement ns
    11.          if ns = 0 then
    12.               increment ipk
    13.               ns ← sets contents of PK[ipk]
    14.               p ← p content of PK[ipk]
    15.               q ← q content of PK[ipk]
    16.         end if
    17.      end for
    18.  end for
```

This algorithm depends mainly on the number of rows and columns of the BM matrix. Thus, if m is the number of rows and n is the number of columns, then the time complexity of this algorithm is O(mn).

This algorithm implements two techniques: tokenization and encrypting. By implementing these techniques together, we save space complexity and time complexity of the tokenization technique alone, which requires O(mn). Then the total amount of saving both space and time complexity is O(mn), where m is the row of the BM matrix, and n is the column of BM, because the size of the token matrix is equal to the size of the BM matrix.

5.4. Comparison-process implementation

In this section, the algorithms and techniques for implementing the comparison of two images is presented and described in detail. We need this operation on the server side, in which the comparison function is installed. Its main purpose is to compare the obfuscated fingerprint images with the stored obfuscated ones. The server sends the result of comparison to the client. This comparison process has been implemented with Matlab. The main comparison function is a Matlab method for comparing images. One of the Matlab methods does the comparisons by drawing a histogram for each image and comparing between these histograms.

5.4.1. Comparison Stage

The main goal of this stage is to compare the obfuscated fingerprint with the stored ones to check if the fingerprint already exists or not.

Consider Algorithm 5.7, which shows how the comparison is done.

```
Algorithm 5.7 Comparing Stage
  Input: encrypted matrix encBM[1..m][1..n] of obfuscated
objects.
Output: “Exist” or “Not Exist”
Steps:
      1. hn1 ← draw histogram for encMatrix
      2. c ← 0
      3. for i ← 1 to total number of obfuscated stored
         fingerprints
      4.       hn2 ← draw histogram for encrypted fingerprint [i]
      5.       f ← result of comparing between hn1 and hn2
      6.       if f < = 0.000009, then increment c
      7. end for
      8. if c > 1,
      9.    then send “Exist”
      10.   else send “Not Exist”
      11.   end if
```

This algorithm depends mainly on the total number of stored obfuscated fingerprints in the server. Thus, if T is the total number of stored obfuscated fingerprints, the time complexity of this algorithm is O(T(mn)), where m is the row of the encBM matrix, and n is the column of encBM.

An example of two cases of the comparison process is shown to demonstrate its validity. In the first case, the client sends an obfuscated fingerprint matrix to the server, which compares it with the obfuscated images already stored in the server. Consider the actual two fingerprints—fingerprint1 and fingerprint2—shown in Figure 5.8.

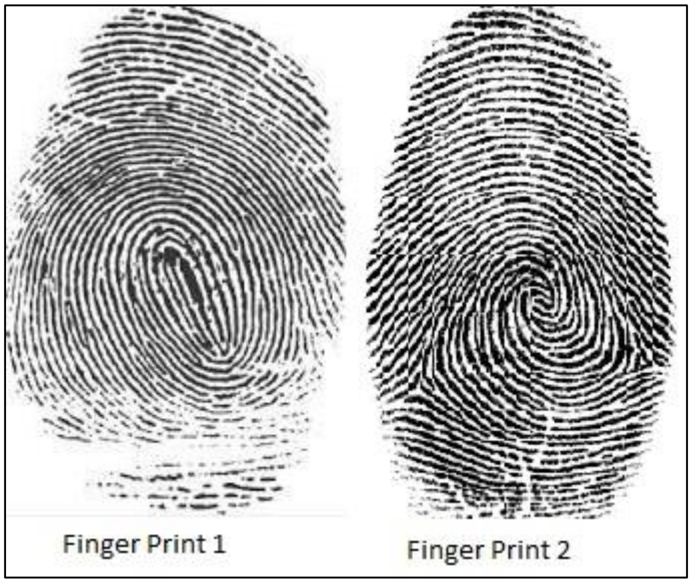
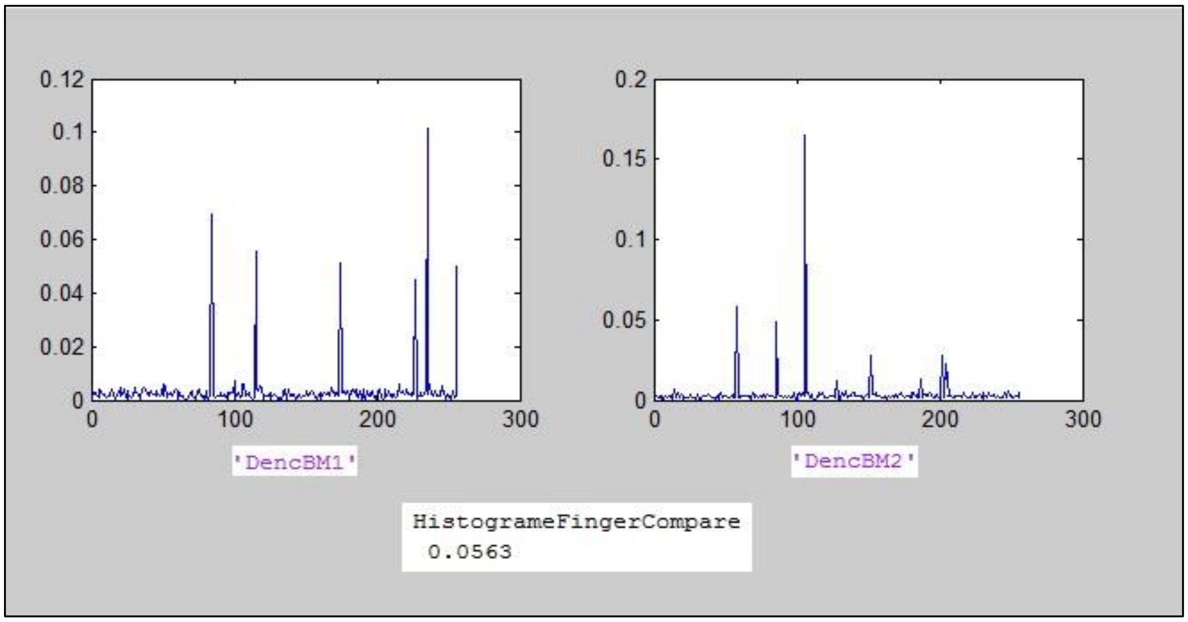


Figure 5.8: Actual fingerprints

A screenshot has been taken of the comparison results between these two obfuscated fingerprints, as an example. The DencBM1 is the obfuscated matrix of fingerprint1, and the DencBM2 is the obfuscated matrix of fingerprint2. Thus, the result of comparison is 0.0563, which is greater than 0.000009, and this is our threshold in the comparison process that has been defined for our experiment. Therefore, the result of comparing these two fingerprints is not matched. Consider Figure 5.9.



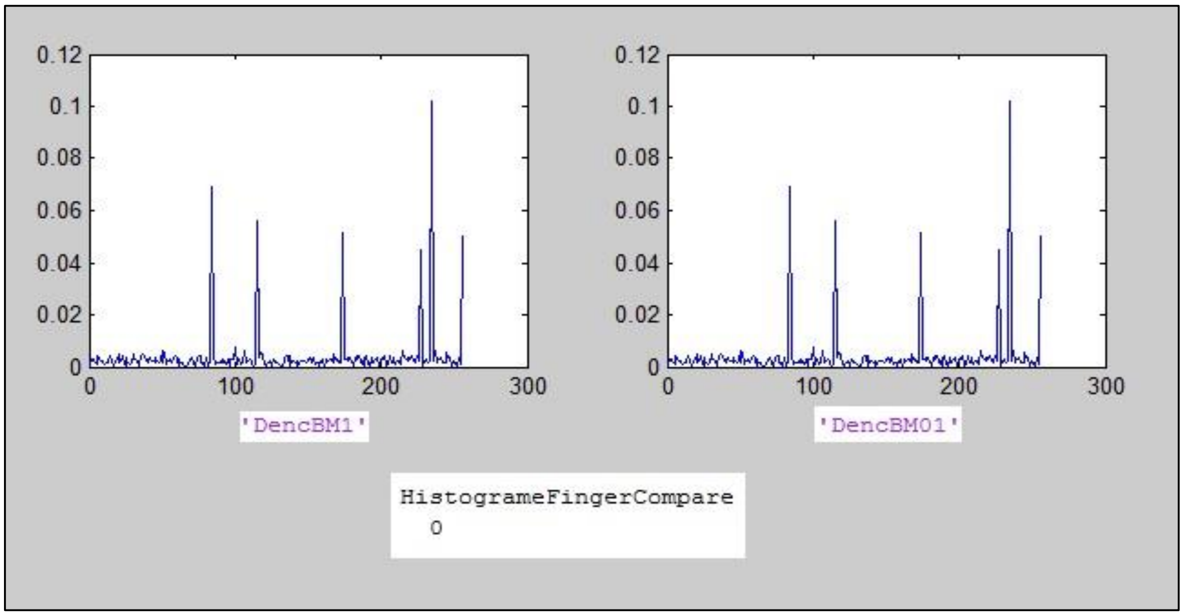
```
HistogrameFingerCompare
 0.0563
```

Figure 5.9: Result of comparing between two different obfuscated fingerprints.

The second case compares the obfuscated fingerprint1 with the same image stored in its obfuscated form. The DencBM1 is the obfuscated matrix of fingerprint1 and the DencBM01 is the obfuscated matrix of the previously stored fingerprint1. The result of this comparison is shown in Figure 5.10.

Note that the server does not only do the comparison, it also stores each obfuscated fingerprint. The server stores a data set of all obfuscated fingerprint matrices to compare between them and the input one. The server returns the comparison results as “Exist” if the input obfuscated fingerprint matches with one of the stored obfuscated fingerprints. Otherwise the result will be “Not Exist.”

If the result is “Exist,” the server returns the histograms of the matching obfuscated fingerprints along with a copy of the obfuscated matched matrix already stored. The client has a choice to verify if the result is correct. In this case, the client needs to spend extra resources for the comparison between two matrices. In this case, the clients de-obfuscates the matrix received from the server.



```
HistogrameFingerCompare
  0
```

Figure 5.10: Result of comparing between two obfuscated fingerprints

5.5. Implementing de-obfuscating process

After receiving the matched obfuscated matrix from the server, the client de-obfuscates it and compares it with the original integer matrix of the fingerprint. The client keeps the original integer matrix generated from the current input fingerprint during the preprocessing stage, until it receives the final result from the server, and compares that result with its temporary saved matrix.

The client de-obfuscates the received obfuscated matrix of the fingerprint that the server claims it has matched with the input matrix, in order to compare it with the original matrix of the input fingerprint.

In this section, the algorithms and techniques for implementing the de-obfuscating process is presented and described in detail. In the de-obfuscating process, we use some mathematical operations. This process has been implemented with Java, because Java provides many mathematical operations that simplify the implementation of these techniques. These operations save both time complexity and space complexity of the de-obfuscating process.

Consider Algorithm 5.9, which shows how the de-obfuscating process works.

Algorithm 5.9 De-Obfuscating Stage

Input: obfuscated matrix encBM[1..m][1..n] of obfuscated objects, private key PK[1..x]

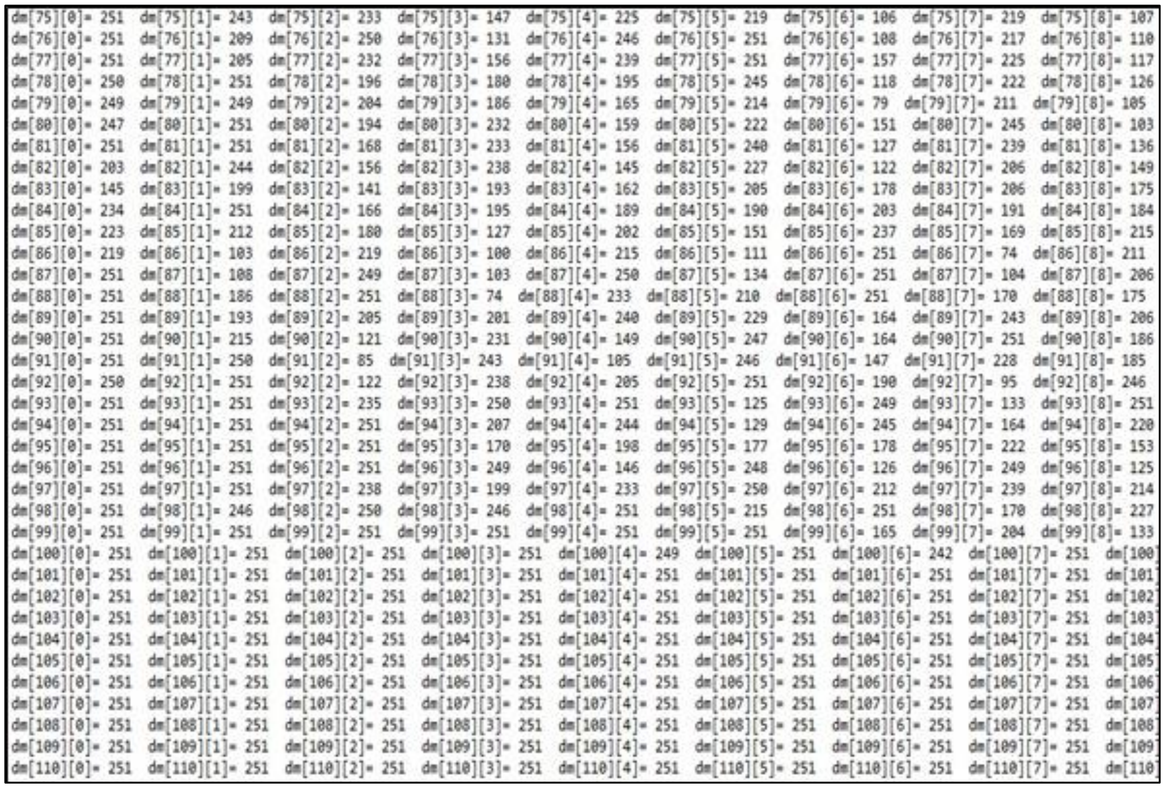
Output: de-obfuscated matrix d[1..m][1..2n] of integer elements.

Steps:

```
1. dc ← 0
2. for r ← 0 to m-1
3.     for c← 0 to n-1
4.          dec ← result of de-obfuscated integer value of
            obfuscated content of encBM[r][c]
5.          break dec block into two integer number by using
            hashed digit counter content in encBM[r][c].
6.          d[r][dc] ← first resulted integer number of
            breaking block dec
7.          Increment dc
8.          d[r][dc] ← second resulted integer number of
            breaking block dec
9.          Increment dc
10.     end for
11.     dc ← 0
12.  end for
```

This algorithm is mainly dependent on the number of rows and columns of encBM matrix. Thus, if m is the number of rows and n is the number of columns, then time complexity of this algorithm is O(mn).

A screenshot has been taken of the result of the de-obfuscating process. However, only a segment of the dm matrix is shown in Figure 5.11.



```
dm[75][0]= 251  dm[75][1]= 243  dm[75][2]= 233  dm[75][3]= 147  dm[75][4]= 225  dm[75][5]= 219  dm[75][6]= 106  dm[75][7]= 219  dm[75][8]= 107
dm[76][0]= 251  dm[76][1]= 209  dm[76][2]= 250  dm[76][3]= 131  dm[76][4]= 246  dm[76][5]= 251  dm[76][6]= 108  dm[76][7]= 217  dm[76][8]= 110
dm[77][0]= 251  dm[77][1]= 205  dm[77][2]= 232  dm[77][3]= 156  dm[77][4]= 239  dm[77][5]= 251  dm[77][6]= 157  dm[77][7]= 225  dm[77][8]= 117
dm[78][0]= 250  dm[78][1]= 251  dm[78][2]= 196  dm[78][3]= 180  dm[78][4]= 195  dm[78][5]= 245  dm[78][6]= 118  dm[78][7]= 222  dm[78][8]= 126
dm[79][0]= 249  dm[79][1]= 249  dm[79][2]= 204  dm[79][3]= 186  dm[79][4]= 165  dm[79][5]= 214  dm[79][6]= 79  dm[79][7]= 211  dm[79][8]= 105
dm[80][0]= 247  dm[80][1]= 251  dm[80][2]= 194  dm[80][3]= 232  dm[80][4]= 159  dm[80][5]= 222  dm[80][6]= 151  dm[80][7]= 245  dm[80][8]= 103
dm[81][0]= 251  dm[81][1]= 251  dm[81][2]= 168  dm[81][3]= 233  dm[81][4]= 156  dm[81][5]= 240  dm[81][6]= 127  dm[81][7]= 239  dm[81][8]= 136
dm[82][0]= 203  dm[82][1]= 244  dm[82][2]= 156  dm[82][3]= 238  dm[82][4]= 145  dm[82][5]= 227  dm[82][6]= 122  dm[82][7]= 206  dm[82][8]= 149
dm[83][0]= 145  dm[83][1]= 199  dm[83][2]= 141  dm[83][3]= 193  dm[83][4]= 162  dm[83][5]= 205  dm[83][6]= 178  dm[83][7]= 206  dm[83][8]= 175
dm[84][0]= 234  dm[84][1]= 251  dm[84][2]= 166  dm[84][3]= 195  dm[84][4]= 189  dm[84][5]= 190  dm[84][6]= 203  dm[84][7]= 191  dm[84][8]= 184
dm[85][0]= 223  dm[85][1]= 212  dm[85][2]= 180  dm[85][3]= 127  dm[85][4]= 202  dm[85][5]= 151  dm[85][6]= 237  dm[85][7]= 169  dm[85][8]= 215
dm[86][0]= 219  dm[86][1]= 103  dm[86][2]= 219  dm[86][3]= 100  dm[86][4]= 215  dm[86][5]= 111  dm[86][6]= 251  dm[86][7]= 74  dm[86][8]= 211
dm[87][0]= 251  dm[87][1]= 108  dm[87][2]= 249  dm[87][3]= 103  dm[87][4]= 250  dm[87][5]= 134  dm[87][6]= 251  dm[87][7]= 104  dm[87][8]= 206
dm[88][0]= 251  dm[88][1]= 186  dm[88][2]= 251  dm[88][3]= 74  dm[88][4]= 233  dm[88][5]= 210  dm[88][6]= 251  dm[88][7]= 170  dm[88][8]= 175
dm[89][0]= 251  dm[89][1]= 193  dm[89][2]= 205  dm[89][3]= 201  dm[89][4]= 240  dm[89][5]= 229  dm[89][6]= 164  dm[89][7]= 243  dm[89][8]= 206
dm[90][0]= 251  dm[90][1]= 215  dm[90][2]= 121  dm[90][3]= 231  dm[90][4]= 149  dm[90][5]= 247  dm[90][6]= 164  dm[90][7]= 251  dm[90][8]= 186
dm[91][0]= 251  dm[91][1]= 250  dm[91][2]= 85  dm[91][3]= 243  dm[91][4]= 105  dm[91][5]= 246  dm[91][6]= 147  dm[91][7]= 228  dm[91][8]= 185
dm[92][0]= 250  dm[92][1]= 251  dm[92][2]= 122  dm[92][3]= 238  dm[92][4]= 205  dm[92][5]= 251  dm[92][6]= 190  dm[92][7]= 95  dm[92][8]= 246
dm[93][0]= 251  dm[93][1]= 251  dm[93][2]= 235  dm[93][3]= 250  dm[93][4]= 251  dm[93][5]= 125  dm[93][6]= 249  dm[93][7]= 133  dm[93][8]= 251
dm[94][0]= 251  dm[94][1]= 251  dm[94][2]= 251  dm[94][3]= 207  dm[94][4]= 244  dm[94][5]= 129  dm[94][6]= 245  dm[94][7]= 164  dm[94][8]= 220
dm[95][0]= 251  dm[95][1]= 251  dm[95][2]= 251  dm[95][3]= 170  dm[95][4]= 198  dm[95][5]= 177  dm[95][6]= 178  dm[95][7]= 222  dm[95][8]= 153
dm[96][0]= 251  dm[96][1]= 251  dm[96][2]= 251  dm[96][3]= 249  dm[96][4]= 146  dm[96][5]= 248  dm[96][6]= 126  dm[96][7]= 249  dm[96][8]= 125
dm[97][0]= 251  dm[97][1]= 251  dm[97][2]= 238  dm[97][3]= 199  dm[97][4]= 233  dm[97][5]= 250  dm[97][6]= 212  dm[97][7]= 239  dm[97][8]= 214
dm[98][0]= 251  dm[98][1]= 246  dm[98][2]= 250  dm[98][3]= 246  dm[98][4]= 251  dm[98][5]= 215  dm[98][6]= 251  dm[98][7]= 170  dm[98][8]= 227
dm[99][0]= 251  dm[99][1]= 251  dm[99][2]= 251  dm[99][3]= 251  dm[99][4]= 251  dm[99][5]= 251  dm[99][6]= 165  dm[99][7]= 204  dm[99][8]= 133
dm[100][0]= 251  dm[100][1]= 251  dm[100][2]= 251  dm[100][3]= 251  dm[100][4]= 249  dm[100][5]= 251  dm[100][6]= 242  dm[100][7]= 251  dm[100]
dm[101][0]= 251  dm[101][1]= 251  dm[101][2]= 251  dm[101][3]= 251  dm[101][4]= 251  dm[101][5]= 251  dm[101][6]= 251  dm[101][7]= 251  dm[101]
dm[102][0]= 251  dm[102][1]= 251  dm[102][2]= 251  dm[102][3]= 251  dm[102][4]= 251  dm[102][5]= 251  dm[102][6]= 251  dm[102][7]= 251  dm[102]
dm[103][0]= 251  dm[103][1]= 251  dm[103][2]= 251  dm[103][3]= 251  dm[103][4]= 251  dm[103][5]= 251  dm[103][6]= 251  dm[103][7]= 251  dm[103]
dm[104][0]= 251  dm[104][1]= 251  dm[104][2]= 251  dm[104][3]= 251  dm[104][4]= 251  dm[104][5]= 251  dm[104][6]= 251  dm[104][7]= 251  dm[104]
dm[105][0]= 251  dm[105][1]= 251  dm[105][2]= 251  dm[105][3]= 251  dm[105][4]= 251  dm[105][5]= 251  dm[105][6]= 251  dm[105][7]= 251  dm[105]
dm[106][0]= 251  dm[106][1]= 251  dm[106][2]= 251  dm[106][3]= 251  dm[106][4]= 251  dm[106][5]= 251  dm[106][6]= 251  dm[106][7]= 251  dm[106]
dm[107][0]= 251  dm[107][1]= 251  dm[107][2]= 251  dm[107][3]= 251  dm[107][4]= 251  dm[107][5]= 251  dm[107][6]= 251  dm[107][7]= 251  dm[107]
dm[108][0]= 251  dm[108][1]= 251  dm[108][2]= 251  dm[108][3]= 251  dm[108][4]= 251  dm[108][5]= 251  dm[108][6]= 251  dm[108][7]= 251  dm[108]
dm[109][0]= 251  dm[109][1]= 251  dm[109][2]= 251  dm[109][3]= 251  dm[109][4]= 251  dm[109][5]= 251  dm[109][6]= 251  dm[109][7]= 251  dm[109]
dm[110][0]= 251  dm[110][1]= 251  dm[110][2]= 251  dm[110][3]= 251  dm[110][4]= 251  dm[110][5]= 251  dm[110][6]= 251  dm[110][7]= 251  dm[110]
```

Figure 5.11: A part of the 2D dm matrix

Then a small comparison is done by using the Matlab environment, because it can easily and simply compare between two matrices in linear time. We use the histogram-comparison method to do so. Consider Figure 5.12, which shows an example of the comparison results between the input fingerprint’s original integer matrix and the de-obfuscated fingerprint, which has been converted into its original integer matrix form.

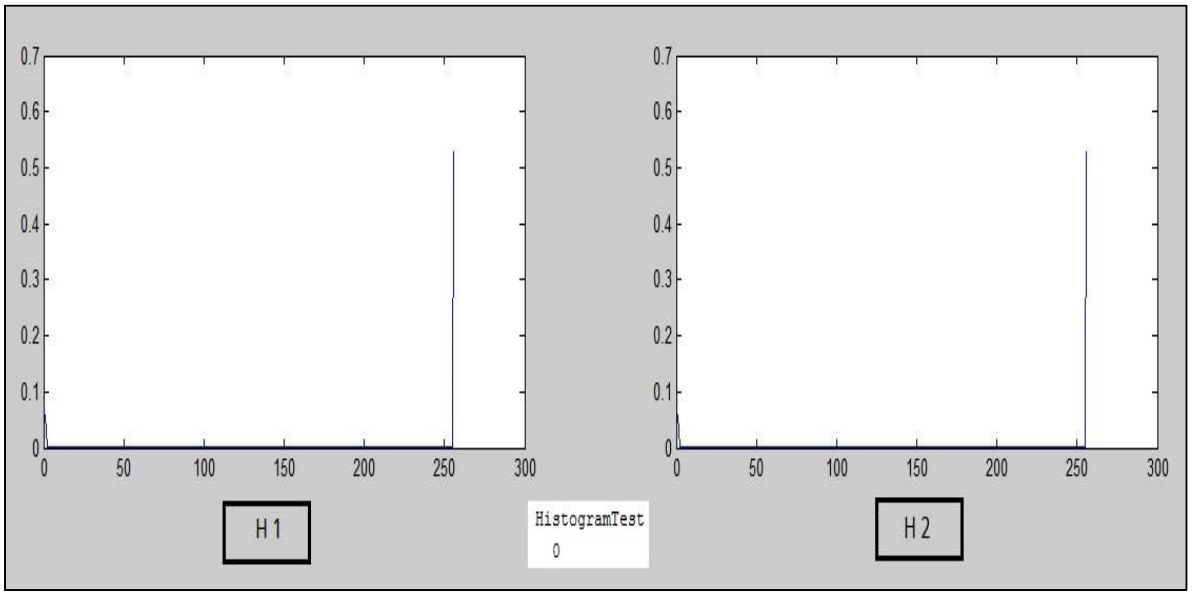


Figure 5.12: Result of comparing original integer matrices

Figure 5.12 shows the histogram test between the integer matrix of the input fingerprint (H1), and the received de-obfuscated matrix (H2). The result of the comparison is equal to zero, which means that the server computation result is correct.

In this chapter, we have presented the implementation of our proposed framework. We have demonstrated that the result of obfuscating data using our framework is reasonably acceptable, meaning that our proposed framework can be used to hide actual data from the cloud server without affecting the comparing process of the server. We have also reduced the computation by using sets of blocks with fewer private keys, which positively impacts the computational complexity. In the next chapter, we will present the result of testing and compare it with other similar approaches.

CHAPTER 6: DISCUSSION AND EVALUATION

This chapter presents a critical examination of our findings and discusses what has been learned from our research. In order to test the effectiveness of the proposed framework, two experiments are conducted. In the first experiment, we are going to show the effects of reducing the image size on both time and accuracy of the final results. While in the second experiment, we are going to show the effects of various amounts of used sets for grouping the obfuscated matrix elements. In addition, in this chapter we outline, with impartiality, the advantages and the disadvantages of our approach. We also compare our proposed approach with other similar techniques, in regard to used techniques, speed, and security.

6.1 Image-reduction experiment

In this experiment, we test how both the obfuscation time and the comparison accuracy are affected by reducing the image size. A comparison between six fingerprint images is done in their original form and on their obfuscated form. The reduction criterion we used is the number of pixels of the image size. As we reduce image size by three turns—that is, in turn 1—we reduce the size into half of the original size; in turn 2, we further reduce the image by half, which is equivalent to one-fourth of the original image size; and the reduction continues in that manner. Note that each image’s original dimension is 172 * 255 pixels. In turn 1, the image is reduced to 86 * 129 pixels. In the second reduction turn, the reduced size resulting from turn 1 is further reduced to 44 * 66 pixels. In the last turn, it is reduced to 22 * 34 pixels.

Consider Table 6.1, which shows the comparison results of each reduction turn and the average error and the percentage error of each reduction turn. After each reduction turn, we compare the reduced image size of the fingerprint with the first fingerprint image (denoted as Fing1), which also has been reduced by using the same reduction process. Therefore, we first compare each fingerprint image with the first fingerprint image (Fing1) before reducing them, and their original size is 177 * 255 pixels. The value of comparing these fingerprints represents the accurate comparison value. Then all of the fingerprint images are reduced to their halved size, and each of these reduced images is compared with the reduced fingerprint 1 (Fing1). We do the reduction turn three times, and each time, we compare each of these reduced images with the first fingerprint, which is also being reduced by the same reduction rate. After collecting all of these comparison values, we calculate the average error for each turn and compare it with the average error of the comparing images before reducing them. The experiment scenario is the following:

1. Compare each fingerprint image with the first (Fing1); in this step all fingerprint images are in their original size, 177 * 255 pixels.
2. Obfuscate each fingerprint image and compare each fingerprint image with the first obfuscated fingerprint image (enc (Fing1)).
3. Defined the required time for obfuscating each fingerprint image.
4. Reduce each fingerprint image to its half size.
5. Compare each reduced fingerprint image with the first one reduced (Fing1).
6. Obfuscate each reduced fingerprint image and compare each reduced fingerprint image with the first obfuscated and reduced fingerprint image (enc (Fing1)).
7. Define the required time for obfuscating each reduced fingerprint image.

8. Repeat step 4 to do the calculation for the second reduction turn.

9. Repeat step 4 to do the calculation for the third reduction turn.

After collecting the result of each step, we count the average error and the percentage error of each reduction turn, and the standard comparison result is the result of comparing each image in its original form without reducing its size.

Table 6.1: Reduction Test: Comparing Results

Size	**Original Size T0 172 * 255**	**Reduction Turn T1 86 * 129**	**Reduction Turn T2 44 * 66**	**Reduction Turn T3 24 * 34**
Fing1	0	0	0	0
enc(Fing1)	0	0	0	0
Fing2	0.0059	0.008937	0.0032	0.0378
enc(Fing2)	0.0057	0.0082	0.0028	1.0069
Fing3	0.0347	0.057069	0.0013	0.0345
enc(Fing3)	0.028	0.031	0.0014	0.0925
Fing4	0.0128	0.0071	0.0092	0.0513
enc(Fing4)	0.0116	0.0023	0.0075	0.0057
Fing5	0.00066327	0.00048757	0.0015	0.0162
enc(Fing5)	0.00065335	0.0033	0.0021	0.009131
Fing6	0.0147	0.0123	0.0155	0.053
enc(Fing6)	0.0241	0.0038	0.0167	0.079514
Image comp Avg error	0.011460545	0.014315595	0.005116667	0.032133333
Enc image comp Avg error	0.011675558	0.0081	0.005083333	0.1989575
Image comp Percentage error	0	24.91199153	55.35407202	180.38
Enc image comp Percentage error enc	1.8761179	29.32273291	55.64492497	1636.02128

Table 6.1 shows that the error rate is directly proportional with image reduction, which means that the image-compression accuracy is inversely proportional with image reduction. However, there is a small rate of errors when the image is obfuscated, which means that obfuscating the image does not have a significant effect on the image-compression accuracy. Consider Figure 6.1, which shows the effect of image reduction on the accuracy of the process.

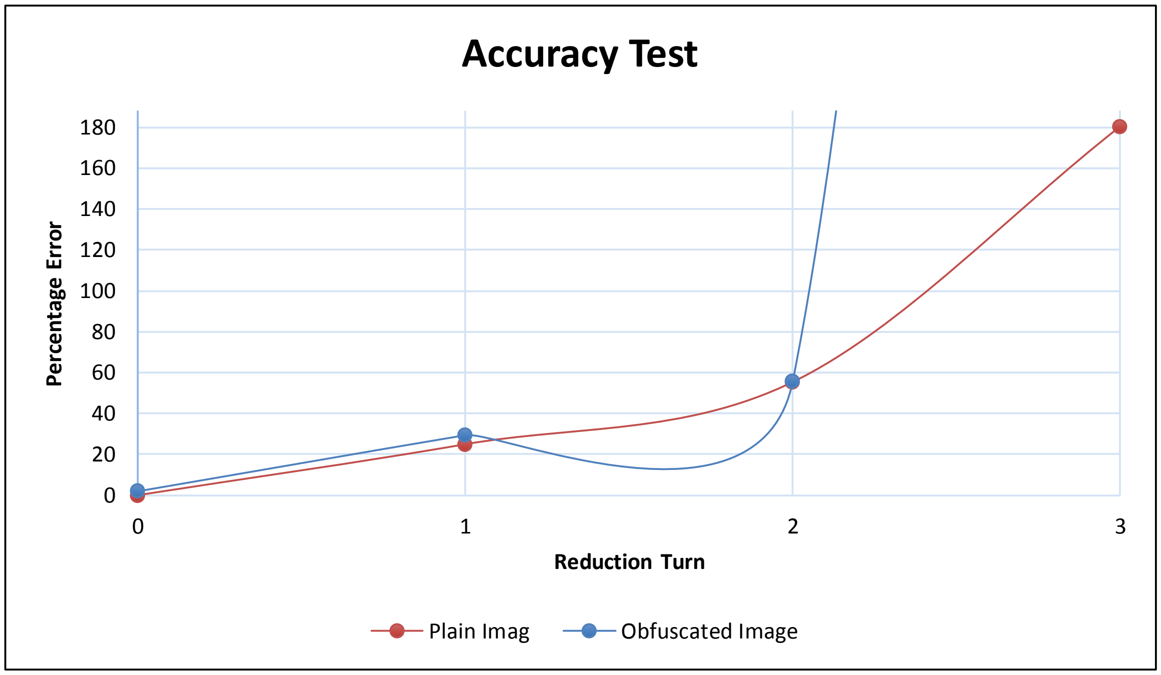


Figure 6.1: Reduction Test: Comparison Results

Consider Table 6.2, which records the required time to obfuscate each image in each reduction turn.

Table 6.2: Reduction Test: Obfuscation Time Test

Size	**Original Size T0 172 * 255**	**Reduction Turn T1 86 * 129**	**Reduction Turn T2 44 * 66**	**Reduction Turn T3 24 * 34**
encTime(Fing1)	4.921	1.784	0.524	0.2
encTime(Fing2)	4.549	1.621	0.54	0.21
encTime(Fing3)	4.946	1.691	0.52	0.2
encTime(Fing4)	4.865	1.662	0.5	0.2

encTime(Fing5)	4.698	1.603	0.78	0.46
encTime(Fing6)	4.918	1.532	0.52	0.19
Time Obfuscation Avg	4.816166667	1.648833333	0.564	0.243333333

Table 6.2 shows that the obfuscation time is directly proportional to the image reduction, because as much as the image is reduced, the required time to obfuscate is also reduced accordingly. Figure 6.2 shows the effect of image reduction on the obfuscation time.

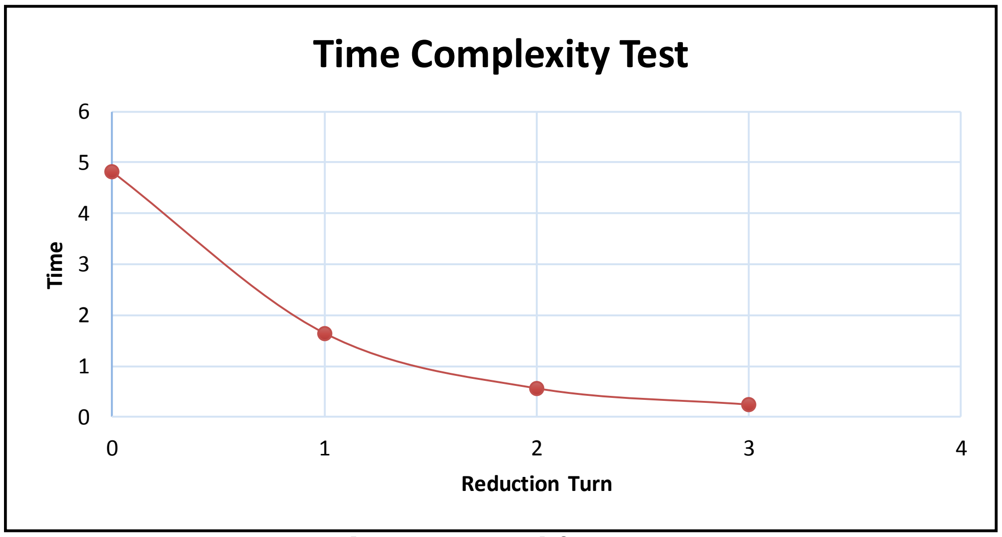


Figure 6.2: Reduction Test: Obfuscation Time Test

The presented results of the image-reduction tests suggest that the accuracy of the image comparison is affected by reducing image size. However, the required time to obfuscate the image is significantly decreased if the image size is reduced, yet we must choose an appropriate reduction size. As our experiment shows, the turn 1 reduction (86 * 129) is the most optimal because it does not significantly impact or vary in the comparison result. At the same time, turn 1 significantly reduces the required time for the obfuscated image. On the other hand, in order to solve the error rate of the comparison result, we can decrease the image-comparison threshold. For example, if the threshold is equal to 0.003

and if the result of comparing two images is less than or equal to that threshold, we reasonably conclude that these two images are nearly the same. However, this threshold can be decreased further to get a more accurate comparison result. Note that the threshold value is defined only for our experiment purposes.

6.2 Sets experiment

In this experiment we test how both the obfuscation time and the comparison accuracy are affected by the total number of sets used in each fingerprint image. A comparison between six fingerprint images is done in their obfuscated form. The changing parameter is the total number of sets used. We first group all obfuscated matrix elements into only one set. Then we distribute the obfuscated matrix elements into five sets, then into ten sets, and so on, until we have thirty sets. In each distributing cycle we compare each fingerprint image with the first fingerprint image (Fing1), and the size of each fingerprint is reduced from 172 * 255 elements to 86 * 129 elements. The experiment scenario is the following:

1. Obfuscate each fingerprint image by using only one set.
2. Compute the required time for obfuscating each fingerprint image.
3. Compare each obfuscated fingerprint image with the first image obfuscated (Fing1).
4. Obfuscate each fingerprint image by using five sets.
5. Compute the required time for obfuscating each fingerprint image.
6. Compare each obfuscated fingerprint image with the first obfuscated (Fing1).

We repeat step 4 by increasing the total number of sets by five, until we reach thirty sets.

After collecting the result of each step, we count the average and the percentage errors of each grouping cycle, and the standard comparison result is the result of comparing each obfuscated image by using only one set. Consider Table 6.3, which shows the comparison results of each grouping cycle and the average and the percentage errors of each grouping cycle.

Table 6.3: Set Test: Comparison Results

Sets	**0**	**5**	**10**	**15**	**20**	**25**	**30**
Fing1	0	0	0	0	0	0	0
Fing2	0.0082	0.009	0.0143	0.01231	0.0169	0.0109	0.01511
Fing3	0.031	0.0079	0.0127	0.0196	0.0113	0.0381	0.0378
Fing4	0.0023	0.0065	0.0141	0.01222	0.0113	0.01044	0.01488
Fing5	0.0033	0.0121	0.0146	0.00184	0.018	0.00533	0.00409
Fing6	0.0038	0.0077	0.0035	0.0182	0.0094	0.0037	0.00364
Error enc Avg	0.0081	0.0072	0.009867	0.010695	0.01115	0.011412	0.012587
Percentage Error enc	0	11.1111	21.8107	32.03703704	37.65432	40.88477	55.39095

Table 6.3 shows that the error rate is directly proportional to the total amount of sets used, which means that the image-compression accuracy is inversely proportional to the total amount of sets used. Figure 6.3 below shows the impact of the total number of sets used to obfuscate the image on the image-compression accuracy. The experiment clearly demonstrates that when a greater number of sets are used, the error rates gradually increase. For example, the error rate for ten sets is around 21. This increased to 32 for 15 sets. However, it slightly increases to 38 for 20 sets, and it keeps on increasing as more sets are used to obfuscate the image.

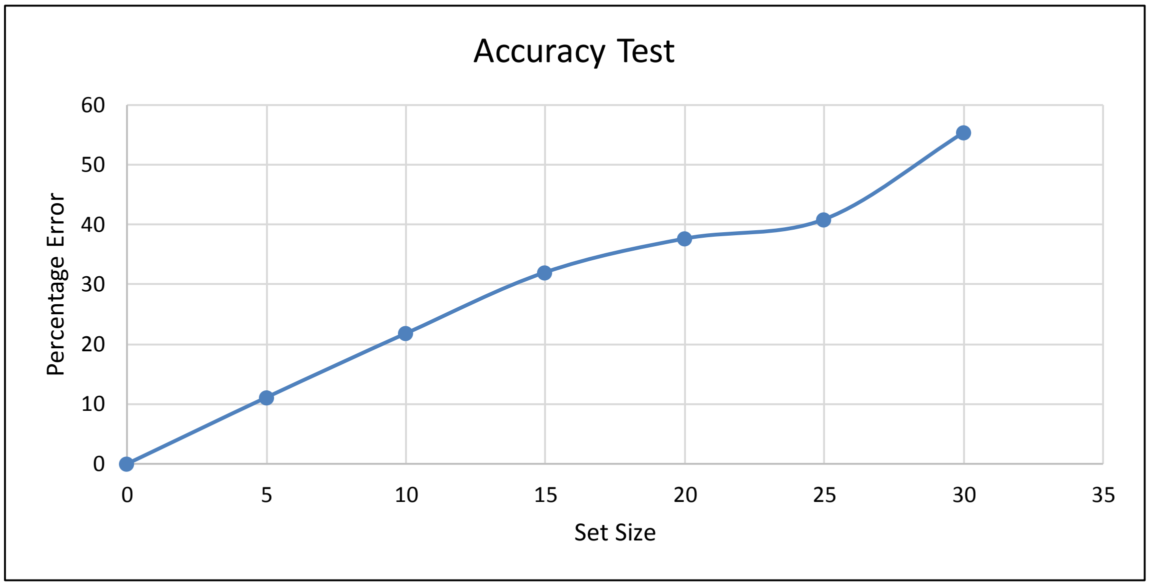


Figure 6.3: Set Test: Comparison Results

Table 6.4: Set Test: Obfuscation Time Test

Sets	**0**	**5**	**10**	**15**	**20**	**25**	**30**
encTime (Fing1)	1.1611	1.523	1.784	1.611	1.944	1.688	1.781
encTime (Fing2)	1.921	1.592	1.621	1.881	1.611	1.617	2.001
encTime (Fing3)	1.621	1.623	1.691	1.599	1.941	2.073	1.571
encTime (Fing4)	1.581	1.623	1.662	1.611	1.6	1.691	1.733
encTime (Fing5)	1.612	1.901	1.603	1.631	1.571	1.782	1.894
encTime (Fing6)	1.73	1.572	1.532	1.931	1.601	1.651	1.601
Time Obfuscation Avg	1.60435	1.639	1.64883	1.71067	1.71133	1.75033	1.7635

Consider Table 6.4. It records the required time to obfuscate each image in each grouping cycle. Table 6.4 shows that the obfuscation time does change steadily with the total number of sets used, because in each grouping cycle, the required time for obfuscating keeps on increasing. We can conclude that the total number of sets used to obfuscate an image does not have a significant impact on the required time for obfuscating, because none of the presented grouping cycles requires more than two seconds in order to be obfuscated. Consider Figure 6.4, which shows the total required obfuscated time using different set sizes.

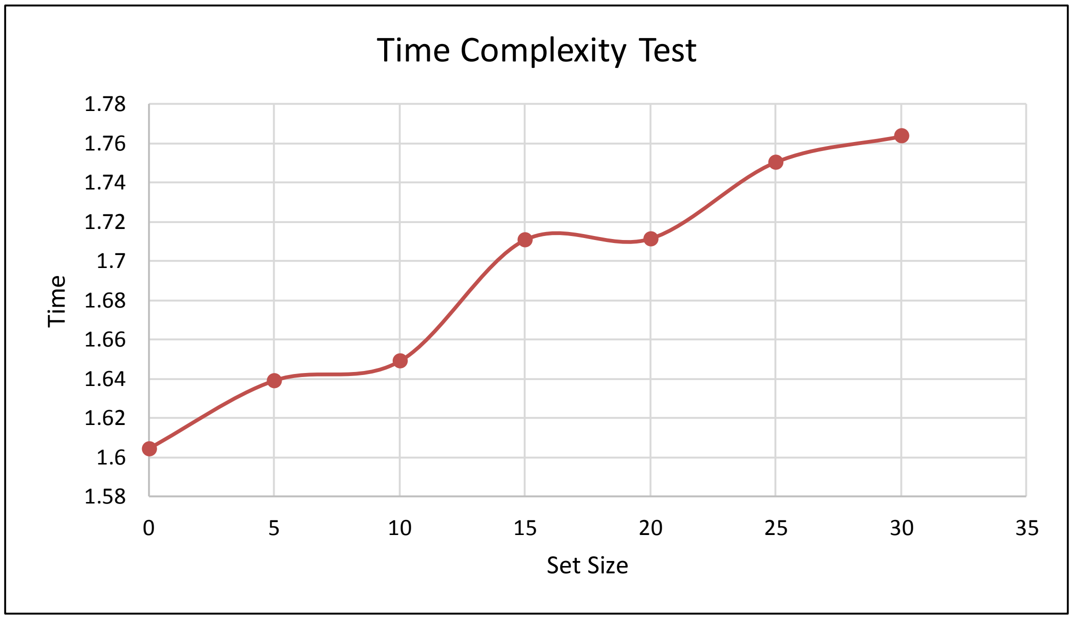


Figure 6.4: Set Test: Obfuscation Time Test

With these results of the set test, we conclude that the total number of sets used for obfuscating an image impacts the comparison time. The required time for obfuscating image is increased as the number of the required for encrypting blocks that belong to each set. Note that in our approach the total number of sets is defined randomly by the program; however, in this experiment we have controlled that size and the total number of used sets.

In order to solve the error rate of the comparison result, we can decrease the image-comparison threshold. For example, if the threshold is equal to 0.003, and the result of comparing two images is less than or equal to this threshold, we can conclude that the two images are nearly the same. However, after obfuscating the image by any number of sets, we can decrease the threshold in order to get an accurate comparison result. Note that the threshold value is defined for the experiment purposes.

6.3 Discussion

In general, our framework first reduces the image size to an appropriate size and converts it from a colored image to gray to reduce the number of dimensions of the matrix—that is, the image is changed from three to two dimensions. These steps, size reduction and changing dimensions, are essential for saving computational complexity in the obfuscating stage as shown in Table 6.2. Our experiments show the direct impact of reducing image size on the obfuscation time. Note that if the image is not converted to gray, then the obfuscation will take O(n3); whereas after converting the image to gray, the obfuscation work requires O(n2), which is the same as reported in [9]. However, the obfuscation operations in our framework require less computation than required for the approach in [24], which are O(n3) operations in the obfuscation stage.

Our framework is based on a single server system. Therefore, our approach, to verify the server's results, must be an FHE system, or we must incorporate some operations done on the client's side to verify the server's final results. Therefore, our verification operation is done only if the server's final result is "Exist!!" Once the server claims that there is a match between the input image and one of the stored images, it returns the obfuscated matched image to the client, so the client can decrypt it and compare it with the input image that was sent to the server. All of these operations require O(n2) work, which is the same verification requirement in [9]. However, in [24] the verification operations require less work because it only needs O(n) work. Nevertheless, our framework does not leak any private information as reported in the [25] approach.

Our approach uses the tough-to-crack Rabin cipher, which is highly secure because of its resistance to factorization. However, if its public key is factorized into the correct two prime numbers that it consists of, then this secure technique will likely be cracked. Therefore, in our framework we omit the public key and choose to use only the private key; this approach leaves no clues that malicious entities could use to crack the cipher. Using our new techniques, we encrypt our plain data by using multiple keys for multiple sets. Therefore, our approach is vastly more secure than the approach in [27] because our framework requires less computational complexity. Because our public key is factorized into three large prime numbers, rather than the two in the decryption technique in [27], the result of decryption is equal to eight candidate roots rather than two, as there isn't any technique to identify the correct root. In our approach, we keep using two large prime numbers without using a public key, and our decryption technique results in only four candidate roots using our tokenization technique, which helps us to find the correct root among these four roots. In our approach, once it recognizes the correct root, it stops computing the other roots.

In our framework, we use the Garner algorithm [5] in the decryption technique, because it reduces the multiplication operations used for solving the Chinese remainder theorem by half. In [5], the approach reduces the resulting four roots to two; however, it does not use any technique to discover the correct one. In our approach, the decryption technique still produces four roots, but we can recognize the correct root among these four by using our tokenization technique.

In order to obfuscate a matrix, the existing approaches provide two choices to encrypt the elements in the matrix: One is to encrypt the entire matrix using only one private key, and this saves the client excessive computational efforts. The other choice is to encrypt each element of the matrix with different private keys, resulting in the client having to save a large number of keys. In the first choice, there is a high risk that all elements will be disclosed if the key is compromised. In the second approach, although the computational overhead is high, compromising one key reveals only one element. Both approaches have their own advantages and disadvantages. We address the disadvantages of both approaches by grouping matrix elements into sets of different sizes, and each element in a set is encrypted by the same private key of the set, where each set has its own private key to encrypt its elements. The client needs to generate and save an array of private keys, and the size of that array should be equal to the number of sets used to group matrix elements. This approach is a compromise between the two existing approaches.

However, our proposed framework has a few limitations. The first weakness is that all outsourced data, in order to be computed correctly by the server, must be encrypted with the same array of private keys. This is because all obfuscated images are of the same size and the elements of each matrix must be grouped in the same order of sets with the same variances in size.

The second weakness of our approach is that when the client sends an image to be computed by the server, the client must save the original integer matrix of the input image, because our de-obfuscation technique only returns the original integer matrix of the image. Therefore, in order to check the results sent by the server, the client must keep the original integer matrix so that he or she can compare it with the de-obfuscated integer matrix received from the server. This problem can be addressed by adding an additional tool to the de-obfuscation technique. The new tool can transform the integer matrix to a matrix of pixels. However, this issue is outside the scope of this research.

The third weakness is that we noticed that our decryption techniques sometimes confuse and wrongly recognize an incorrect root among all of the four resulting roots. This problem occurs only if two of the four resulting roots have the same token value. However, this problem rarely occurs when we decrypt a matrix of size 86 * 129.

6.3.1. Comparative Analysis

In this section, we provide a comparison that considers the complexity order, advantages, and disadvantages of the Rabin and RSA improvements in our approach. We also provide another comparison that considers the complexity order, advantages, and disadvantages of the proposed systems in [9] and [24] with our system.

Consider Table 6.5 below, which shows the comparison of the complexity order, advantages, and disadvantages of each Rabin improvement in our approach.

Table 6.5: Comparative Analysis between Rabin's and RSA Algorithms.

Algorithm	**Encryption Speed**	**Decryption Speed**	**Advantages**	**Disadvantages**
[48]	$O(n2)$	$O(n3)$	Overcomes Rabin decryption failure. No extra computation is required during decryption.	Plain text domain is restricted.
[30]	$O(n2)$	$O(n3)$	Overcomes decryption failure of solving CRT.	Requires more computational complexity in both encryption and decryption processes than the normal RSA technique.

[44]	*O*(*n*2)	*O*(*n*3)	Overcomes decryption failure of solving CRT.	Requires more computational complexity in both encryption and decryption processes than the normal RSA technique.
[37]	*O*(*n*2)	*O*(*n*3)	Security is higher than RSA	Decryption failure.
our approach	*O*(*n*2)	*O*(*n*2)	Overcomes decryption failure of solving CRT. High security. Decryption methodology is faster than the normal Rabin decryption methodology.	Decryption has small probability of failing.

Table 6.6: Comparative Analysis between Our System and Other Provided Systems.

System	**Encryption Speed**	**Decryption Speed**	**Advantages**	**Disadvantages**
[9]	*O*(*n*2)	*O*(*n*2)	High security	Decryption system may lack, if a failure happened in the verification server
[24]	*O*(*n*3)	*O*(*n*3)	Higher decryption speed than the other systems	Leak of privacy information

Our approach	$O(n2)$	$O(n2)$	Overcomes decryption failure of solving CRT. High security. Decryption methodology is faster than the normal Rabin decryption methodology.	All outsourced data must be decrypted with the same array of private keys. Decryption has small probability of failing.

CHAPTER 7: CONCLUSION AND FUTURE WORK

The main goal of this research was to develop an approach that can secure clients' outsourced data and can ensure privacy of these data while being processed by or stored in the cloud severs or any external agent. The client uses cloud computing mainly to do the complex computations on the outsourced data, in order to eventually save these data on the cloud server. The secure outsourcing would not be effective if the client uses effort-consuming and complex techniques, such as an FHE technique. The client needs to use a lightweight approach that can secure outsourced data and allow the cloud servers to compute on these data correctly without knowing the data's actual value. To achieve this multifaceted approach, this thesis has proposed techniques that not only address clients' security and privacy needs, but also allow cloud servers to compute on the outsourced data accurately without knowing the data's actual value or form. An added benefit to our approach is that if the cloud server or storage were compromised by unauthorized entities, the privacy of the obfuscated image would remain intact.

Our specific contributions in this thesis are:

- The proposed obfuscating technique uses a modified Rabin cipher without a public key.
- Our approach ensures that the possibility of guessing the two large prime numbers (private key) is very weak.
- We have introduced a new block technique, which builds blocks of different lengths, yet each block contains a fixed number of elements.

- We have decreased computational complexity by requiring fewer numbers of private keys, a strategy that is based on our new technique of creating a random number of sets with varied size.
- We have addressed the drawback of the Rabin cipher's decryption technique by introducing tokenization and a digit counter, both of which support the de-obfuscation operations.

The limitations identified in the previous chapter are to be addressed in future research.

- We could develop a new technique that could convert the integer matrix back to its image from.
- We aim to research further how we could use classification techniques that compare between images by using the features of images.
- We could explore how optimal garbled computing could efficiently be used to outsource images to cloud.

REFERENCES

[1] Adrian, David, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger et al. "Imperfect forward secrecy: How Diffie-Hellman fails in practice." In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 5-17. ACM, 2015.

[2] Ananth, Prabhanjan, Nishanth Chandran, Vipul Goyal, Bhavana Kanukurthi, and Rafail Ostrovsky. "Achieving privacy in verifiable computation with multiple servers–without FHE and without preprocessing." In *Public-Key Cryptography–PKC 2014*, pp. 149-166. Springer Berlin Heidelberg, 2014.

[3] Ariffin, Muhammad Rezal Kamel, M. A. Asbullah, and Nor Azman Abu. "A new efficient asymmetric cryptosystem based on the square root problem." *arXiv preprint arXiv:1207.1157* (2012).

[4] Ariffin, Muhammad Rezal Kamel, Muhammad Asyraf Asbullah, Nur Azman Abu, and Zahari Mahad. "A new efficient asymmetric cryptosystem based on the integer factorization problem of N= p2q." *Malaysian Journal of Mathematical Sciences* 7, no. S (2013): 19-37.

[5] Asbullah, Muhammad Asyraf, and Muhammad Rezal Kamel Ariffin. "Fast decryption method for a Rabin primitive-based cryptosystem." *International Journal of Advancements in Computing Technology* 6, no. 1 (2014): 56.

[6] Asbullah, Muhammad Asyraf, and Muhammad Rezal Kamel Ariffin. "Rabin-$ p $ cryptosystem: Practical and efficient method for Rabin based encryption scheme." *arXiv preprint arXiv:1411.4398* (2014).

[7] Atallah, Mikhail J., Konstantinos N. Pantazopoulos, John R. Rice, and Eugene E. Spafford. "Secure outsourcing of scientific computations." *Advances in Computers* 54 (2002): 215-272.

[8] Bellare, Mihir, and Phillip Rogaway. "Introduction to modern cryptography." *UCSD CSE* 207 (2005): 207.

[9] Benjamin, David, and Mikhail J. Atallah. "Private and cheating-free outsourcing of algebraic computations." In *Privacy, Security and Trust, 2008. PST'08. Sixth Annual*

Conference on, pp. 240-245. IEEE, 2008.

[10] Blanton, Marina, Mikhail J. Atallah, Keith B. Frikken, and Qutaibah Malluhi. "Secure and efficient outsourcing of sequence comparisons." In *Computer Security–ESORICS 2012*, pp. 505-522. Springer Berlin Heidelberg, 2012.

[11] Bellare, M., and P. Rogaway. (1994). "Optimal asymmetric encryption." In *Advances in Cryptology—EUROCRYPT'94* (pp. 92-111). Springer Berlin Heidelberg.

[12] Bishop, David. *Introduction to Cryptography with java Applets*. Jones & Bartlett Learning, 2003.

[13] Boneh, Dan. "Simplified OAEP for the RSA and Rabin functions." In *Advances in Cryptology—CRYPTO 2001*, pp. 275-291. Springer Berlin Heidelberg, 2001.

[14] Castagnos, Guilhem, Antoine Joux, Fabien Laguillaumie, and Phong Q. Nguyen. "Factoring pq 2 with quadratic forms: nice cryptanalyses." In *Advances in Cryptology–ASIACRYPT 2009*, pp. 469-486. Springer Berlin Heidelberg, 2009.

[15] Chung, Kai-Min, Yael Kalai, and Salil Vadhan. "Improved delegation of computation using fully homomorphic encryption." In *Advances in Cryptology–CRYPTO 2010*, pp. 483-501. Springer Berlin Heidelberg, 2010.

[16] Damgård, Ivan, and Yuval Ishai. "Constant-round multiparty computation using a black-box pseudorandom generator." In *Advances in Cryptology–CRYPTO 2005*, pp. 378-394. Springer Berlin Heidelberg, 2005.

[17] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51, no. 1 (2008): 107-113.

[18] Diffie, Whitfield, and Martin E. Hellman. "New directions in cryptography."*Information Theory, IEEE Transactions on* 22, no. 6 (1976): 644-654.

[19] Du, Wenliang, and Mikhail J. Atallah. "Privacy-preserving cooperative statistical analysis." In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pp. 102-110. IEEE, 2001.

[20] Du, Wenliang, and Mikhail J. Atallah. "Secure multiparty computation problems and their applications: a review and open problems." In*Proceedings of the 2001 workshop on New security paradigms*, pp. 13-22. ACM, 2001.

[21] Elia, Michele, Matteo Piva, and Davide Schipani. "The Rabin cryptosystem revisited." *arXiv preprint arXiv:1108.5935* (2011).

[22] Fox, Armando, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. "Above the clouds: A Berkeley view of cloud computing." *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28, no. 13 (2009): 2009.

[23] Galindo, David, Sebastiá Martýn, Paz Morillo, and Jorge L. Villar. "A practical public key cryptosystem from Paillier and Rabin schemes." In*Public Key Cryptography—PKC 2003*, pp. 279-291. Springer Berlin Heidelberg, 2003.

[24] Gennaro, Rosario, Craig Gentry, and Bryan Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers." In *Advances in Cryptology–CRYPTO 2010*, pp. 465-482. Springer Berlin Heidelberg, 2010.

[25] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." In *STOC*, vol. 9, pp. 169-178. 2009

[26] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "How to play any mental game." In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218-229. ACM, 1987.

[27] Hashim, Hayder Raheem. "H-RABIN CRYPTOSYSTEM." *Journal of Mathematics and Statistics* 10, no. 3 (2014): 304.

[28] Haraty, Ramzi A., A. N. El-Kassar, and Bilal Shibaro. "A comparative study of rsa based digital signature algorithms." *Journal of Mathematics and Statistics* 2, no. 1 (2006): 354-359.

[29] Katz, Jonathan, and Rafail Ostrovsky. "Round-optimal secure two-party computation." In *Advances in Cryptology–CRYPTO 2004*, pp. 335-354. Springer Berlin Heidelberg, 2004.

[30] Kurosawa, Kaoru, Toshiya Ito, and Masashi Takeuchi. "Public key cryptosystem

using a reciprocal number with the same intractability as factoring a large number." *Cryptologia* 12, no. 4 (1988): 225-233.

[31] Kurosawa, Kaoru, Wakaha Ogata, Toshihiko Matsuo, and Shuichi Makishima. "IND-CCA public key schemes equivalent to factoring n= pq." In *Public Key Cryptography*, pp. 36-47. Springer Berlin Heidelberg, 2001.

[32] Lindell, Yehuda. "Parallel coin-tossing and constant-round secure two-party computation." *Journal of Cryptology* 16, no. 3 (2003): 143-184.

[33] Malkhi, Dahlia, Noam Nisan, Benny Pinkas, and Yaron Sella. "Fairplay-Secure Two-Party Computation System." In *USENIX Security Symposium*, vol. 4. 2004.

[34] Motahari-Nezhad, Hamid R., Bryan Stephenson, and Sharad Singhal. "Outsourcing business to cloud computing services: Opportunities and challenges." *IEEE Internet Computing* 10 (2009).

[35] Neto, Paulo. "Demystifying cloud computing." In *Proceeding of Doctoral Symposium on Informatics Engineering*, 2011.

[36] Okamoto, Tatsuaki, and Shigenori Uchiyama. "A new public-key cryptosystem as secure as factoring." In *Advances in Cryptology—EUROCRYPT'98*, pp. 308-318. Springer Berlin Heidelberg, 1998.

[37] Rabin, Michael O. *Digitalized signatures and public-key functions as intractable as factorization*. No. MIT/LCS/TR-212. MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1979.

[38] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms." *Foundations of secure computation* 4, no. 11 (1978): 169-180.

[39] Rivest, Ronald L., Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM* 21, no. 2 (1978): 120-126.

[40] Roy, Indrajit, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. "Airavat: Security and Privacy for MapReduce." In *NSDI*, vol. 10, pp. 297-312. 2010.

[41] Schmidt-Samoa, Katja. "A new rabin-type trapdoor permutation equivalent to factoring." *Electronic Notes in Theoretical Computer Science* 157, no. 3 (2006): 79-94.

[42] Takagi, T. (1997). Fast RSA-type cryptosystems using n-adic expansion. In*Advances in Cryptology—CRYPTO'97* (pp. 372-384). Springer Berlin Heidelberg.

[43] Takagi, Tsuyoshi. "Fast RSA-type cryptosystem modulo p k q." In *Advances in Cryptology—CRYPTO'98*, pp. 318-326. Springer Berlin Heidelberg, 1998.

[44] Williams, H. (1980). A modification of the RSA public-key encryption procedure (Corresp.). *Information Theory, IEEE Transactions on*, *26*(6), 726-729.

[45] Xiao, Liangliang, Osbert Bastani, and I-Ling Yen. "An Efficient Homomorphic Encryption Protocol for Multi-User Systems." *IACR Cryptology ePrint Archive* 2012 (2012): 193.

[46] Yao, Andrew. "How to generate and exchange secrets." In *Foundations of Computer Science, 1986, 27th Annual Symposium on*, pp. 162-167. IEEE, 1986.

[47] Yao, Andrew C. "Protocols for secure computations." In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160-164. IEEE, 1982.

[48] Zahari Mahad, and Muhammad Rezal Kamel Ariffin. " Rabin-RZ: A New Efficient Method to Overcome Rabin Cryptosystem Decryption Failure Problem" *International Journal of Cryptology Research* 5, no. 1 (2015): 11-20.