The Islamic University of Gaza

Deanship of Research and Graduate Studies

Faculty of Information Technology

Master of Information Technology

الجامعـــة الإســلاميـــة بغـــــزة

عمادة البحث العلمي والدراسات العليـا

كليـــة تكنـــولـــوجـيـــا المعلومـــات

ماجستير تكنولوجيـــا المعلـومـــات

A Model for Detecting Integrity Violation on Files Stored on Cloud with Error Localization and Correction

نموذج اكتشاف انتهاك سلامة تكاملية الملفات المحفوظة على تقنية السحابة مع تحديد موقع الخطأ و تصحيحه

By:

Eman Ahmed Kalloub

Supervised By:

Dr. Tawfiq Barhoom

Associate Professor – Applied Computer Technology

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Information Technology

March , 2019

إقـــــرار

أنا الموقعة أدناه مقدمة الرسالة التي تحمل العنوان:

A Model for Detecting Integrity Violation on Files Stored on Cloud with Error Localization and Correction

نموذج اكتشاف انتهاك سلامة تكاملية الملفات المحفوظة على تقنية السحابة مع تحديد موقع الخطأ و تصحيحه

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وأن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل الاخرين لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

Declaration

I understand the nature of plagiarism, and I am aware of the University's policy on this. The work provided in this thesis, unless otherwise referenced, is the researcher's own work and has not been submitted by others elsewhere for any other degree or qualification.

Student's name:	Eman A. A. Kalloub	اسم الطالبة:
Signature:	*Eman A. A. Kalloub*	التوقيع:
Date:	05/03/2019	التاريخ:

بسم الله الرحمن الرحيم

الجامعة الإسلامية بغزة
The Islamic University of Gaza
عمادة البحث العلمي والدراسات العليا
هاتف داخلي: 1150

الرقم ..... Ref ج س غ/35/
التاريخ ..... Date 2019/04/10م

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة عمادة البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحثة/ **ايمان أحمد عبد الفتاح كلوب** لنيل درجة الماجستير في كلية **تكنولوجيا المعلومات**/ برنامج **تكنولوجيا المعلومات** وموضوعها:

نموذج اكتشاف انتهاك سلامة تكاملية الملفات المحفوظة على تقنية السحابة مع تحديد موقع الخطأ وتصحيحه

A Model for Detecting Integrity Violation on Files Stored on Cloud with Error Localization and Correction

وبعد المناقشة التي تمت اليوم **السبت 30 رجب 1440**هـ الموافق **2019/04/06**م الساعة **الحادية عشرة صباحاً**، في **قاعة اجتماعات كلية تكنولوجيا المعلومات** اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. توفيق سليمان برهوم	**مشرفاً ورئيساً**	.....
د. وائل فكري السراج	**مناقشاً داخلياً**	.....
د. أحمد فؤاد عبد العال	**مناقشاً خارجياً**	.....

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في **كلية تكنولوجيا المعلومات/برنامج تكنولوجيا المعلومات**.

واللجنة إذ تمنحها هذه الدرجة فإنها توصيها بتقوى الله تعالى ولزوم طاعته وأن تسخر علمها في خدمة دينها ووطنها.

والله ولي التوفيق،،،

عميد البحث العلمي والدراسات العليا

أ. د. مازن إسماعيل هنية

التاريخ: 21/ 4 / 2019م | الرقم العام للنسخة 3107414 | اللغة ع | ☑ ماجستير ☐ دكتوراه

الموضوع/ استلام النسخة الإلكترونية لرسالة علمية

قامت إدارة المكتبات بالجامعة الإسلامية باستلام **النسخة الإلكترونية** من رسالة

للطالب/ إيمان أحمد عبد العظاح كحيل

رقم جامعي: 220194090 قسم: تكنولوجيا المعلومات كلية: تكنولوجيا المعلومات

وتم الاطلاع عليها، ومطابقتها بالنسخة الورقية للرسالة نفسها، ضمن المحددات المبينة أدناه:

- تم إجراء جميع التعديلات التي طلبتها لجنة المناقشة.
- تم توقيع المشرف/المشرفين على النسخة الورقية لاعتمادها كنسخة معدلة ونهائية.
- تم وضع ختم "عمادة الدراسات العليا" على النسخة الورقية لاعتماد توقيع المشرف/المشرفين.
- وجود جميع فصول الرسالة مجمَّعة في ملف (WORD) وآخر (PDF).
- وجود فهرس الرسالة، والملخصين باللغتين العربية والإنجليزية بملفات منفصلة (PDF +WORD)
- تطابق النص في كل صفحة ورقية مع النص في كل صفحة تقابلها في الصفحات الإلكترونية.
- تطابق التنسيق في جميع الصفحات (نوع وحجم الخط) بين النسخة الورقية والإلكترونية.

ملاحظة: ستقوم إدارة المكتبات بنشر هذه الرسالة كاملة بصيغة (PDF) على موقع المكتبة الإلكتروني.

١٩٥

والله ولي التوفيق،

توقيع الطالب ..... إدارة المكتبة المركزية .....

Abstract

Cloud computing, as a pool of configurable resources virtualized as services over the internet, spread widely between almost all the technologies available. One aspect was using cloud servers as a storage system for storing and managing data files. Sharing stored files between multiple users exposes the file to both authorized and unauthorized alterations. Unauthorized alteration on these files may cause integrity violation and errors in those files.

Many integrity violation detection models were proposed. Among several techniques used for integrity violation detection, hashing took quite a good part. Whole file hashing and partial file hashing techniques were proposed but neither performed error localization nor error correction.

The model of this thesis adds a contribution to previous works. Thus the model objects to not only detect integrity violation but to localize and correct file integrity violation. The model requires the user to subject his file to pre-processing before outsourcing it to cloud servers. The metadata resulting from pre-processing takes the shape of row and column hash values that will be used as a material at integrity check request. Using files saved metadata, the model shall be able to detect integrity violation, localize the indexes of violated characters, and correct the localized violations. Several experiments for testing the model were conducted and applied on files of different sizes.

The evaluation of the proposed method was based on the success and accuracy of integrity violation detection and whether the violation was localized and corrected. The size of the metadata needed to perform the check process was also evaluated. Results show that the model was executed within 0.4723 µs average execution time. Also, the size of metadata was about 0.944 from the original file size when the original file size was bigger than or equal to 40KB.

***Keywords*:** Integrity Detection , Cloud Storage, File Sharing, Violation, Error , Violation Localization, Violation Correction, Accuracy, Metadata , Execution Time.

ملخص الدراسة

مع ظهور الحوسبة السحابية و انتشار خدماتها المتمثلة في الموارد التكنولوجية الافتراضية عبر الانترنت كان اهم استخداماتها كنظام لتخزين الملفات و مشاركتها بين المستخدمين. قد تؤدي مشاركة الملفات بين المستخدمين الى تعرضها للتعديلات المصرح بها و غير المصرح بها مما قد يؤدي الى انتهاك سلامتها وتكامليتها وحدوث اخطاء فيها. لهذا السبب قدم العديد من الباحثين نماذج للكشف عن سلامة الملفات المشاركة عبر السحابة. File Hashing كان اهم الطرق المستخدمة سواء كان hash لكل لمحتوى الملف او جزئي لبعض محتويات الملف. الهدف الاساسي للأبحاث السابقة كان الاقرار بسلامة الملف من الاخطاء او الاقرار بانه تم انتهاك سلامة الملف لكنها لم تحاول تحديد موقع الخطأ او تصحيحه .

في هذا البحث العلمي نقدم نموذج قادر على الكشف عن الملفات المنتهكة بالإضافة إلى قدرته على تحديد مكان الخطأ و تصحيحه. استخدام النموذج المقدم بالبحث يتطلب من المستخدم أن يحفظ بيانات عن الملف قبل مشاركته على السحابة. هذه البيانات عبارة عن مجموعة من الhash values لصفوف و أعمدة محتوى الملف ليتم استخدامها في مرحلة تحديد مكان الخطأ (احداثيات الخطأ) و تصحيحه.

تم اجراء العديد من التجارب لاختبار دقة و صحة هذا النموذج. بالإضافة إلى قياس حجم البيانات (metadata) اللازم حفظها على قاعدة بيانات خارجية. و تظهر النتائج أن النموذج كان قادرا على تحديد مكان الخطأ و تصحيحه في وقت متوسطه الحسابي 0.4723 (µS)، في حين أن حجم البيانات اللازم حفظها (metadata) يساوي 0.944 من حجم الملف الأصلي عندما يكون حجم الملف الأصلي أكبر من 40 كيلو بايت .(KB)

﴿ يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ﴾

[المجادلة : 11]

Dedication

To my guardian angel , my Father

To the unconditional love of my life, my Husband

To my beloved Mother and Sisters

Acknowledgment

First of all , I thank Allah for all the strength he gave me to finish this work , despite all of the hardship I've faced. I also thank my priceless family. Special thanks to my father, my mother, my husband and my beloved sisters for their enormous encouragement and support.

My thanks and gratitude to my supervisor : Dr. Tawfiq Barhoom for his support and continuous guidance during this research. Without his detailed supervision, this thesis would not have been possible.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

EOL	End of Line
EOF	End of File
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
IaaS	Infrastructure As A Service
KB	Kilo Bytes
LRC	Longitudinal Redundancy Check
MAC	Message Authentication Code Algorithm
MD5	Message Digest Algorithm
µs	Microseconds
NIST	National Institute of Standards and Technology
PaaS	Platform As A Service
PHP	Hypertext Preprocessor
POR	Proof of Retrievability
POS	Proofs of Possession
SaaS	Software As A Service
SHA	Secure Hash Algorithm
SQL	Structured Query Language
TRC	Transverse Redundancy Check

Chapter 1
Introduction

Chapter 1: Introduction

To be in the era of speed and dominance of the internet, sharing became easy and comprehensive. Cloud computing provided a complementary storage system to support the need of sharing for different users and facilities. With that came security challenges and many researchers contributed in providing solutions.

Specifically speaking, sharing files can cause integrity violation on file data. Unauthorized modification on files may cause violation of files integrity. For that reason, researchers proposed many solutions to state if the integrity of file has been violated or not. File hashing before sharing file on the cloud was a good method to detect the occurrence of violation of integrity on the file. A variety of methods and techniques regarding file hashing aimed for detecting integrity violation were also proposed.

In this thesis, hashing was used to detect integrity violation of shared files. Violation is defined as an unauthorized change of some of the file content. The model of the thesis aims to state if an integrity violation has occurred, localize the exact location (index) of violation and correct the violation.

In this chapter, an introduction of cloud as storage, file sharing and hashing as file security is briefly reviewed. Also, all the aspects of this thesis and how it's organized are explained.

1.1 Background and Context

With the huge advancement of technology, accessing data over the network via various computing devices comes the desire of storing data files on storage system. Storage system allows users to store and access their data from any device and from any location via a network. Cloud computing literally provided services and resources that keep up with the growing needs of organizations and individuals.

Cloud computing has been used as a popular business model where business computing resources are delivered as a utility on demand to customers over the internet (Aldossary & Allen, 2016; Leesakul, Townend, & Xu, 2014). Generally, cloud computing is defined from the point of view or according to the technology used for system development. In other words, cloud computing is defined as a business model that present computing resources as a service on demand to users and customers over the internet (Mell & Grance, 2011).

Although many storage systems exist the benefits of cloud as storage surpass them especially in scalability, portability and cost reduction (Rajathi & Saravanan, 2013). Google drive is an example of cloud storage that provides services that include saving files on the cloud and sharing files between multiple users.

Outsourcing sharable data files to the cloud allow many users to access this data. Some authorized users may commit unauthorized modifications. As well as valuable data stored on the cloud are vulnerable to unauthorized access and alteration since the data is handled by external parties. This lead to many researches and measures to detect security breaches of data stored in the cloud including integrity violation (Anil & Thanka, 2013).

Because an online storage system like the cloud is used to store important files with user's alteration allowed it’s important to protect the integrity of files and detect any integrity violation. Moreover, untrustworthy cloud servers make users think twice before saving important data on the cloud (Rong, Nguyen, & Jaatun, 2013).

Many studies were conducted with the purpose of protecting the integrity of files stored on the cloud and detecting if a violation has occurred. Recently, a study surveyed most of those techniques in details (Zafar et al., 2017). Most of the previous researches conducted integrity check on shared files. They mostly aimed to check if the integrity of files has been violated or not. Using different hashing functions and schemes, their focus did not slide anywhere near localizing the violation or correcting it.

With that in mind, a model for detecting integrity violation on shared files is proposed. The highlight of the proposed model is that it will conduct violation localization and violation correction. The model will consist of three modules: First, file processing is used to produce metadata from file content. Second, Violation localizations module is called to localize indexes of violations. Third, Violation correction module is called to correct violations at localized indexes. Metadata of the file is mostly a set of hash values. That's why the violation location is identified by the location of the changed hash value.

The three modules of the proposed model will be explained in details in chapter three. The model gives an assurance to the file owners. This is demonstrated by presenting a process to recover the original file content and provide the file owner with a downloadable version. This last step is provided as an extra confirmation of the models efficiency.

1.2 Statement of Problem

Outsourcing files to cloud servers move control over files from file owner to cloud servers. To assure file owners about the integrity of their outsourced files, many integrity violation models were proposed. It's true that previous studies detected integrity violation, but none of them approached a way to localize and correct the violations.

1.3 Objectives

1.3.1 Main Objective

The main objective of this thesis is to develop an integrity violation detection model empowered with the ability to localize and correct the detected violations. The model will be applied on text file.

1.3.2 Specific Objectives

1. Collect a set of files with different sizes.
2. Select a secure and collision resistant hash function.
3. Design the model to consist of three modules: File processing, Violation localization and Violation correction.

4. Design the model's client interface to save file data and send check requests.
5. Implement the proposed model to meet its main objective.
6. Relieve the model user by providing a downloadable version of the restored file content.
7. Evaluate the efficiency of the model in terms of accuracy, execution time and metadata size.
8. Present two optimization models and conclude the enhancements and the drawbacks.
9. Evaluate and discuss the three models and conduct comparisons.

1.4 Scope and Limitations

The proposed model is about developing techniques to be taken after reporting a file's integrity as violated. File processing constructs a row-column matrix from file contents. Rows will lead to a list of row hash values, so will the columns. The two lists will be compared with another two lists computed at the request of integrity check. The comparison defines the violated rows and columns. Localization works on the violated rows and columns to locate the violated characters indexes. Correction aims to correct the localized violated characters.

1. The model is applied only on text files and did not consider tabular data.
2. The files size used on testing did not exceed two megabyte.
3. The model steps and runs a small integrity check process by comparing hash values computed for whole file before and after sharing file on the cloud. This process avoids unnecessary execution of model modules when the file is actually integral.
4. The procedures taken after detecting violation are : File processing, Violation Localization and Violation Correction.
5. Integrity was the only security challenge of public cloud storage discussed on this work.
6. Correcting the violation is done on the localized violated indexes.
7. As limitation :

a. The research is only empirical and not comparative that's because other researches did not perform violation localization or correction.
b. Violation correction for each violation is guaranteed on a condition , violated row or column was caused due to one character change only.
c. If the row or column has multiple character violations, the corresponding columns or rows respectively must have only on character change.

1.6 Signification

Sharing files between users on storage systems controlled by other parties than the file owner can be a stressful concern to file owner. That's why many researchers provided ways to check if a shared file is still integral. Hashing was used as means for checking integrity and relived file owners about their files integrity. The proposed model gives extra assurance and develops a way for detecting the violation location. Not only that, but it will also correct those violations and try to recover the file original content. If integrity is guaranteed back, that's a whole new level of files integrity security.

1.8 Overview of Thesis

Chapter Two discusses previous studies regarding using hashing as a technique to detect integrity violations of files. Whole file hashing and partial file hashing are discussed. It also presents the cloud system and briefly discusses cloud as storage.

Chapter Three explains all three modules of the proposed approach. The File Processing Module, Violation Localization Module and Violation Correction Module are explained in details. Flowcharts and pseudo code are used to demonstrate the used methods and techniques.

Chapter Four illustrates experiments used through testing the proposed model and discusses the results extensively to highlight the model accuracy.

Chapter Five presents two optimization models of the original model. It also holds experiments and discusses results.

Chapter Six concludes the research contributions and conclusions. Also presents plans for future work.

Chapter 2
Literature Review

Chapter 2: Literature Review

Using cloud services deployed as a storage system became trendy between organizations and independent file owners. With trends, comes many security challenges. Special mention On this thesis to the integrity of shared files. Researchers worked incrementally to provide solutions regarding detecting integrity violations. File hashing techniques contributed big time among provided solutions. On this chapter, file sharing using a cloud storage system is discussed as well as the role of hashing in detecting integrity violation.

This chapter is divided into two sections:

- Section one discusses the related concepts for this research.
- Section two discusses the related works for this research.

2.1 Related Concepts

This section introduces the technological concepts mentioned in this research. Each related concept is explained briefly.

2.1.1 Cloud Computing

The National Institute of Standards and Technology(NIST) (Armbrust et al., 2009) characterized cloud computing as "… a pay-per-use model for enabling available , convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks , servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. "

Although there are many definitions for cloud computing, the ultimate goal noticed is to allow customers to run their everyday IT infrastructure in the cloud (Voorsluys, Broberg, & Buyya, 2011). Figure 2-1 illustrates cloud computing service types with examples.

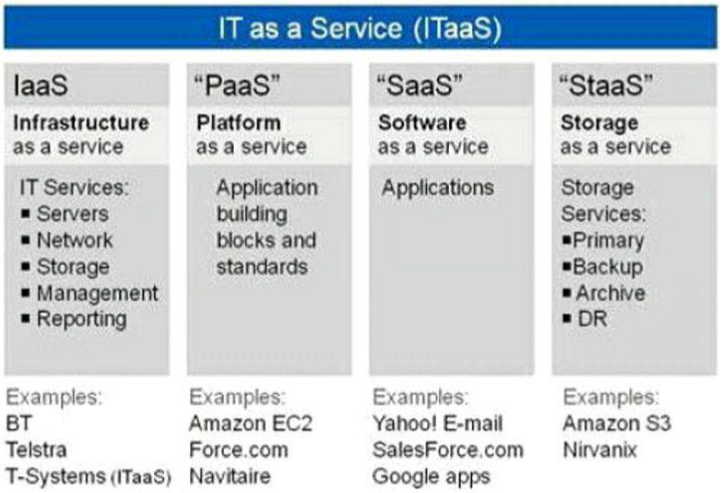


Figure 2. 1: Cloud Computing Service Types with Examples

(Voorsluys et al., 2011)

The service models on Figure 2-1 are explained as follows :

1. **Software as a Service (SaaS):**

That's where the cloud provides online software to users. The software is hosted on a cloud infrastructure where the user has no control over it. Using a thin client interface such as a web browser, users can benefit from the software (Mell & Grance, 2011).

2. **Platform as a Service (PaaS):**

This where the consumer can deploy an application to a cloud infrastructure. The infrastructure chosen to host the application usually supports the programming language, services, and tools needed for the program. The consumer can only control his deployed application configuration (Mell & Grance, 2011).

3. **Infrastructure as a Service (IaaS):**

This is where the consumer is given a limited control on the resources environment he used to run or deploy a software. Resources such as storage, operating system and, network components (Mell & Grance, 2011).

2.1.2 Cloud Storage

One of the main uses of the cloud is for data storage where data is stored on multiple third-party servers. The idea is that data is stored on a virtual server that does not exist in reality but an alias used to reference virtual space (Wu, Ping, Ge, Wang, & Fu, 2010). Financially speaking, cloud storage are typically cheaper than physical storage. Also from a security point of view, data stored on the cloud is secure from hardware crashes and accidental erasure. So by storing data on the cloud, data is easily accessed and often at lower cost.

Cloud provides storage services such as Google Drive, Dropbox and Microsoft SkyDrive. Those services are popularly used for file backup and data archival. Due to their ease of use, high scalability and accessibility (Bessani et al., 2014). There are three main models of cloud storage:

1. **Public Cloud Storage:**

Cloud storage is considered public when cloud resources are available to public users over the internet and none of these resources is stored in the company's data center. Meaning that the cloud storage service provider and the company data center are separate. Examples of public clouds include Amazon Elastic Compute Cloud (EC2), IBM’s Blue Cloud, Sun Cloud, Google AppEngine and Windows Azure Services Platform (Venkatesh & Eastaff, 2018).

2. **Private Cloud Storage:**

When the data center is owned by a single company and the cloud storage infrastructure is integrated with the company's data center, It's called a private cloud. It's managed by the cloud storage and maintenance control is given to the company. It's typically used by large enterprises (Venkatesh & Eastaff, 2018).

3. **Hybrid Cloud Storage:**

Is a combination of public cloud storage and private cloud storage where data meant to be accessible publically are stored in the public cloud while critical data are stored in the company's private cloud (Venkatesh & Eastaff, 2018).

2.1.3 Cloud Security

Cloud security reserved a good cut in the research field. As cloud computing has been adopted by many technologies it became important to provide an appropriate solution to face arising security issues. A book (Samarati, di Vimercati, Murugesan, & Bojanova, 2016) discussed the mechanisms of cloud security regardless of model and deployment. Network, data and application security were presented.

A study discussed the security breaches cloud services and cloud data exposes to. The study discussed all logical security aspects regarding confidentiality, integrity, and availability of data (Rao & Selvamani, 2015). They suggested that encrypting the data before storing it to the cloud server was a good solution to integrity issues. Hashing file before uploading it to the cloud and using the hash later to make sure the data on the file is not altered and integrity is maintained was also suggested.

2.1.4 Hashing Process

Hashing is the process of using some hash function taking data of some size as an input and outputting a hash value of a fixed-length (Chi & Zhu, 2017). Many hash functions and methods were proposed. When talking about secure hashing, Cryptographic secure hashing not only produces a fixed-length hash value but also secure and irreversible (Chi & Zhu, 2017). MD and SHA families are types of un-keyed cryptographic hash functions.

Example of SHAs family, SHA256 hash function provides a unique 256-bit value of the file which ensures that no two different files will produce the same hash value (Borshack, Thomas, Einav, & Taron, 2016). SHA256 will be used in this thesis. An example of a keyed hashing algorithm is Message Authentication Code (MAC). The key is used in producing a message digest. The message digest is produced using a hash function for the appended to the message. A comparison is held to check integrity (Sodhi, Gaba, & Technology, 2018). Figure 2-2 shows different secure hash algorithms of the SHA family and their properties (Dang, 2015).

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Figure 2. 2: Different Secure Hashing Algorithms

(Dang, 2015)

2.1.5 Integrity Violation Detection

Data integrity protection means keeping data safe from unauthorized modification. Remote cloud servers are used to store outsourced data, but these servers might be insecure and unreliable. The integrity of outsourced data becomes a concern due to loosing full physical control over the outsourced data (Aldossary & Allen, 2016). For those reasons keeping data integrity and detecting any integrity violation is a necessity. Normally data owner or a third party can perform an integrity check on the data (Aldossary & Allen, 2016). Many techniques and methods were conducted under the name of integrity violation detection. Some of these researches used the whole file to verify integrity and others verified it based on randomly chosen blocks of data (Zafar et al., 2017).

2.1.6 Error Control Codes

The idea of computing hash value for each row and each column of data was inspired from Longitudinal Redundancy Check (LRC) and Transverse Redundancy Check (TRC) Error correction code algorithms. LRC is a horizontal redundancy check applied to a stream of data bits or a message. LRC breaks the data into words of a fixed number of bits and computes XOR operation of the words together. The result of LRC is appended to the data word and sent to the receiver. The receiver then breaks the received message into words including the LRC appended value and performs XOR. If the result is all zeroes, then the integrity of the message is not violated and no error occurred. LRC can only detect if an error occurred but cannot correct that error. For the purpose of

correcting the error, TRC was combined with LRC method. TRC is like LRC but is a vertical redundancy check (Bawaneh, Alkoffash, Alqrainy, & Muaidi, 2016). Assuming that a message was divided into 8-bit words.

Table2. 1: TRC and LRC

	Message Words							LRC
	1	0	1	0	1	1	0	0
	0	1	1	1	0	1	1	1
	1	1	1	0	0	0	0	1
	0	0	0	0	1	1	1	1
TRC	0	0	1	1	0	1	0	

If a change occurred on some bit, that means that the value of the corresponding TRC and LRC has changed and the cross between them LRC and TRC localize the violated bit. The correction of that bit is by flipping the bit. The proposed model of this thesis performs Row hash (RH) instead of LRC and column hash (CH) instead of TRC. The model also works on bytes of data instead of bits. In each cell, there will be a character, not a bit. The cross between RH and CH will localize the byte with error or violation.

2.2 Related Works

In this section, the previous research's on detecting file integrity violation are discussed. The focus is on researches that used hashing as the main method for the detection of integrity violation.

2.2.1 File Hashing Applications

File hashing was used in many applications other than integrity violations. In this section, some of the applications are briefly presented.

2.2.1.1 Hash Based Carving

File hashing was used on hash-based carving (Garfinkel & McCarrin, 2015), which is a technique for finding matching file blocks on media storage. Rather than whole file hashing, carving was used to find blocks of the file that has been fragmented or modified. Using MD5 hashing algorithm, hash values for 4KiB file blocks stored on the database were compared with hash values for 4KiB sectors on storage media. The fact that MD5 is not collision resistant was not relevant, Regardless was used due to its speed.

2.2.1.2 File Backup System

Using file hash key, the file is checked if it already has a backup on the backup system (De Spiegeleer, 2010). Each file to be backed up must have a hash key. The file is backed up if it's hash key comparison against two hash key lists results with no existing backup. One is a local list and the other is central list.

2.2.1.3 Optimize File Reads

Reading a file from the hard disk may require multiple disk access. A research (Lensing, Meister, & Brinkmann, 2010) used file hash to possibly accessing small file read in one disk access. The approach computes the expected location of the file by applying a hash function on the file path.

2.2.2 Integrity Violation Detection Techniques

This section discusses the previous schemes that detected violation of integrity on files shared on the cloud. Schemes based on whole file hash and random block hash are also discussed.

2.2.2.1 Whole File Hash Integrity Check

Whole file hashing means a need to access the whole file content to determine its integrity. Usually, whole file hashing is more suitable for small sized files with a few megabytes (Han, Liu, Chen, & Gu, 2014).

One of the earliest schemes to check the integrity of files saved on the cloud was using Message authentication code algorithm (MAC). MAC algorithm runs on the client side, where the client computes the MAC for data file, outsources the file to the cloud and later downloads the whole file for integrity check. The client compares the MAC value computed after downloading the file with the one computed before outsourcing the file. Since calculating the MAC for the file takes a lot of time and consumes more bandwidths, the author suggested hash value computation instead (Aldossary & Allen, 2016). An ability to detect integrity violation was achieved but determining the violated content was not considered.

A research (Ora & Pal, 2015) presented a solution to maintain the integrity of data saved on the cloud. MD5 was used to hash the file and a copy of the hash value is sent to the data owner for verification purpose later. Similar to the thesis in hand method, the data is safe if the hash value of the data present on the cloud is matched with the hash value present at the owner end. Otherwise, integrity was violated. The study managed to detect integrity violation but no error localization was conducted.

Whole file hashing was used as an indicator of file integrity violation detection. A thesis (Lulu, 2016) presented a model to check the correctness of data saved on the cloud. SHA256 was used. The method retrieves the file from the cloud and conduct a whole file hashing and compares the result with a previous hash value stored on some

intermediate database. Error localization was not considered and only a single hash value was a result of whole file hash.

Another research(Luo & Bai, 2011) proposed a protocol for data integrity check of data saved on a remote server. The protocol is based on HLAs and RSA signature. The cloud server runs a generation proof to proof data storage correctness.

2.2.2.2 Random-Data Selection Based Integrity Check

The schemes proposed in this section focus on choosing random blocks of data to verify the integrity of the file. Usually, these methods are applied on large sized files (Ateniese, Di Pietro, Mancini, & Tsudik, 2008). The techniques explained in this section vary between Proofs of Retrievability (POR) and Proofs of Possession (POS). POR and POS focus on generating proofs for a storage provider to check the integrity of clients data without downloading data (Zhu, Hu, Ahn, Yu, & systems, 2012).

Proof of integrity schemes in which the customer can use to check the correctness of his data saved on the cloud. Researchers (Kumar & Saxena, 2011) proposed a data integrity proof scheme based on selecting random bits of some file saved on the cloud. This method contributed in minimizing the size of the proof and thus reducing the bandwidth consumption of the network. The method informs the client about the integrity of his data, but does not step to localize or correct the cause of integrity violation.

2.2.2.1 Proof of Retrievability (POR)

Proof of Retrievability (POR) technique was used as an integrity verification scheme in many researches. It's based on a cryptographic formula where the owner of data doesn't need a local copy of data in order to achieve integrity instead, its audited remotely(Xu & Chang, 2012).

2.2.2.1 Proof of Possession (POS)

POS are Proofs that proof if the server still possesses the original data. By which can be considered as means for remotely checking data integrity without data retrieval from the server.

Proof of Possession (Ateniese et al., 2007) was used to check if the data outsourced to a third party is still integral. The client uses the system to generate metadata for the file and tags within the modified file. Tag generation is based on using a hash function. Since hash values are collision resistant, so are the tags. Using a challenge generated by the user, the server accesses blocks of data randomly and generates a proof. Figure 2-3 shows the sequence of proof of position. This research checks for server possession of data, But does not perform localization or correction of the blocks that caused tags mismatch.

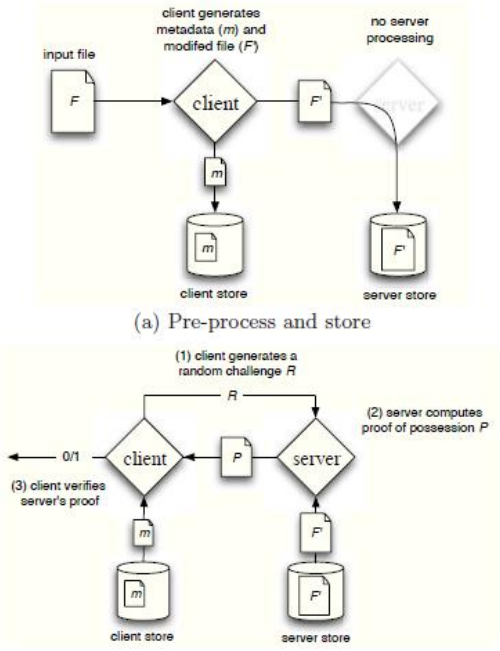


Figure 2. 3: Verify Server Position

(Ateniese et al., 2007)

The authors (Ateniese et al., 2008) proposed a scheme to proof data possession based on cryptographic hash function. But the lack of randomness on their system made it possible to deceive data owners (Zhu et al., 2012).

2.3 Summary:

This chapter discussed and explained the concepts related to this work. Also, a set of related works were discussed as summarized in table 2-1. Previous works as shown in Table 2-1 contributed heavily on the field of research regarding protecting the integrity of files stored on the cloud. Whether it was whole file hashing or partial file hashing, the aim was acknowledging if file integrity was preserved or have been violated. Means of localizing the violations and correcting them was not considered. The research in hand aims to develop an integrity detection model empowered with the ability to localize and correct violations. Table 2.2 summarizes the related works.

Table2.2 : Summarization of the related work

Paper	Methodology	Result	Shortcomings
Data Security and Integrity (Ora & Pal, 2015)	MD5 was used to hash the file and a copy of the hash value is sent to the data owner for verification purpose later.	Data owner verifies the integrity of the file.	The study managed to detect integrity violation but no error localization was conducted.
A Model to Detect the Integrity Violation (Lulu, 2016)	The method retrieves file from the cloud and conduct a whole file hashing and compares the result with a previous hash value stored on some intermediate database .SHA256 was used.	Experiments results display that the method was able to determine if a violation of integrity occurred.	Error localization was not considered and only a single hash value was a result of whole file hash.
Data integrity proofs in cloud storage(Kumar & Saxena, 2011)	Researchers proposed a data integrity proof scheme based on selecting random bits of some file saved on the cloud.	This method contributed in minimizing the size of the proof and thus reducing the bandwidth consumption of the	Error localization and correction was not considered.

		network.	
Proof-able Data Possession (Ateniese et al., 2007)	A proof is generated at client request and compared to meta data at client storage.	The method returns a probability of correct possession of data on the server. With minimum bandwidth consumption.	Statement of file integrity is reported. Not able to define the block that caused violation.

Chapter 3
Proposed Model

Chapter 3: Proposed Model

3.1 Overview

This chapter presents and discusses the proposed model of the thesis in hand. The research presents a model that checks for integrity violation on shared files. When the integrity of a file shared on the cloud is reported to be violated, a localization and correction for those violations are conducted. The research aims to state the localized violations and correct them. Localization and Correction processes are based on comparison of hash values computed for file before sharing on the cloud with ones computed when an integrity check is requested. The proposed model has the following modules:

1. File Processing, Violation Localization and Violation Correction are the main modules called upon the request of an integrity check. Executing the modules means the file was reported to be violated.
2. ***File Processing Module*** means constructing a row matrix from file contents where each row and column will lead to a hash value. The hash values are used to detect the violated rows or columns which will later lead to procedures to localize and correct the violated character. Three lists are a result of this module, one for row hash values , the second for column hash values and the last contains the characters occurred on file content.
3. ***Violation Localization Module*** attempts to specify the location or index of characters that caused integrity violation. The index is a combination of row number and column number. A change of hash value for the corresponding row or column puts the index in a list of localized violated indexes.
4. ***Violation Correction Module*** conducts a correction process on the list of localized indexes for the violated characters. Looping over a predefined list of characters, the character that leads the changed hash value of the row or column to match the original hash is the correct character.

The model gains its ability to localize the violated characters from the set of results produced from file processing module. Applying file processing on the file before and after outsourcing it to the cloud hands sets of before and after hash values to the localization module.

The localization module contributes on specifying and localizing the exact characters in the file that caused file integrity to be violated. This can be achieved by comparing every one of the corresponding hash values computed for file when file processing step was performed. Not only localizing, but correcting the violated character is also presented. The model also provides the user with a downloadable version of the file containing the corrected content.

3.2 System Architecture

As shown in Figure 3-1, the presented model consists mainly of three modules : file processing , violation localization and violation correction modules.

Figure 3. 1: Main modules

1- **File Processing:** process the file content into a row-column matrix which will be used on creating the row and column hash values for the file matrix.
2- **Violation Localization:** to localize the index of violation or error that lead to a change in some hash value.
3- **Violation Correction:** to correct the violations localized from the previous localization module.

The three modules are called upon two system procedures: Save file data and Check file integrity. To save file data only file processing module is called. However, all

the three modules are called when a request to check the integrity of shared file whose data was once saved on the system.

3.3 System Procedures

The proposed modules are executed during two main procedures: Save File Data and Check File Integrity Procedures. Figure 3.2 shows saving file data system procedure.

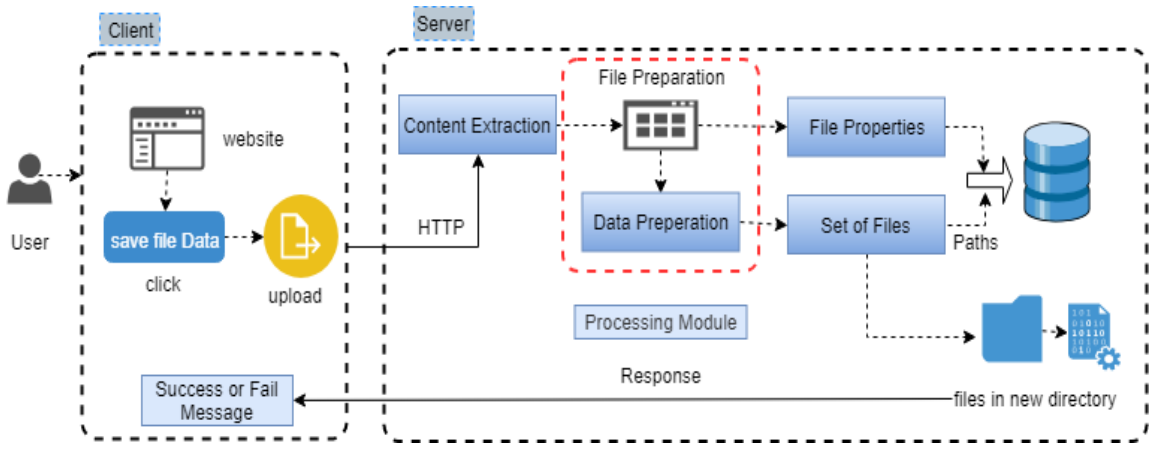


Figure 3. 2: Save File Data system procedure

3.3.1 Save File Data Procedure

On the server side , the system receives a file from the client, that represents the file to be processed before being shared on the cloud.

As shown on Figure 3-2, this procedure consists of three processes and the File Processing module. Each are explained as follows:

3.3.1.1 Content Extraction Process

To prepare the file for the processing module, content extraction process prepares file contents, number of lines and maximum characters in line. Contents of the file are extracted as lines and stored into array. The size of the array is the number of file lines . The maximum characters in line is the value of the line with the most characters.

3.3.1.2 Processing Module

After content extraction, the next step is to call the processing module. The processing module will be explained in details in section 3.4.1. (See Figure 3-4 and Figure 3-5 section 3.4.1). At the end of the processing module, the actual output is arrays of hash values. These arrays are interpreted according to the request of the user. At the request of saving data, the arrays are saved into files. File paths are stored into database and actual files are saved to a storage system. At the request of check file integrity, the arrays are dealt with as arrays. At the current system procedure, the arrays are file metadata to be used later when check integrity of file is requested.

3.3.1.3 File Properties Process

Represent the File Name , Number Of Lines , Maximum Line Length and File Hash. These properties are stored into the data base and play a major role when retrieved from data base at the check file integrity procedure.

1. **File Name Property:** the name of the file stored on database is unique. Every user must remember the name of the file when he requests integrity check.
2. **File Hash Property:** prevents unnecessary computation on the second file to be checked. Meaning that if the hash value for the file to be checked matches the one stored on database, Then your file is integral. Otherwise your file's integrity is violated and file processing, localization and correction modules will be called. SHA256 is used to create file hash value.
3. **Number Of Lines Property:** this property defines the limit on number of rows when creating row matrix as it equals number of lines in the file. Meaning that when dealing with the file to be checked for integrity, only this number of lines will be considered. Any lines out of this limit are ignored and discarded from the checking process.
4. **Maximum Line Length:** is the value of the line with the most characters. This property means that every line will be translated into a row of elements with the maximum line length property. Meaning number of columns in each row is identical for each line in file.

The importance of the last two properties appears when creating row matrix for the file to check. The file to check row matrix must have the same number of rows and columns as the row matrix for the saved file. Since the hash values lists results from row matrix. And since comparing two lists acquires they have the same length. It makes sense for the two matrices to have equal row and column numbers.

3.3.1.4 Set of Files Process

On this process, a directory under file name is created on a dedicated storage server. That directory will contain the two files that resulted from file processing module. These files content is retrieved when performing integrity check. Also the paths for those files are saved on the database. This process is called only on save data procedure. Note that no files will be created for the file to be checked for integrity. At the end of this procedure , a message with the success or fail will be returned to the user at the client side.

3.3.2 Check Integrity Procedure

Figure 3.3 shows Check Integrity System Procedure. This procedure calls all the three modules of the proposed approach. It also presents four processes. Each is explained in the following sub sections:

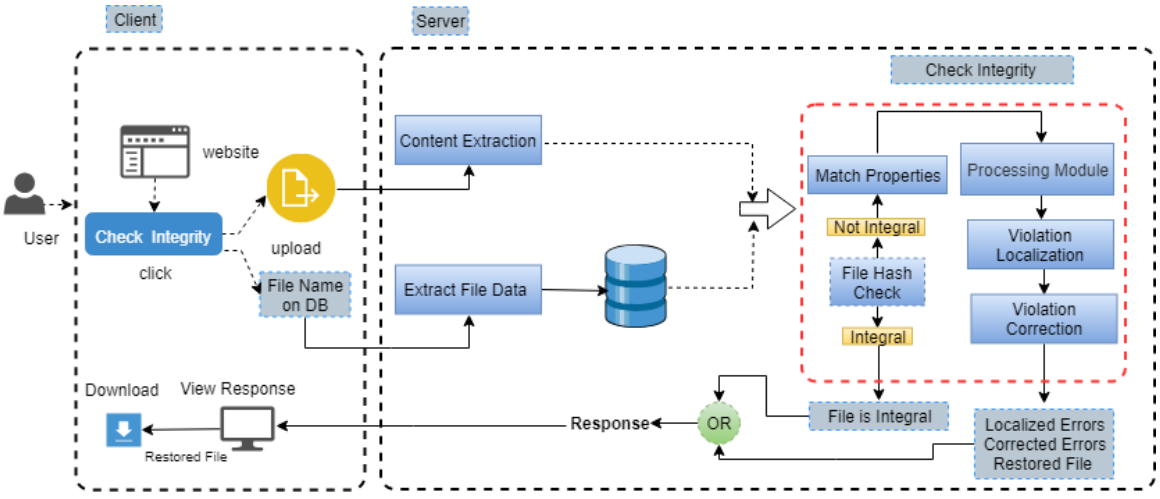


Figure 3. 3: Check File Integrity system procedure

3.3.2.1 Extract File Data

Based on the name of the file stored on data base given By the user, file data is retrieved from data base. The data includes file hash, number of lines and maximum line length. The extracted data is used by the file hash process and the match properties process.

3.3.2.2 File Hash Check

This process checks if the file is integral by comparing two hash values. The hash value computed for the content from content extraction process with the one retrieved from data base. When a match found, a message is returned to the client that the file is integral. Otherwise, the procedure continues to the match properties process.

3.3.2.3 Match Properties

Comparing two arrays requires that the two have the same size. And since arrays in the proposed approach are a result of some matrix. Then it makes sense for every two matrices to have the same number of rows and the same number of columns. With that in mind, matching properties equalizes number of rows by equalizing the number of lines for the file to be checked with number of lines from database. The same goes for columns, which is about equalizing the maximum line length property. For example, if the file to be checked has number of lines m more than the number of lines n for file saved on the data base, Only the first n of the m lines will be considered. The rest are ignored and the same philosophy goes when dealing with columns. Violation Localization and Violation Correction modules will be explained in section 3.4.2 and 3.4.3 .

3.4 Model Modules

The proposed model consists of three modules: File processing module, Violation localization module and Violation correction module.

3.4.1 File Processing Module

File processing is about giving files used by the proposed model a similar structure. Having similarity in structure leads to having similarity at the size or length of the set of results. The set of results are mainly arrays. And a known fact in the programming world, comparing two arrays means they should have the same length. The Processing module will lead to three results for each file processed by the processing module. These three results are : Row Hash Values, Column Hash Values and List of Characters. File processing is held twice for the file, once before sharing the file. And the other after sharing the file and requesting integrity check on the file. Note that List of characters is only needed before sharing the file. Processing the file goes through two stages explained as follows: Stage One , is about creating the row matrix from file content. Stage Two is about using the row matrix constructed at stage one in computing row hash values as well as column hash values.

3.4.1.1 File Processing Stages One

Stage one of file processing is about creating the row matrix from file contents. The process of creating a row matrix from the content of the file is considered as the seed for the rest of file processing module. Figure 3-4 shows the first stage of file processing in which the following steps are performed:

- Read the contents of the uploaded file as a set of lines.
- Initialize an empty row matrix, where
    - Number of rows equals the number of lines in the file.
    - Number of columns equals the length of the line with maximum number of characters.
- All rows in the matrix will have the same number of columns.
- Fill each row columns with the corresponding line characters.
- After reaching end of line , fill the remaining empty columns with an empty string.

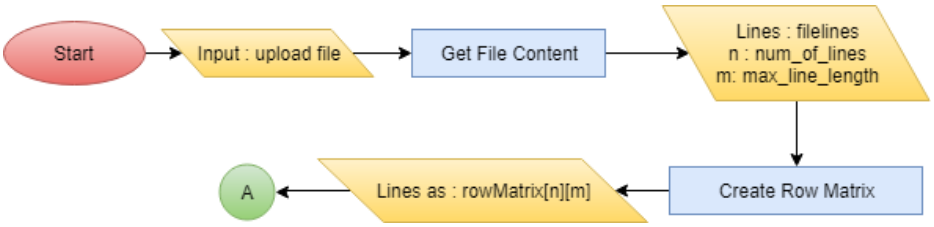


Figure 3. 4: First stage of file processing

This stage is about being able to index each character in the file by its row and column number. This will help in localizing the index of violation. That is since the violated character changes the hash value for the row and column it locates in. The pseudo code for creating the row matrix is shown right below.

- **Create Row Matrix**

Suppose a set of lines [L0 , L1 , L2 ,…, Ln] which represent the lines of a file uploaded by the user for processing, n is the number of lines, and m is the length of the line with maximum characters in line among the given lines :

1. First, define an empty matrix that will contain all the characters in the file. Every line will occupy a row in the matrix and all rows will have the same length. Also an empty character is defined.
2. Second , loop on every line in L and get line characters and put them in the current index of the row matrix. At the end of the characters in the line, fill the rest of the empty indexes at the current row with the defined empty string char.
3. After looping on all the lines, the row matrix is filled with the lines characters and returned to be used on stage two. Note that the rows in the row matrix define the lines and the number of columns is identical for every row.

Algorithm 1 Creating the row matrix from file content

1. **Input :** Array of file lines extracted from the uploaded file L[$\mathbf{L_0}$, $\mathbf{L_1}$, $\mathbf{L_2}$,…, $\mathbf{L_n}$] , and
   n : number of lines
   m : maximum characters in line
2. **Output :** Row matrix of size **n X m** where each element in the matrix is a character and each line allocates in an index n of the matrix.
3. x = 0
4. char = null
5. rowMatrix = array
6. **for**(r=0 to n) do
7. {
8. line = L[r]
9. end = size of(line)
10. **for**(c=0 to m) do
11. {
12. **if** (x < end) then
13. {
14. char = line[x]
15. x = x + 1
16. }
17. **end if**
18. rowMatrix[r][c] = char
19. }
20. **end for**
21. x = 0
22. }
23. **end for**
24. **return** rowMatrix

By the end of stage one of file processing, a row matrix that represents the file characters and empty character filling results. Stage two will use the resulting row matrix as explained in section (3.4.1.2).

3.4.1.2 File Processing Stages Two

The created row matrix is used in creating row hash list , column matrix which will lead to column hash list and list of characters array. As shown in Figure 3-5, Row hash is an array that contains the hash values for every row in the row matrix. Column hash is an array that contains the hash values for every column in the column matrix. Finally the arrays of hash values are saved into separate files on a dedicated server and the paths for those files will be saved on the database for use at the request of integrity check.

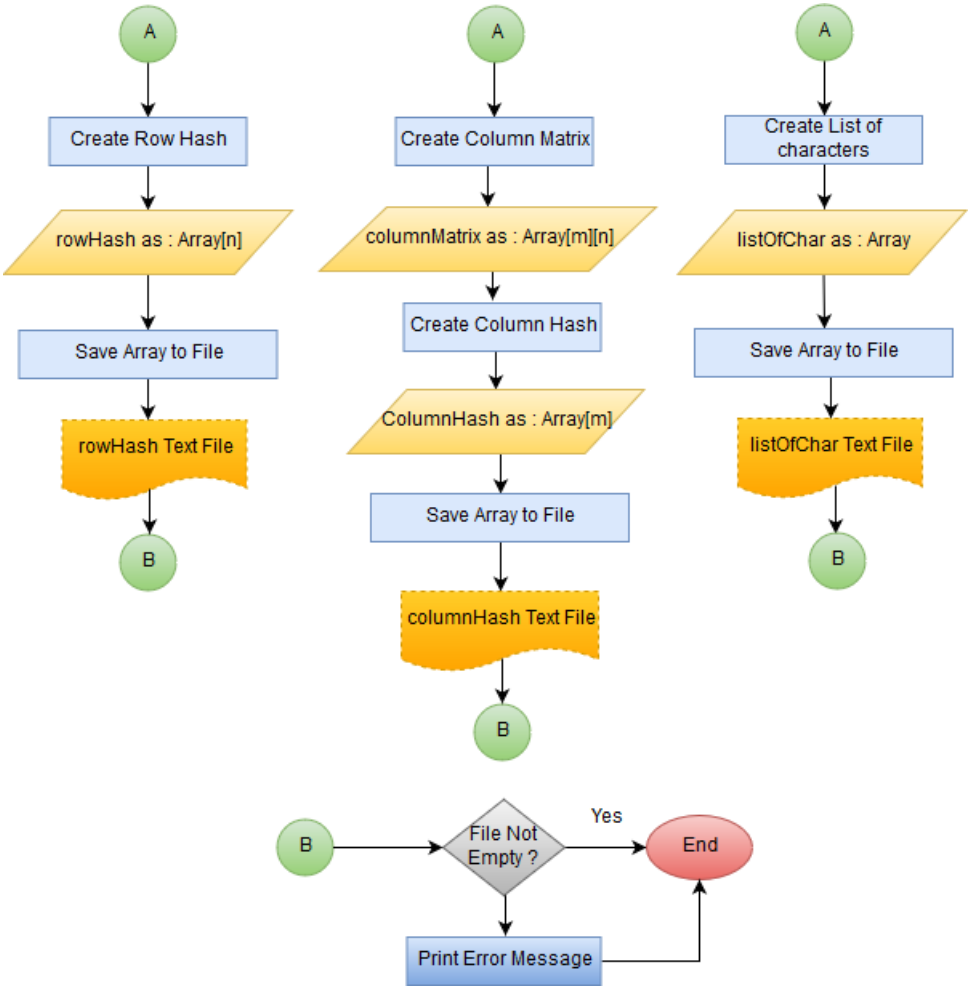


Figure 3. 5: Second Stage of File Processing

- **Creating the Column Matrix**

Creating the column matrix is simply the process of filling every index m X n with the element that exists at index n X m at the row matrix. By the end of this process, column matrix will have m columns. The number of column at the row matrix will equal the maximum line length among the file lines.

Algorithm 2 Creating column matrix

```
1.  Input    : row matrix of size n X m
2.  Output : column matrix of size m X n where each element in the matrix is a
    character and      each column allocates in an index m of the matrix.
3.  columnMatrix = array
4.  for(r=0 to  m) do
5.  {
6.        for(c=0 to n) do
7.        {
8.             columnMatrix[r][c] =  rowMatrix[c][r]
9.            }
10.          end for
11. }
12. end for
13. return columnMatrix
```

As the proposed approach depends on comparing hash values in order to detect integrity violation of file and to localize these violations. Choosing appropriate hash function and a secure one is taken into consideration. As shown in Figure 3-6 , array of element will lead to an array of hash values. SHA256 hash function is used.

Figure 3. 6: Apply hash function

Figure 3-5 also show creation of list of characters.. At the end of file processing module, every file must have three arrays :

1. **Row Hash Values :** Is defined as a list of hash values **RH [rh_0, rh_1 , rh_2,…, rh_n] ,** rh as in row hash, where each hash value is a result of applying some hash function on the content of every row inside the Row matrix **R [r_0 , r_1 , r_2 ,….., r_n],** r as in row**.** Row content is concatenated into a string and hashed using the chosen hash function. See Figure 3-7.

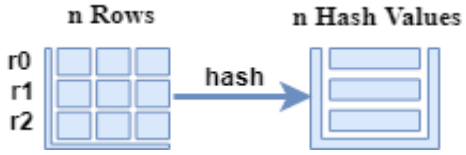


Figure 3. 7: Row hash values

2. Column Hash Values : Is defined as a list of hash values **CH [ch_0, ch_1 , ch_2, …. , ch_m]** where each hash value is a result of applying a hash function on the content of every column in Column matrix **C [c_0, c_1 ,c_2,…., c_m]**. Content of every column is concatenated into a string and hashed using the chosen hash function. See Figure 3-8.

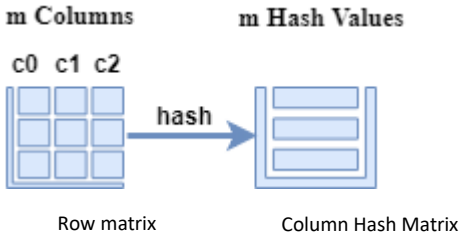


Figure 3. 8: Column hash values

2. **List of Characters:** That is defined as a list contains all the characters on the row matrix without duplication. Every character in that list is a potential correct character for the violated character. The list is retrieved from row matrix elements and is used at the violation correction module.

The process of creating files is only necessary when saving data for the file on the database and are extracted when an integrity check on that file is requested. Note that file processing is also called when performing integrity check. However, no files are created and checking is performed on the resulting arrays directly.

3.4.2 Violation Localization Module

By Violation localization, the intent is to determine the character that caused the original hash value for both row and column it locates on to change. Then the index of that character is the index that caused the violation. The localization module of this model is accurate under three assumptions:

a- First Assumption:

- The change in the row hash can be due to multiple character change but,
- The change in the column hash is only due to one character change.

b- Second Assumption:

- The change in the column hash can be due to multiple character change but,
- The change in the row hash is only due to one character change.

c- Third Assumption :

- The change in both row hash and column hash is due to one character change and,
- No two characters locate in the same row or column or in the same index.

The localization module is the module that results with a set of localized indexes of the violations that occurred on the file. These indexes are then passed to the correction module and dealt with so that all violations are cleared. In the proposed model, localization is done throw two stages: First, localizing the violated rows and columns. Second, localizing the violations indexes. Figures 3-9 and 3-10 show the two stages of localization.

3.4.2.1 Localization Stage one : Localize Violated Rows and Columns

Figure 3.9 shows the first stage of localization. The process takes four parameters as input: Two-row hash arrays and Two-column hash arrays. RH [rh1, rh2, rh3, …. ,

rhn] as for row hash values retrieved from database and RH* [rh*1, rh*2, rh*3, ….,rh*n] as for row hash values computed at the instance of check request. CH [ch0, ch1, ch2, …. , chm] as for column hash retrieved from database and CH* [ch*0, ch*1, ch*2, …. , ch*m] as for column hash values computed at the instance of check request.

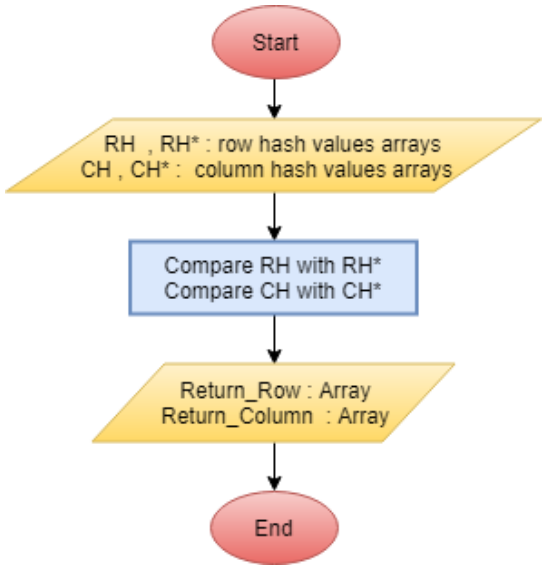


Figure 3. 9: Localize Violated Rows and Columns

The output of the process shown in Figure 3-13 is a result of applying a built-in function on every two corresponding arrays. The applied function compares RH and RH* elements and returns an array called Return_Row. And compares CH with CH* elements and returns an array called Return_Column. Whenever two elements on a given two arrays mismatch, their corresponding index is pushed and saved in a result array. Meaning that Return_Row array will contain row indexes that imply violated rows. And Return_Column array will contain column indexes that imply violated columns.

By the end of this process, the violated rows and columns are determined and all is needed is to determine the indexes of violations. The thought that comes to mind is to loop over the returned columns for every row and that is it. Meaning that every row index in the return row array is violated at every column index in the return column

array. That is absolutely correct when all of the violations have occurred in one row or in one column or only one violation is detected. Section 3.4.2.2 illustrates the second stage of localization: localizing violations indexes.

3.4.2.2 Localization Stage Two: Localizing the Violation indexes

Figure 3-10 shows the second stage in the localization module. To determine the violation indexes, a loop over the **Return_Row** and **Return_Column** arrays will get that done. The index is described as **(row , column)**, which is the location of the violation. For an index to take place as violated, it means that the hash value for the row or the column it belongs to has changed. In another say, it mismatched the row hash value retrieved from the saved file.

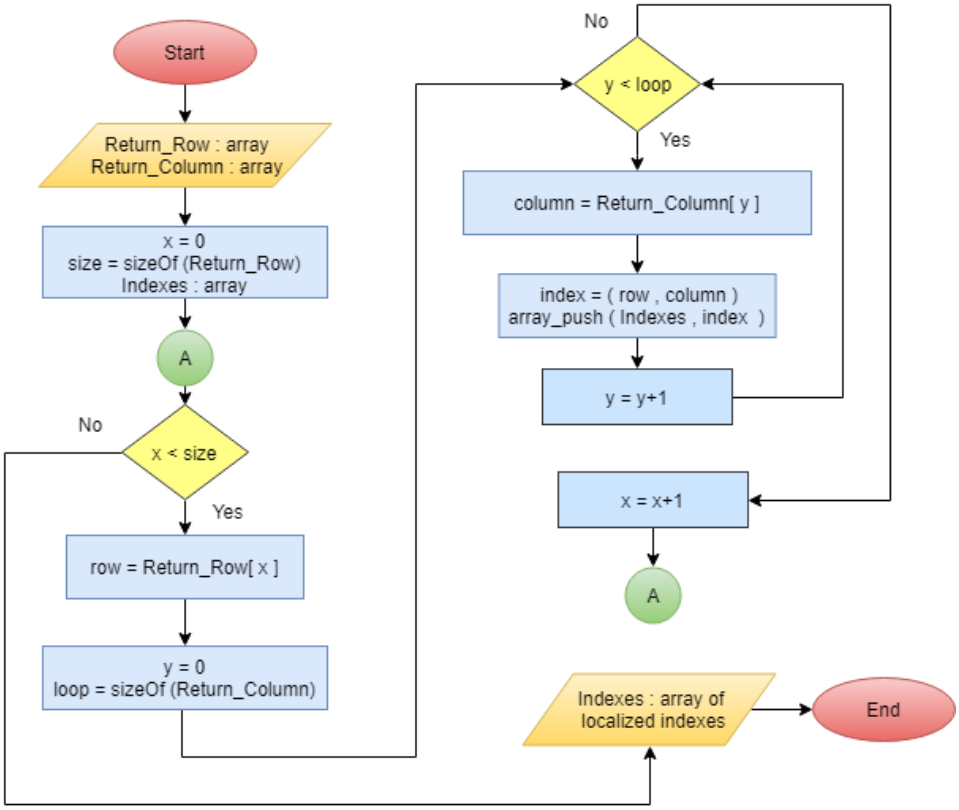


Figure 3. 10: Localizing the Violation Indexes

The second stage of localization will return an array that contains the localized indexes of violations.

Localized Indexes Array :

Implies all the indexes that were localized as violations. To be more specific, every row at return row array from stage one of localization may have a set of indexes. This set is the set of violated characters that will be corrected for that row. The indexes are a result of a cross between every value from the **Return_Row** with all the values from the **Return_Column**. Table 3-1 shows a small demonstration of the crossing.

Table 3. 1: Localization Indexes

Return_Row	Return_Column	Indexes		
2	**0**	**(2,0)**	**(2,1)**	**(2,2)**
3	**1**	**(3,0)**	**(3,1)**	**(3,2)**
5	**4**	**(5,0)**	**(5,1)**	**(5,2)**

By the end of the second stage of violation localization module, the array of localized indexes is passed to the violation correction module.

3.4.3 Violation Correction Module

Correction is actually about figuring out which character is the one that returns the row hash and column hash to its original hash value. Figure 3-11, Shows the steps taken at the Violation Correction Module.

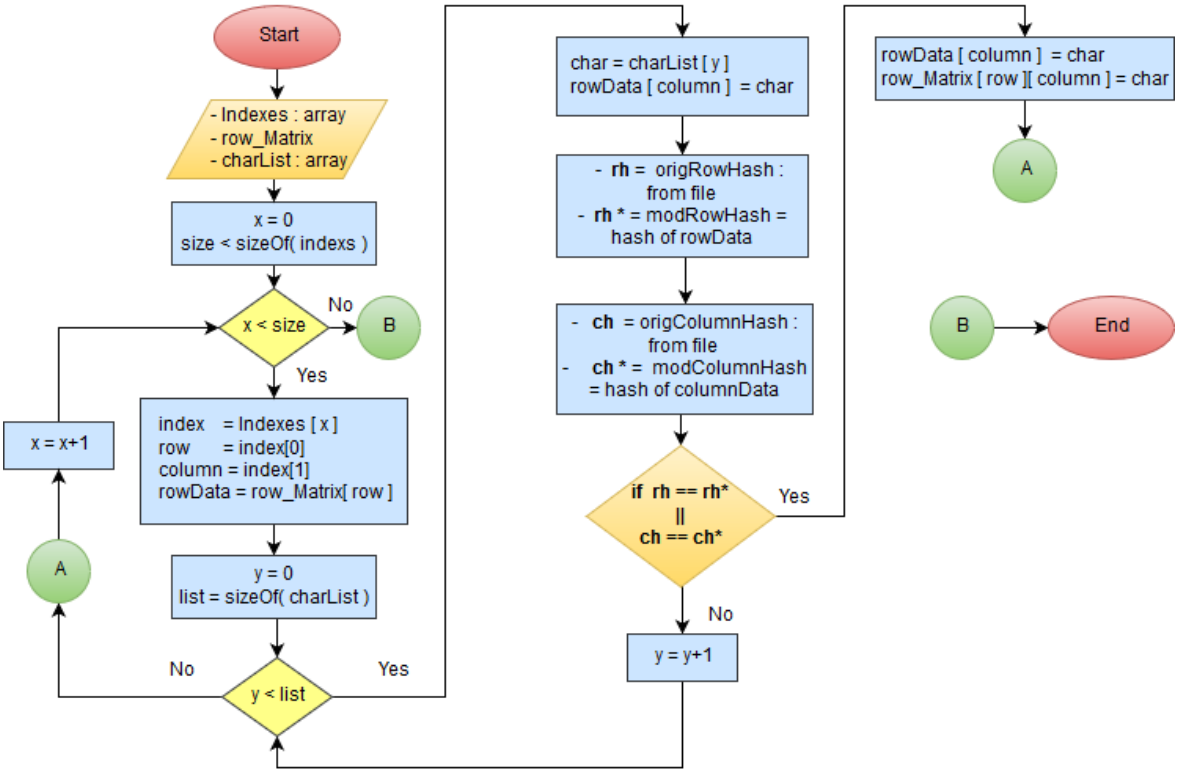


Figure 3. 11: Violation Correction Module

Figure 3.11 is **s**imply explained as follows:

For every localized index **x (rx , cx)** in indexes array:

Given :

- The violated character value at index x from row matrix : violated character **vc**.
- The correct row hash value **rh** for the index at row **rx** and the correct column hash value **ch** for the index at column **cx**.
- The list of characters that are possibly to correct the violation.

Do the following :

For every character at the list **Lc** :

- Replace the violated character at index **x** in the row matrix with **Lc**.
- Compute the new row hash **rh*** and column hash **ch***.

- If the hash computed **ch*** equals the correct hash **ch** or if the hash computed **rh*** equals the correct hash **rh** , then
- Then character **Lc** is the correction of the violated character **vc**.
- Modify the character at the row matrix index **x** , so that it becomes the character **Lc** instead of **vc**.
- Modify the character at the row matrix index **x** , so that it becomes the character **Lc** instead of **vc**.

By the end of the correction violation model, all the violations are corrected and the row matrix is modified with the correct characters.

3.4.4 Finishing Procedures

The finishing procedures of the model aim to give extra assurance to the user of the model about results accuracy.

The procedures include:

- Generate a new content from the modified row matrix and save to a downloadable file for the user.
- Check if the hash value of the downloadable file matches the original one on the database. If matched, display assurance message.

3.5 Model Interface

Figure 3-12 shows a snapshot of the model interfaces. Each interface is explained one at a time.

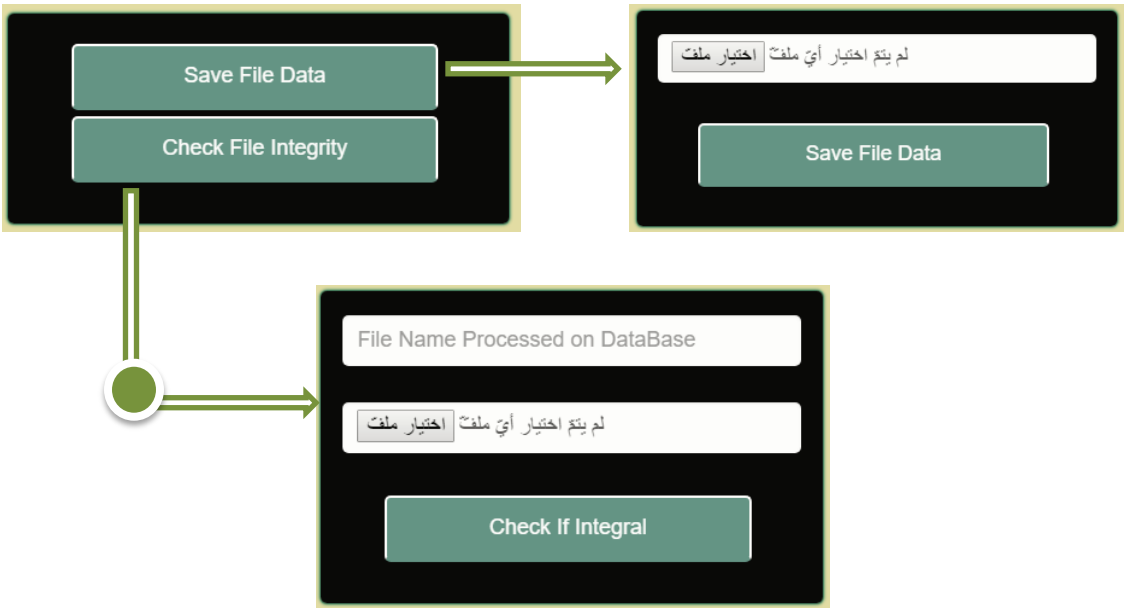


Figure 3. 12: Snapshot of the proposed system

Figure 3-12 shows three interfaces of the model. Starting from the left, the interface shows the home page of the system website. The second interface opens when the user clicks on the save file data button to process the file and save its data on the database. Finally, the last interface opens when the user clicks on the check file integrity button. File processing, localization and correction modules are called on this interface.

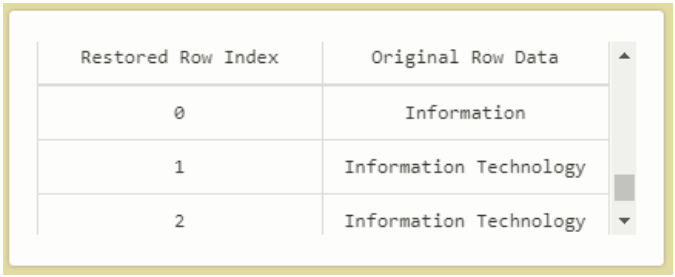
Figure 3-13 shows the expected result when the violation is localized and corrected. Each localized and corrected character is referred to by its row and column indexes.

Row Index	Column Index	Localized Violation
0	0	N
0	1	N

Row Index	Column Index	Corrected Violation
0	0	I
0	1	n

Figure 3. 13: Expected result snapshot

Figure 3-14 shows the original data restored for the violated row. Every row that was detected by the proposed approach to be violated, will be restored and feedback is given to the user.



Restored Row Index	Original Row Data
0	Information
1	Information Technology
2	Information Technology

Figure 3. 14: Restored violated row

A downloadable copy of the checked file will be available for the user. The given file is modified with corrected characters and provided for the user to download. Figure 3-15 shows a snapshot of the downloadable file.

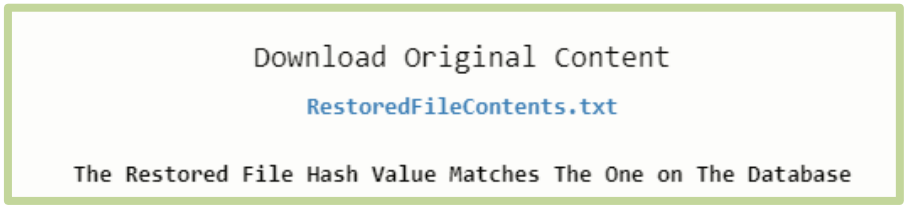


Figure 3. 15: Restored file

The message shown in Figure 3-15 is based on a test result of hash value comparison. The hash values are a result of the following steps :

- The row matrix constructed for the file to check is modified with every corrected character.
- After all the violations are corrected, the content of the row matrix is imploded as a string and hashed with a hash256 algorithm.

- The resulting hash value is compared with the original hash value for the file retrieved from the database. If they match, then the message **'The Restored File Hash Value Matches The One on The Database'** is displayed.
- The message is the assurance of the model effectiveness.

3.6 Summary

This chapter presented the proposed model of this thesis. The three modules composing the model were also presented. Also, the procedures and the interfaces of the model were discussed.

File processing module prepares file content as a row-column matrix in order to facilitate the indexing of each character in the file. This module will be used on the original file before outsourcing it to the cloud as well as the file needed to check for integrity. Each time it's used a set of hash values for file rows and file column are produced. Those will be used upon localizing the violations. Violation localization module is about comparing the row hash values arrays to detect the violated rows. The same will be done for the column hash values. This module results in a set of violated indexes. Violation correction module takes the list of violated indexes and replaces the violated character at the violated index with another character. The character that leads the violated row hash value to match the original one, is the correct character. This module will applied until all the violations are corrected. The next chapter shows the experiments held to test the model. It also clarifies the limitations on violations indexes precisely.

Chapter 4

Experiments and Results Evaluation

Chapter 4: Experiments and Evaluations

4.1 Overview

Building an integrity check model demands having metadata about the original data. The metadata is usually produced from a pre-processing procedures that are applied to the original data. Design challenges such as accuracy , the size of metadata and the execution time of the model are evaluated in this chapter.

Testing and evaluation of the proposed model are held in this chapter. The measures taken in evaluating the proposed model are discussed. The effectiveness and efficiency of the model will be evaluated by several experiments and result's discussion. Moreover, the testing of the model shall declare the limitation of the model precisely. Knowing the limits of this work will be beneficial in introducing enhancements and future ideas.

4.2 Evaluation Objective

The objective of the evaluation is to test the accuracy of model results and discuss other evaluation aspects such as computation efficiency and storage efficiency. The accuracy of the model is evaluated with the models objective in mind. If the model is proved by the results and testing to be able to detect, localize and correct all of the violations. Then the model is considered to be highly accurate.

4.3 Aspects of Evaluation

Evaluating the model depends on assessing how much the designed model meets its objective. As mentioned throughout the thesis so far, the proposed model aims to achieve three main goals:

- Check if the integrity of an outsourced file has been violated or not.
- Localize the source of violation or in other words the index of violated character.
- Correct all the localized violations if possible.

The three goals of the model are evaluated within the following aspects:

4.3.1 Accuracy

The accuracy of the model is considered high, if:

- The model can localize the indexes of violations.
- The model can correct localized violations.

As evidence of the accuracy of the results, an assurance message will be displayed to the model user. The message means that the hash value of the restored file with the corrected localized violation matches the original one that was once saved on the database.

4.3.2 Computation Efficiency

Computation efficiency is about the computation time it takes the proposed model to execute its tasks. Since the proposed model introduces three modules, the time executing every module has an effect on the computation time of the model. The factors that affect the computation time of the proposed model are :

1- Save File Data time which includes :

- Time for processing the file before outsourcing to the cloud, which is: Time for generating metadata about the original data which are row hash file and column hash file and list of characters file.

2- Check File Integrity time which includes :

- Time for processing the file to check.
- Time to Retrieve the data saved on the database.
- Time to localize the violations.
- Time to correct the violation.

Since the time to check file integrity calls all the three modules of the model, it shall be measured and evaluated. However, the time to save file data calls one module, thus time is predicted to be small. So the time it takes the model to check file integrity and respond to the user matters the most.

4.3.3 Computation Storage

For the proposed model to be able to perform integrity check at user request, the system or the model must have some metadata. The metadata is data about the original data that needs to be stored on a storage system. In this model, the metadata is:

- Row Hash values file.
- Column Hash values file.
- List of Characters file.

The size of metadata is supposed to be smaller than the size of the original data. Otherwise, the need for extra storage for the metadata other than the one for the original data can be an issue. Storing hash values inside files tend to take a lot of storage and can be a problem in the model. Tackling this issue and why it was considered an issue, will be discussed throughout results and discussions.

4.4 List of Experiments

Several experiments were conducted as the key factors to clarify the model's ability to represent its objectives in full. The experiments focused on localizing and correcting violations on four experiments. Also, they highlight the limits of the model regarding violation index :

Table 4. 1: List of Experiments Applied to Test the Model

Experiment Number	Experiment Title	Experiment Description
Experiment #1	One violation on file	File has one character change
Experiment #2	Multiple violations on one row only	File has multiple character change on one row only.
Experiment #3	Multiple violations on one column only	File has multiple character change on one column only
Experiment #4	Multiple violations on Distinct locations	File has multiple violations where the violations do not locate in the same row or column

The specified four experiments were chosen because they can guarantee that every localized violation is corrected. They are also applied under the condition that :

- A change of the file content is actually a replacement process. Meaning that all the violations that resulted from some original character replacement will be handled as a violation on its own. However, if a character was added or deleted, then the characters behind it that locate on the same row will be shifted. Thus, that character and the shifted characters will all be considered as violations.

4.4.1 Experimental Environment

This section describes the experimental environment used when testing the model was held. It also explains in details the four experiments described in Table 4.1.

4.4.1.1 Test Environment Properties

The model testing was held on a local machine laptop (DELL). With Windows 10 x64 operating system, Core i7 processor and 6 GB RAM.

PHP 5 was used as a programing language.

4.4.1.2 Data Set

To evaluate the model, a set of files of different sizes were collected manually and some downloaded from internet websites. The content of files was English characters and a variety of special characters and numbers. Only the text file type was used to test the model. Figure 4-1 shows files classification according to size.

- Small (5 – 100 KB)
- Medium (100 KB – 1 MB)
- Large (1 – 16 MB)
- Giant (> 128 MB)

Figure 4. 1: File Size Classification

(Lulu, 2016).

Based on the classification in Figure 4-1 :

1. 8 small files, 7 medium files, 2 large files were used to test this model.

2. SHA256 hashing algorithm was used to create hash values.
3. Empty lines are filtered from file data.

4.4.1.3 Model Experiments

This section explains in details the four experiments on Table 4-1.

4.4.3.1 Experiment #1: One violation on file

The first experiment was conducted on a file with one violation only. Figure 4-2 shows File original content and highlights the violated character. It also shows the violated content.

Original Content	Violated Content
1 EmanAhmed	1 EmanXhmed

Figure 4. 2: Experiment #1-original and violated content

Based on the data saved for the original content, the model will detect this file as violated and return the localized and corrected violations as shown in Figure 4-3. The figure also shows the time it took the model to execute the three modules at the integrity check request.

Violation Number	Row Index	Column Index	Localized Violation
0	0	4	X

Correction Number	Row Index	Column Index	Corrected Violation	Corrected Row
0	0	4	A	EmanAhmed
ExecutionTime Online =	0.0034630298614502			

Figure 4. 3: Experiment #1- Localization and Correction

Figure 4-3 is explained as in Table 4.2 :

Table 4. 2: Explanation of Figure 4.3 on Experiment #1

Violated Row	Violated Column	Localized violation index	Corrected Violation
0	4	[0 , 4] : X	A

To give extra assurance of the model accuracy, the model displays the message shown in Figure 4-4 as evidence of correcting all of the detected violation.

Figure 4. 4: Experiment #1- Assurance Message

4.4.3.2 Experiment #2 : Multiple violations on one row only

This experiment shows a file that is violated on multiple locations only on one row or line. Figure 4-5 shows file original content and violated content.

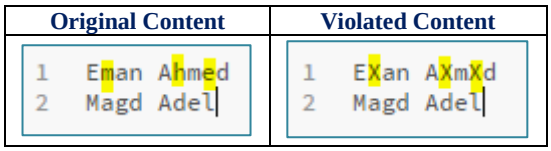


Figure 4. 5: Experiment #2- original and violated content

When an integrity check request is made , Figure 4-6 shows the returned result.

Violation Number	Row Index	Column Index	Localized Violation
0	0	1	X
1	0	6	X
2	0	8	X

Correction Number	Row Index	Column Index	Corrected Violation	Corrected Row
0	0	1	m	
1	0	6	h	
2	0	8	e	Eman Ahmed
ExecutionTime Online =	0.0016789436340332			

Figure 4. 6: Experiment #2 -Localization and Correction

Figure 4-6 is explained in Table 4.3:

Table 4. 3: Explanation of Figure 4.6 on Experiment #2

Violated Row	Violated Column	Localized violation index	Corrected Violation
0	1	[0 , 1] : X	M
0	6	[0 , 6] : X	H
0	8	[0 , 8] : X	E

Figure 4-7, ensures the model efficiency and accuracy.

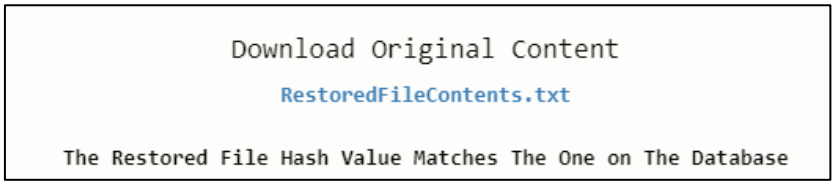


Figure 4. 7: Experiment #2- Assurance Message

4.4.3.3 Experiment #3 : Multiple violations on one column only

This experiment shows a file that is violated on multiple locations only on one column. Figure 4-8 shows file original content and violated content.

Original Content	Violated Content
1 Eman Ahmed 2 Magd Adel	1 EmXn Ahmed 2 MaXd Adel

Figure 4. 8: Experiment #3- original and violated content

When an integrity check request is made , Figure 4-9 shows the returned result.

Violation Number	Row Index	Column Index	Localized Violation
0	0	2	X
1	1	2	X

Correction Number	Row Index	Column Index	Corrected Violation	Corrected Row
0	0	2	a	Eman Ahmed
1	1	2	g	Magd Adel
ExecutionTime Online =	0.0017850399017334			

Figure 4. 9: Experiment #3- Localization and Correction

Figure 4-9 is explained as in Table 4.4:

Table 4. 4: Explanation of Figure 4.9 on Experiment #3

Violated Row	Violated Column	Localized violation index	Corrected Violation
0	2	[0 , 2] : X	A
1	2	[1 , 2] : X	G

4.4.3.4 Experiment #4 : Multiple violations on Distinct Locations

This experiment shows a file that is violated on multiple locations where none of the violations locate on the same row and on the same column. In other words, the violation is caused by one character change per row or column. Figure 4-10 shows file original content and violated content.

Original Content	Violated Content
1 Eman Ahmed 2 Magd Adel	1 EXan Ahmed 2 Magd Xdel

Figure 4. 10: Experiment #4-original and violated content

When an integrity check request is made, Figure 4-11 shows the returned result

Violation Number	Row Index	Column Index	Localized Violation
0	0	1	X
1	0	5	A
2	1	1	a
3	1	5	X

Correction Number	Row Index	Column Index	Corrected Violation	Corrected Row
0	0	1	m	Eman Ahmed
1	0	5	A	Eman Ahmed
2	1	1	a	
3	1	5	A	Magd Adel
ExecutionTime Online =	0.0021817684173584			

Figure 4. 11: Experiment #4- Localization and Correction

Besides the localized and corrected violations, the results on Figure 4-11 displays some values that were not violated in the first place. That is one drawback of the model at this type of experiment. The actual violations on the file derived from Figure 4-11 are as in Table 4-5:

Table 4. 5: Explanation of Figure 4.11 on Experiment #4

Violated Row	Violated Column	Localized violation index	Corrected Violation
0	1	[0 , 1] : X	M
1	5	[1 , 5] : X	A

Despite this drawback, the model is able to localize and correct the violations accurately and the assurance message is displayed in Figure 4-12.

Figure 4. 12: : Experiment #4 -Assurance Message

The question that comes to mind, why this drawback happens? That is because the only way to perform localization based on the metadata about the file is as previously explained in section 3.4.2 Figure 3-11. The solution suggested for solving this drawback is:

Extra metadata is needed about the columns of each row. That will enable the localization to refer every column in the list of violated columns to its own row. Which the proposed model does not have. Table 4-6 shows experiments of the model applied to different file sizes. The table also shows the execution time and the metadata size for each file. In all of the experiments in Table 4.6.

1- The model was able to correct all the localized violations.

2- Execution time was measured at 25 violations in each checked file.

Table 4. 6: Model Tested on Different File Sizes in Which All the Localized Violation Were Corrected

1	2	3	4	5	6	7	8	9
#	File Size (KB)	# Rows	# Columns	Row Hash File Size (KB)	Column Hash File Size (KB)	Metadata Size in Total (KB)	Execution Time (µs)	Metadata size to original data size
1	5	38	148	3	10	13	0.0174	2.60
2	15	166	148	11	10	21	0.0234	1.40
3	30	332	148	22	10	32	0.0457	1.06
4	40	422	148	27	10	37	0.0480	0.925
5	52	528	148	34	10	44	0.0540	0.846
6	60	664	148	43	10	53	0.0636	0.883
7	80	844	148	54	10	64	0.0695	0.800
8	100	1052	148	67	10	77	0.1036	0.770
9	200	2104	148	134	10	144	0.1794	0.720
10	300	3156	148	201	10	211	0.2761	0.703
11	400	4208	148	268	10	278	0.3786	0.695
12	500	5257	274	334	18	352	0.6685	0.694
13	600	6262	274	398	18	416	0.7920	0.693
14	700	7314	274	465	18	483	0.9244	0.690
15	800	8366	274	532	18	550	1.0278	0.687
16	1000	14374	146	913	10	923	1.0674	0.923
17	2000	30247	146	1920	10	1930	2.2899	0.965

Based on the results in Table 4-6, Figure 4-13 shows the percentage of metadata size from the original file size. As seen on the chart in Figure 4-13:

- At the original file size of 40 KB: The size of metadata starts to be smaller than the original file size.

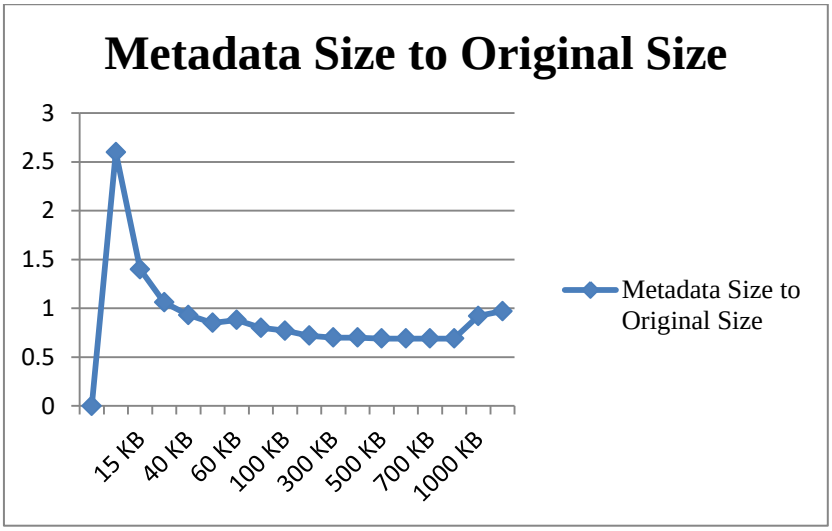


Figure 4. 13: Percentage of metadata size to original data size

Figure 4-14 shows the execution time of the files used in Table 4-6. The maximum execution time was 2.5 µs when a file of one megabyte was used.

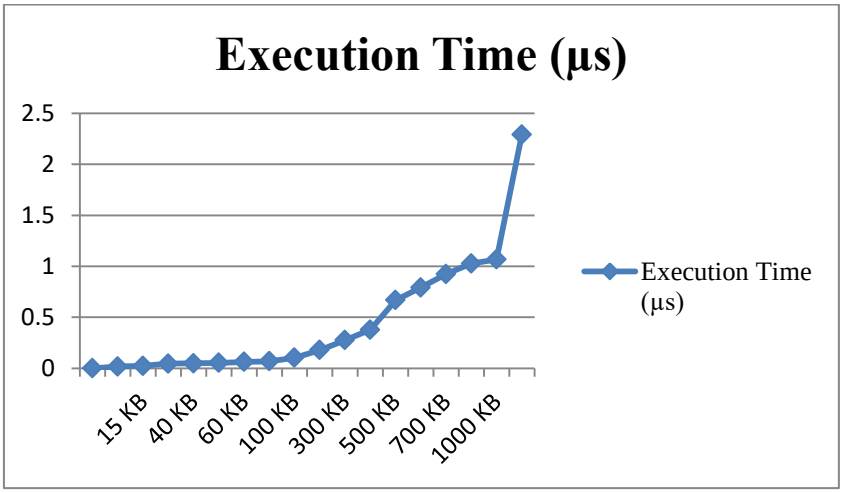


Figure 4. 14: Model execution time for files in Table 4-6

4.5 Model Evaluation

Aspects of evaluation as discussed in section 4.3 are three :

4.5.1 Accuracy Evaluation

The accuracy of the proposed model is measured by three factors:

1- Model instantly detects if file integrity was violated.
   - Within the model limitations, the model is able to :
2- Localize the indexes of violations.
3- Correct all the localized violations.

Table 4-6 shows the files that were tested by the model. All the violations that were discovered on the files were localized and corrected. This concludes that the model's accuracy is high within the four experiments criteria. The problem is that at experiment four on section 4.4.3.4, non-violated characters besides the violated ones are localized as violations. This problem is not considered serious. Since the model keeps the non-violated characters as they are and only corrects the violated ones. So, the overall result proves that the model is highly accurate.

4.5.2 Computation Storage

Computation storage is about the size of extra storage needed other than the original data. The extra storage mentioned is what makes the model perform its tasks accurately. That is row hash file and column hash file sizes are the metadata of the original data for the file. And extra storage is needed for this metadata. Column eight of Table 4-6 shows the size of the metadata needed for each file. Comparing column two which is the file size with column 7, Several notices are noticed.

- It's clear that the size of metadata stays bigger than the size of the original data until the file with size 40 KB.
- Starting with 40 KB file, the size of metadata starts to decrease slightly.
- Computing the average on column 9, it shows that the ***metadata size is 0.944 less than the original data size.***

4.5.3 Computation Efficiency

Computation efficiency is about the time it takes the model to localize and correct the violations on the file. Also, the time for processing the file to check is included.

The execution time is less than one microseconds for files under 100 KB in size. And increases gradually until it reaches 1.05 microseconds for a file with 1000 KB size. Execution time for the tested files is shown in column 8 of Table 4-6 with an average of 0.4723 microseconds.

4.6 Summary

Experiments to test the performance and efficiency of the model were presented in this chapter. Within the limitations of the model, model evaluation evolved around three aspects: Accuracy of results, computation efficiency, and computation storage. The next chapter presents Two optimization approaches for the proposed model. The proposed model accurately localizes and corrects violations within limitations previously explained in four experiments on section 4.4. The accuracy of the model becomes a serious issue and the model becomes inefficient to correct the violations on this case :

When multiple violations come to locate on the same row and column as shown in Figure 4-15.

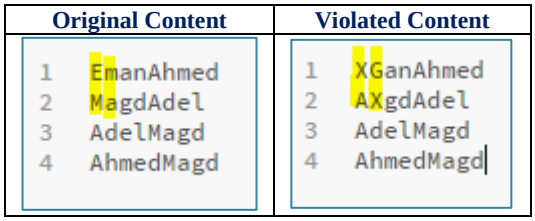


Figure 4. 15: Multiple Violations Cross

Violation Number	Row Index	Column Index	Localized Violation
0	0	0	X
1	0	1	G
2	1	0	A
3	1	1	X

Correction Number	Row Index	Column Index	Corrected Violation	Corrected Row
ExecutionTime Online =	0.0088250637054443			

Figure 4. 16: Localization but No Correction

As seen in Figure 4-16, the model was able to localize the violations but none of them was corrected. This issue is the reason for presenting the two optimization models in chapter 5.

The optimization was based on discarding the limitation of the model regarding violations. So the expected from the two optimization approaches is, being able to localize and correct all the violations even if they locate on the same row or column.

Chapter 5
Model Optimization

Chapter 5: Model Optimization

5.1 Overview

This chapter is divided into two sections that present the two optimization models. Each section explains the modifications made on the previous model and discusses the results. The main purpose of the optimization is to provide a new model that is able to localize and correct all violations regardless of their position or index. The good outcomes and the drawbacks of each optimization model are also discussed.

5.2 First optimization Model : Sub Row Index Addition

If the model of this thesis was to have a file where multiple violations locate on the same column and row, Then the correction would not be possible. However, localization of all the violations with other non-violated characters is a result. See Figure 4-16 on section 4.6. This approach suggests having another type of metadata other than the row and column hash values file. The metadata should enable the model to localize only and only the violated characters. Also, it should correct them all and give an assurance of its efficiency. The purpose of this data is to be able to refer the violated columns in the return column list each to its own row. Meaning that data about the characters of each row is all needed. Let's call it:

Sub Row Hash Values: Since the localization is done based on localizing the violated character, having a hash value for every character makes sense. So sub row hash is an array that contains the hash values of every element in the row matrix. Sub Row Hash **SRH [reh_1 , reh_2, reh_3, …. , reh_{n*m}]** is an array that contains the hash values for the elements in the Row matrix **R [re^1, re^2 , re^3 , …. , re_{n*m}]**, where n is the number of rows, m is the number of columns, **re** is row element and **reh** is row element hash. See Figure 5-1.

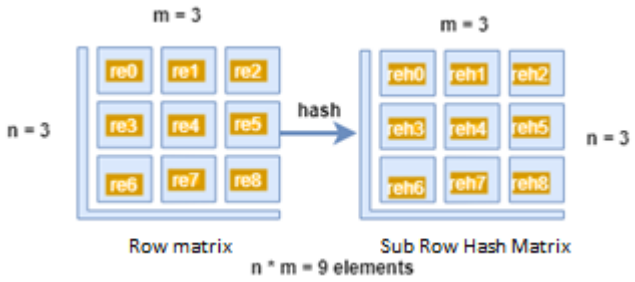


Figure 5. 1: Sub Row Hash Matrix

Let's take some time analyzing this sub row hash data. Looking at Table 4-6, on row one with 5 KB file size. The following is noticed:

- 148 columns result with 10 KB in size for column hash file.
- Applying this on 38 rows, sub-row hash for each row results with 38 X 10 = 380 KB file size.
- Moving the extra storage from 13 KB in total to 383 KB for a 5 KB file size.
- Also, moving the extra storage from 37 KB in total to 4,220 KB for a 40 KB file size.

So to face the big size for extra storage, the suggestion made is as follows:

- Why not store indexes to characters instead of hash values. This will definitely reduce the size needed to refer each character from 32 (256 bit / 8) Byte to one or two or three bytes at most.
- Assuming agreement of the suggestion. The indexes to characters are from the list of characters produced for each file and saved on the database.
- The list of characters is produced from the characters of the its own file when saved on the database. The order of these characters in the list is constant for the file it belongs to.
- The list of characters may contain indexes to 128 ASCII English characters. So the maximum index will occupy three bytes at most. This because indexes to 128 ASCII characters starts from 0 (one-byte) to 127 (three-bytes).

Sticking to what was said, This model adds a sub-row Index file to metadata but only it will contain indexes to characters instead of hash values. This model will be tested on the same set of files shown in Table 4-6. But first, let's see how the second stage of localization will change after the sub row index addition.

5.2.1 First Optimization Model : Localization

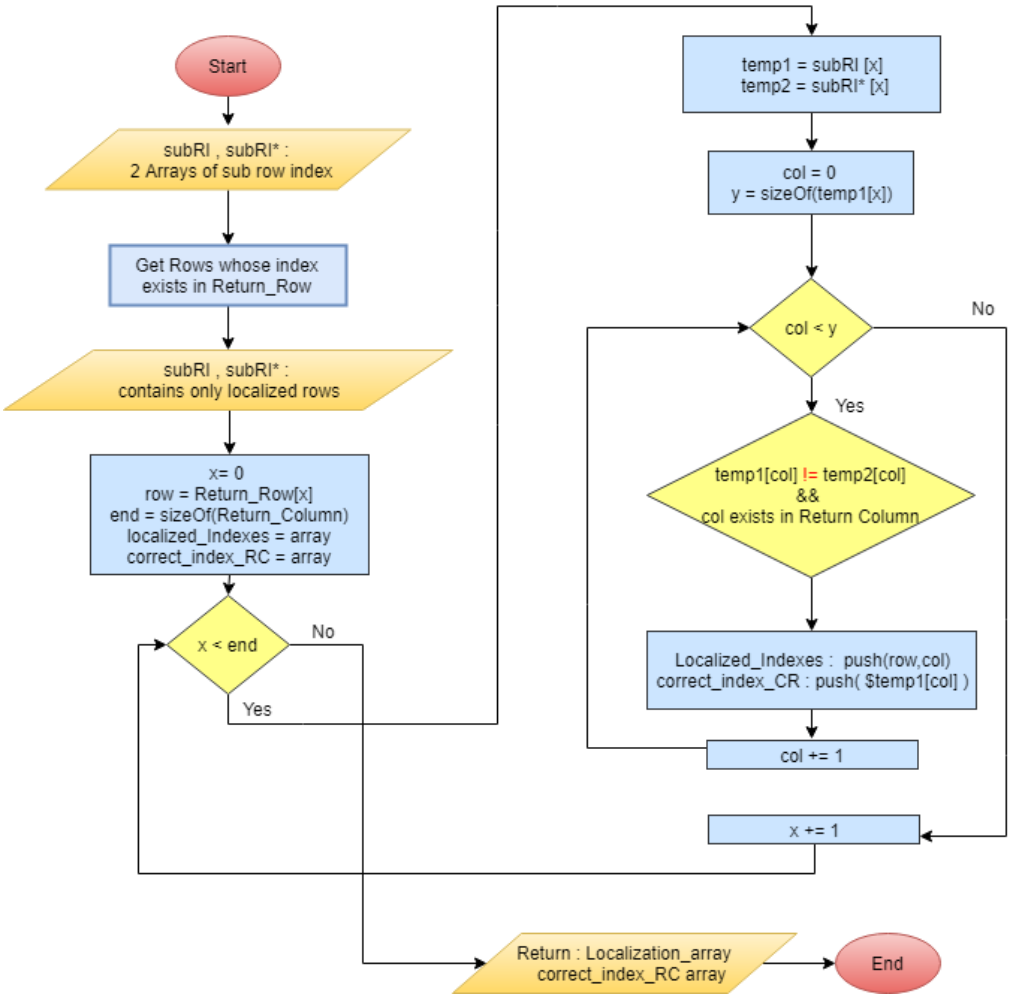


Figure 5. 2: Localizing the Violation in Each Row

Figure 5-2 is explained as follows:

1. Input two sub row index matrices, one retrieved from database **subRI** and the other computed at the request for check **subRI***.

2. Deduct the rows of subRI and subRI* so that they only contain the violated rows whose indexes exists at the return row array from the first stage.
3. Row index is paired with column **index [row][column]** and that implies localized violation index.
4. Initialize the **row** variable with an index from return row to apply the second stage on, Say row = 0.
5. The main concept of the flow chart is as follows:
    a. Start an outer loop that loops at the size of the return column array from the first stage.
    b. Loop on every two corresponding rows on the sub row index matrices. As shown figure 4-11 **temp1** and **temp2**.
    c. Temp1: column index values from database [ci_0 , ci_1 , ci_2 , …, ci_n] elements are compared with Temp2: column index values computed at check request [ci^*_0 , ci^*_1 , ci^*_2 , …, ci^*_n], ci for character index.
    d. Now start another loop that loop at the size of temp1, which is the variable that contains the hash values for the current row. The index of the element currently being compared is the column number.
    e. The condition for localizing the column and pairing it with row currently being checked :
        - If element **ci_0** matches element **ci^*_0** , then that character(column) is not violated. Otherwise, that is a violated column. Say column = 0.
        - If the column exists in the return column array from the first stage, then
            - Violation is localized at index [row][column]
            - Push the index say [0][0] at the localization indexes array
            - Push the correct hash Temp1[0] from Temp1 in another array called correct_char_hash. The purpose of this array will be explained in a minute.
    f. Loop until all the elements in the current temp are done.
    g. Get the next row from the sub row hash. Repeat from step 5 until all the rows are processed.

By the end of the second stage of violation localization module, two arrays are passed to Violation Correction module.

1- **Localized Indexes Array:** previously explained in section 3.4.2 division b.
2- **Correct Characters Index Array:** those index values are the original index values that exist in the sub row index matrix retrieved from the database. Now the question that comes to mind, why prepare those hash values now?.

An answer to that: since the correction process is done only on localized indexes. And the localized indexes result from looping over the sub row index matrix. To prevent repeating the loop to get the correct index value for every localized index , it's saved in the correct array at the instance of localizing. Using the first optimization approach, Table 5-1 shows the results of the same set of files used in Table 4-6.

Table 5. 1: First Optimization Model Tested on Different File Sizes in Which all Localized Violations were Corrected

1	2	3	4	5	6	7	8
#	File Size (KB)	Row Hash File Size (KB)	Column Hash File Size (KB)	Sub Row Index File Size (KB)	Size in Total (KB)	Execution Time (µs)	Metadata size to original data size
1	5	3	10	12	25	0.0246	5
2	15	11	10	44	65	0.0332	4.33
3	30	22	10	87	119	0.0656	3.966
4	40	27	10	115	152	0.0846	3.8
5	52	34	10	145	189	0.0954	3.634
6	60	43	10	174	227	0.1390	3.783
7	80	54	10	230	294	0.1598	3.675
8	100	67	10	286	363	0.1876	3.630
9	200	134	10	571	715	0.3874	3.575
10	300	201	10	856	1,067	0.5707	3.556
11	400	268	10	1,141	1,419	0.7770	3.547
12	500	334	18	2,073	2,425	1.8631	4.85
13	600	398	18	2,467	2,883	2.2074	4.805
14	700	465	18	2,882	3,365	2.5606	4.807
15	800	532	18	3,296	3,846	2.9689	4.807
16	1000	913	10	3,676	4,599	2.9703	4.599

Based on the results in Table 5-1, Figure 5-3 shows the percentage of the metadata size from the original file size. As seen on the chart on Figure 5-3:

- At all file sizes, the size of metadata is never less than the original file size.

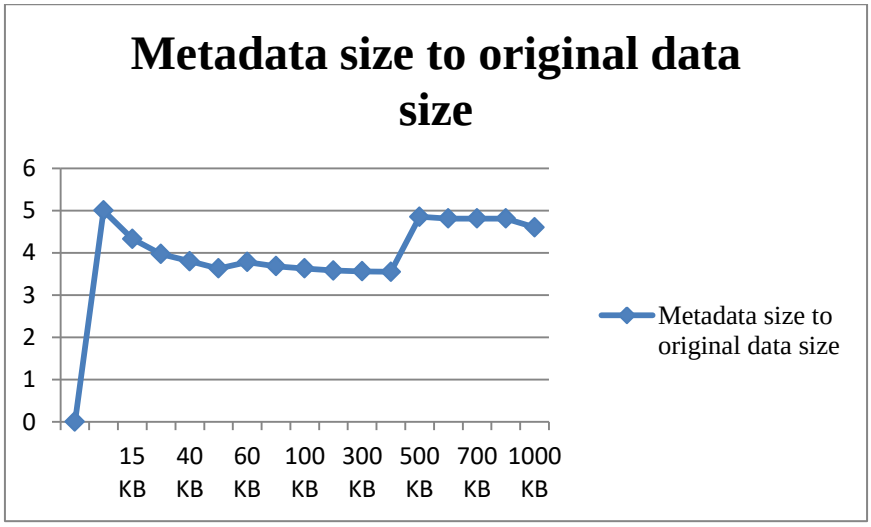


Figure 5. 3: Percentage of metadata size to original data size

Figure 5-4 shows the execution time of the files used in Table 5-1. The maximum execution time was 3 microseconds when a file of one megabyte was used.

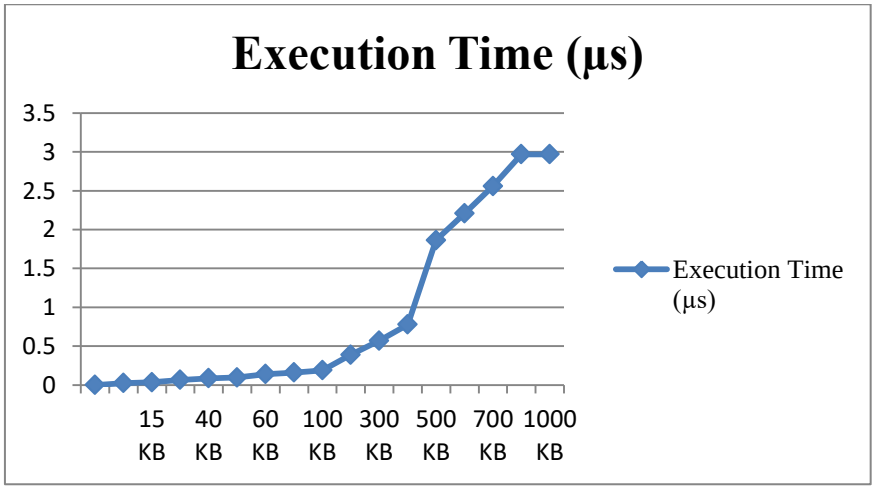


Figure 5. 4: Model execution time for files on Table 5-1

5.2.2 First Optimization Model Discussion

Looking at the last three columns of Table 5-1, the following is noticed:

- Total size for metadata in column 6 is huge compared to the original file size in column 2.
- Computing the average on the last column, ***metadata size is 6.64 bigger than the original data size.***
- The execution time on column 7 looks kind of acceptable to process such a huge amount of metadata. ***The average execution time of this model equals 0.9434 microseconds***.
- The first optimization model is proved to be able to localize and correct every single violation on the file. So the purpose of optimization is achieved.

Comparing the results of Table 4-6 with the results of Table 5-1. Table 5-2 shows a comparison for both the original model and the first optimized model.

Table 5. 2: Compare results of original model and first optimization model

	Original Model	**First Optimization Model**
Accuracy of Model	100 % accurate	100 % accurate
Average Execution Time	0.4723 µs	0.9434 µs
Average Metadata Size to Original Data Size	0.944 smaller than original file size	6.64 larger than original file size

The results in Table 5.2 conclude that:

- The first optimization model was able to correct all the localized violations that were caused by more than one character change. However,
- The original model is faster than the first optimized model.
- The metadata size on the original model is 7 (6.64 / 0.944) times less than the first optimization model.

So in conclusion, the drawbacks of this model surpass its benefits. Such large metadata size will need extra storage and that will not be encouraging for using this model.

5.3 Second optimization Model : Read File Content As One Row

The intents on proposing this model were like this: First, optimize the original model localization and correction to include any position of violation. Second, optimize the first optimized model from metadata size side or point of view.

This model is as simple as the original model of this thesis, it only differs on the following:

- There is only one row that represents the whole file content. This implies that:
    - Row hash file in not needed. Because the row hash will equal the file hash.
- The number of columns is the number of characters in the row. This implies that:
    - Column hash file will contain values of the number of columns.
    - Also, it will contain indexes to characters, not hash values. The indexes are from the list of characters as explained in section 5.2.

The only metadata needed for this approach is :

- File hash value and List of characters stored on the database.
- Column Hash file that contains indexes instead of hash values. Stored on a dedicated server.

Why expect this approach to reduce the size of metadata obtained from the first optimization approach? Because it avoids adding extra bytes to the original file content, unlike the first optimization model.

The first optimization model adds extra bytes during the method of constructing the row matrix:

- The first optimization approach fills line characters into a row with the length of the longest line in the file.
- So, if the line has 3 characters and the row length say 15:
- All of the lines are considered 15 characters long. Based on the example, the first 3 characters are the line characters and the rest 12 are filled with an empty character.
- This method causes the size of metadata to be huge because extra bytes are added to shorter lines.

Figure 5-5 and Table 5-3 demonstrates the second optimization approach. Meaning if your file has the content as in Figure 5-5:

Figure 5. 5: : File Content

The second optimized approach reads the content on Figure 5-6 as in Table 5-3.

Table 5. 3: Content Translated to One Row

E	M	a	N		A	h	m	e	D	EOL	M	a	g	d		A	d	E	L	EOF

Table 5-4 shows the results of testing the second optimized approach on the same data set used in Table 4-6.

Table 5. 4: Second Optimization Model Tested on Different File Sizes in Which All The Localized Violations Were Corrected

1	2	3	4	5	6
#	File Size (KB)	Column Hash File Size (KB)	Size in Total (KB)	Execution Time (µs)	Metadata size to original data size
1	5	12	12	0.0090	2.4
2	15	34	34	0.0342	2.266
3	30	68	68	0.0414	2.266
4	40	95	95	0.0672	2.375
5	52	120	120	0.0690	2.307
6	60	136	136	0.0721	2.266
7	80	189	189	0.0880	2.362
8	100	237	237	0.1113	2.370
9	200	473	473	0.2218	2.365
10	300	709	709	0.3218	2.363
11	400	946	946	0.4245	2.365
12	500	1,182	1,182	0.6685	2.364
13	600	1,404	1.404	0.6433	2.340
14	700	1,641	1.641	0.7385	2.344
15	800	1,877	1,877	0.8248	2.346
16	1000	2,577	2,577	1.0267	2.577
17	2000	5,420	5,420	2.1865	2.710

Based on the results on Table 5-4, Figure 5-6 shows the percentage of metadata size from the original file size. As seen on the chart in Figure 5-6:

- At all file sizes, the size of the metadata is never less than the original file size. It always resides between 2-3 times the size of the original file size.

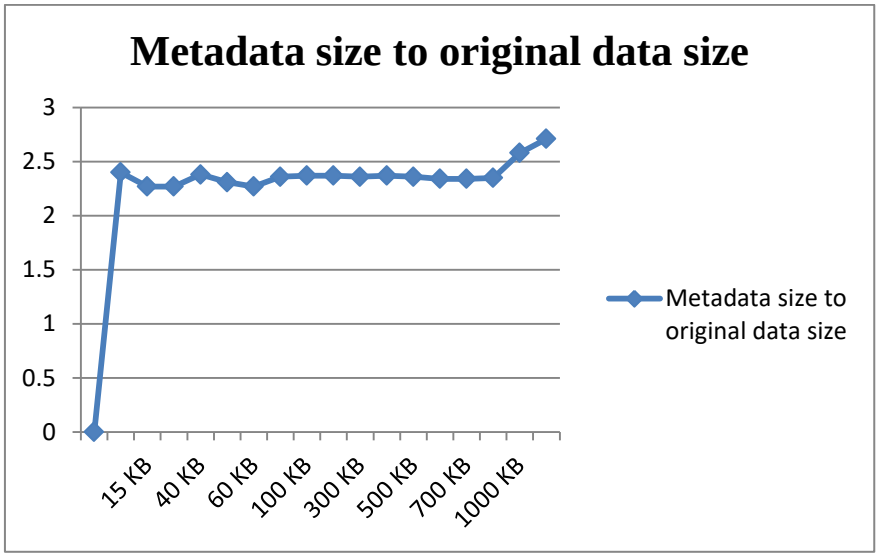


Figure 5. 6: Metadata size to original data size

Figure 5-7 shows the execution time of the files used in Table 5-4. The maximum execution time was 3 µs when a file of one megabyte was used.

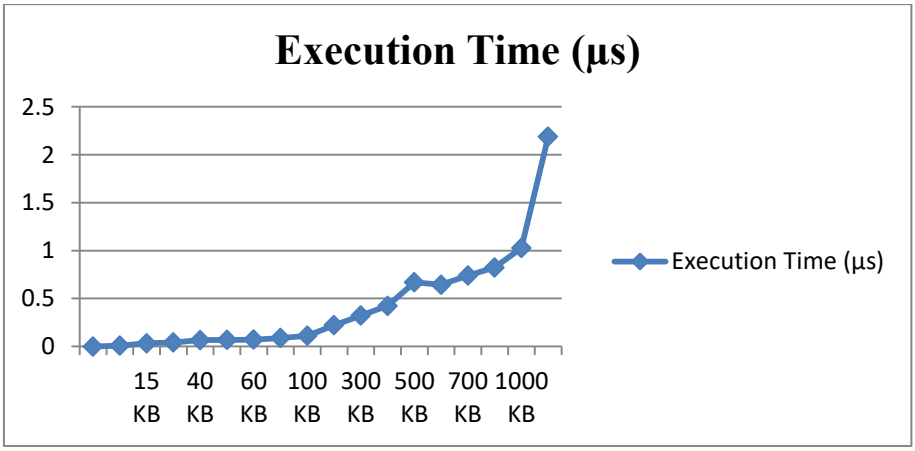


Figure 5. 7: Model execution time for files on Table 5-1

5.3.1 Second Optimization Model Discussion

From Table 5-3, the following is noticed:

- Column 5 shows: Metadata size is within an average of 1.828 larger than the file size.
- Column 6 shows: Execution time has an average of 0.444 microseconds.

Now, let's discuss the results in Table 5-3 with the results from Table 4-6 and Table 5-1.

a. First, Second optimized model is compared with the original model is shown in Table 5-5.

Table 5. 5: Comparison of Second optimization model with the original model

	Original Model	**Second Optimization Model**
Accuracy of Model	100 % accurate	100 % accurate
Average Execution Time	0.4723 µs	0.444 µs
Average Metadata Size to Original Data Size	0.944 smaller than original file size	1.828 larger than original file size

The results in Table 5-5 are interpreted as follows:

- Second optimization model :
  1- Was able to correct the localized violations wherever their index locates. In this point, it surpassed the original model.
  2- Is faster than the original model with an average execution time of 0.444 µs.
  3- Metadata size is 1.936 (1.828 / 0.944) in average larger than the metadata size of the original model.

b. **Second,** Second and first optimized models are compared together in Table 5-6.

Table 5. 6: Comparison of second and first optimization model

	Second Optimization Model	First Optimization Model
Accuracy of Model	100 % accurate	100 % accurate
Average Execution Time	0.444 µs	0.9434 µs
Average Metadata Size to Original Data Size	1.828 larger than original file size	6.64 larger than original file size

The results on Table 5-6 are interpreted as follows:

- Second optimization model :
  4- Was able to correct the localized violations wherever their index locates. In this point, it surpassed the original model.
  5- Twice (0.9434 / 0.444) as fast as the first optimized model.
  6- Metadata size is 3.63 (6.64 / 1.828) in average smaller than the metadata size of the first optimized model.

5.4 Summary

The optimized models aimed to enhance the original model's ability of localizing and correcting violations no matter of violation position or location. The optimization affected both metadata size as well as the execution time. Discussion of results and effects were also presented in this chapter. In overall summary, the second optimized model is the best candidate between the three models from the following aspects:

1- Is able to correct all the violations regardless of their index. In other words, any localized violation in any index will be corrected accurately.
2- The execution time is faster as the results shown in Table 5-6.

Chapter 6

Discussion, Conclusion and Future Work

Chapter 6: Discussion , Conclusion and Future Work

6.1 Overview

In this chapter, discussion of the three models is held. As well as the conclusion of the work done to accomplish the purpose of the presented model is given. Also, suggestions for future work and research are given.

6.2 Discussion of the Three Models

As file sharing over the cloud widely spread, many researchers aimed to build models for integrity violation detection. Various models were presented only to inform the file owner if file integrity was violated or not. The model of this thesis not only detects the violation of file integrity but also accurately localize the violated characters and corrects them. The contribution of this model can be used as an infrastructure to localizing and correcting violations in future works.

The evaluation of this model was based on measuring three factors: accuracy, execution time and metadata size. Table 6-1 shows the results of the proposed model. The results were conducted on 17 files with 25 violations in each file.

Table 6. 1: Original Model Evaluation

	Original Model
Accuracy of Model	100 % accurate according to limitations
Average Execution Time	0.4723 µs
Average Metadata Size to Original Data Size	0.944 smaller than original file size

The accuracy of the model is considered high if all the localized violation were corrected. ***And the results of the experiments showed the model to be highly accurate***. Let's not forget that in this model, the index of violation was limited to the indexes description for four experiments on Table 4-1 on section 4.4.

The execution time of the model to respond to the user must be measured since it's an online service. The model was able to convey an approximately small execution time.

This is proved by experiments results to be computationally efficient. ***The average execution time was 0.4723 microseconds***.

As for the extra storage needed for the metadata, it requires the metadata size to be smaller than the original data size. The model was able to achieve this for files larger than 40KB. In average, ***metadata size was 0.944 smaller than the original file size.***

Because violation location was limited to limited locations (indexes) explained in section 4.4, two optimization models were also presented in chapter 6. They aimed to enhance the original model by including any violation location. The two optimized models were able to locate and correct violations at any location (index). Table 6-2 shows the results of the three models.

Table 6. 2: Comparison of the three models

	Original Model	**First Optimization Model**	**Second Optimization Model**
Accuracy of Model	100 % accurate within limited violation indexes	100 % accurate to all violation indexes	100 % accurate to all violation indexes
Average Execution Time	0.4723 µs	0.9434 µs	0.444 µs
Average Metadata Size to Original Data Size	0.944 smaller than original file size	6.64 larger than original file size	1.828 larger than the original file size

To clarify the results shown in Table 6-2, Figure 6-1, Figure 6-2 And Figure 6- show charts of the results.

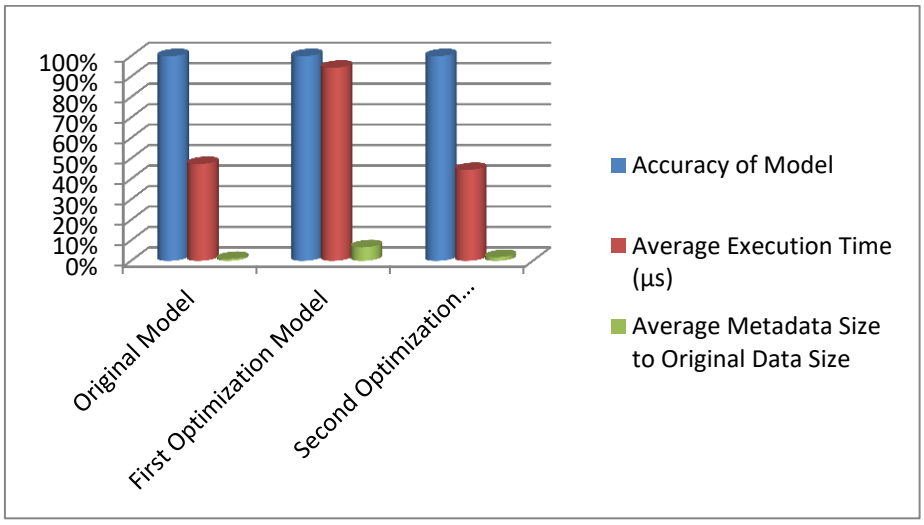


Figure 6. 1 : Comparison of the Three Models

As Figure 6-1 shows, The second optimization model is the closest model to the original model. Except that it has no limits on violations locations and can correct all localized violations. Looking at the execution time column, the chart shows that the two models just mentioned have almost equal average execution time. Figure 6-2 details the execution time at every file size used during the testing process. The columns in green show that the second optimization model has a better execution time than the original model.

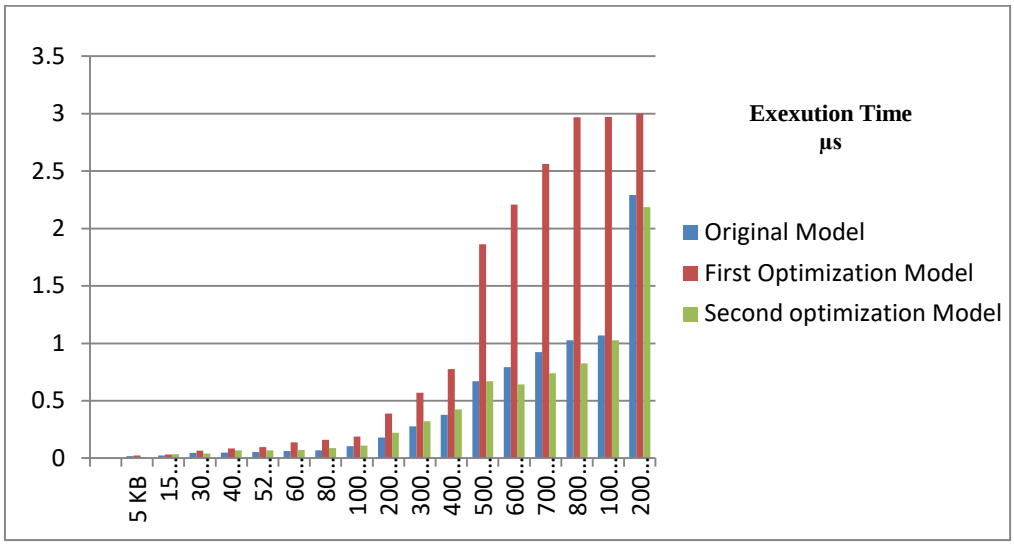


Figure 6. 2: Three models execution time

Figure 6-3 shows the comparison of metadata size of the three models. As seen in the chart, the second optimization model is the closest to the original model and future works aimed to reduce metadata size are suggested.

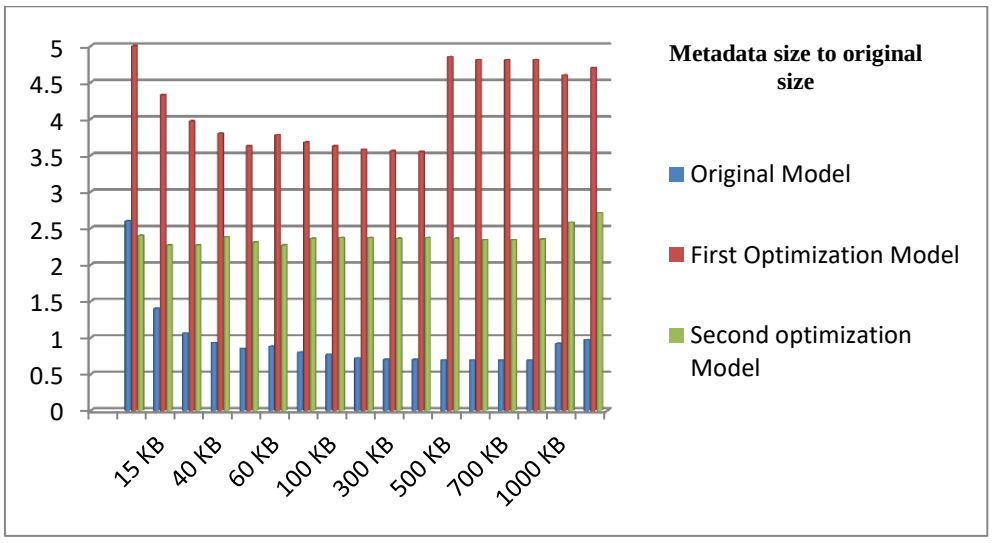


Figure 6. 3: Three models metadata size comparison with original file size

Description of violations Positions is available in Table 6-3.

Table 6. 3: Description of violation positions

Violation Position	Interpretation	Description
SR : SC	Single Row : Single Column	The file has only one violation
SR : MC	Single Row : Multiple Columns	Multiple violations locate in one row
SC : MR	Single Column : Multiple Rows	Multiple violations locate in one column
MR : DC	Multiple Rows : Distinct Columns	Multiple violations in multiple rows where each row is violated in one column
MC : DR	Multiple Columns : Distinct Rows	Multiple violations in multiple columns where each column is violated in one row
MR : MC	Multiple Rows : Multiple Columns	Multiple violations in multiple rows where each row is violated in more than one column and vice versa.

The positions of violations described in Table 6-3 will be used to refer to each model. That is where each model can localize and correct without exception will be listed in Table 6-4.

Table 6. 4: Position violation capability for each model

	Violation Position	Original Model	First Optimization Model	Second Optimization Model
Single Violation	SR : SC	√	√	√
Multiple Violations	SR : MC	√	√	√
	SC : MR	√	√	√
	MR : DC	√	√	√
	MC : DR	√	√	√
	MR : MC	×	√	√

Table 6-4 clearly shows that the original model can correct the violations at all locations except the ones described at the last row. However, the two optimized models were able to correct the violations in the case from the last row.

6.3 Conclusion

The evaluation of the proposed method was based on the success and accuracy of integrity violation detection and whether the violation was localized and corrected. The size of the metadata needed to perform the check process was also evaluated. The model was proved by testing and results to be highly accurate within described limitations. Results show that the model was executed within 0.4723 μs average execution time. Also, the size of metadata was about 0.944 from the original file size when the original file size was bigger than or equal to 40KB. Two models were presented in chapter 5 to enhance the original model of the thesis. Both aimed to target the original model weakness about the limited allowed violation position. Although both of the optimization models were able to fade this weakness successfully, other weakness appeared on the surface.

The results in Table 6-2 shows the results of the three models:

1- Metadata size was a problem in both of the optimization models since its always larger than the original data size.
   - The first optimization model produced metadata 6.64 on average larger than the original data size.
   - The second optimization model produced metadata 1.82 in average larger than the original data size.
2- As for execution time, the results in Table 6-2 shows that the second optimization model was faster than both the original and the first optimization model.

From the results, the second optimization model shows the best enhancements on two sides:

- Correction includes any localized index for violation.
- Execution time in the fastest with an average of 0.444 microseconds.

6.4 Future Work

The future work is suggested to provide solutions to the metadata size problem in the second optimization model. A near future goal is to reduce the size of metadata to equal the size of the original file.

Also, the second optimization model is empowered with the ability to restore the original content of the violated file. So, ***its recommended to be used as a file restoration system***. Under the condition that the user has already saved his file data on the system previously. See Figure 6-4.

Figure 6. 4: Restored Original File Content

Also, suggestions to apply the model on other file types shall be considered in future studies.

REFERENCES

References

Aldossary, S., & Allen, W. (2016). Data security, privacy, availability and integrity in cloud computing: issues and current solutions. *International Journal of Advanced Computer Science and Applications, 7*(4), 485-498.

Anil, S. L., & Thanka, R. (2013). A survey on security of data outsourcing in cloud. *International Journal of Scientific and Research Publications (IJSRP), 3*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., . . . Stoica, I. (2009). *Above the clouds: A berkeley view of cloud computing*. Retrieved from

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., & Song, D. (2007). *Provable data possession at untrusted stores.* Paper presented at the Proceedings of the 14th ACM conference on Computer and communications security.

Ateniese, G., Di Pietro, R., Mancini, L. V., & Tsudik, G. (2008). *Scalable and efficient provable data possession.* Paper presented at the Proceedings of the 4th international conference on Security and privacy in communication netowrks.

Bawaneh, M. J., Alkoffash, M. S., Alqrainy, S., & Muaidi, H. (2016). A Practical Comparison between Signature Approach and Other Existing Approaches in Error Detection over TCP. *Communications and Network, 8*(01), 31.

Bessani, A. N., Mendes, R., Oliveira, T., Neves, N. F., Correia, M., Pasin, M., & Verissimo, P. (2014). *SCFS: A Shared Cloud-backed File System.* Paper presented at the USENIX Annual Technical Conference.

Borshack, R., Thomas, A. F., Einav, E., & Taron, P. E. (2016). Large scale file storage in cloud computing. In: Google Patents.

Chi, L., & Zhu, X. (2017). Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (CSUR), 50*(1), 11.

Dang, Q. H. (2015). *Secure hash standard*. Retrieved from

De Spiegeleer, K. (2010). Efficient computer file backup system and method. In: Google Patents.

Garfinkel, S. L., & McCarrin, M. J. D. I. (2015). Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. *14*, S95-S105.

Han, S., Liu, S., Chen, K., & Gu, D. (2014). *Proofs of retrievability based on MRD codes.* Paper presented at the International Conference on Information Security Practice and Experience.

Kumar, R. S., & Saxena, A. (2011). *Data integrity proofs in cloud storage.* Paper presented at the Communication Systems and Networks (COMSNETS), 2011 Third International Conference on.

Leesakul, W., Townend, P., & Xu, J. (2014). *Dynamic data deduplication in cloud storage.* Paper presented at the Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on.

Lensing, P., Meister, D., & Brinkmann, A. (2010). *hashfs: Applying hashing to optimize file systems for small file reads.* Paper presented at the Storage Network Architecture and Parallel I/Os (SNAPI), 2010 International Workshop on.

Lulu, S. T. (2016). *A Model to Detect the Integrity Violation of Shared File in the Cloud.* The Islamic University–Gaza, Gaza, Palestine.

Luo, W., & Bai, G. (2011). *Ensuring the data integrity in cloud data storage.* Paper presented at the Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

Ora, P., & Pal, P. (2015). *Data security and integrity in cloud computing based on RSA partial homomorphic and MD5 cryptography.* Paper presented at the Computer, Communication and Control (IC4), 2015 International Conference on.

Rajathi, A., & Saravanan, N. (2013). A survey on secure storage in cloud computing. *Indian Journal of Science and Technology, 6*(4), 4396-4401.

Rao, R. V., & Selvamani, K. (2015). Data security challenges and its solutions in cloud computing. *Procedia Computer Science, 48*, 204-209.

Rong, C., Nguyen, S. T., & Jaatun, M. G. (2013). Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering, 39*(1), 47-54.

Samarati, P., di Vimercati, S. D. C., Murugesan, S., & Bojanova, I. (2016). Cloud security: Issues and concerns. *Encyclopedia on cloud computing*, 1-14.

Sodhi, G. K., Gaba, G. S. J. J. o. E. S., & Technology. (2018). AN EFFICIENT HASH ALGORITHM TO PRESERVE DATA INTEGRITY. *13*(3), 778-789.

Venkatesh, A., & Eastaff, M. S. (2018). A Study of Data Storage Security Issues in Cloud Computing.

Voorsluys, W., Broberg, J., & Buyya, R. (2011). Introduction to cloud computing. *Cloud computing: Principles and paradigms*, 1-41.

Wu, J., Ping, L., Ge, X., Wang, Y., & Fu, J. (2010). *Cloud storage as the infrastructure of cloud computing.* Paper presented at the Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on.

Xu, J., & Chang, E.-C. (2012). *Towards efficient proofs of retrievability.* Paper presented at the Proceedings of the 7th ACM symposium on information, computer and communications security.

Zafar, F., Khan, A., Malik, S. U. R., Ahmed, M., Anjum, A., Khan, M. I., . . . Jamil, F. (2017). A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends. *Computers & Security, 65*, 29-49.

Zhu, Y., Hu, H., Ahn, G.-J., Yu, M. J. I. t. o. p., & systems, d. (2012). Cooperative provable data possession for integrity verification in multicloud storage. *23*(12), 2231-2244.