



**CLOUD STORAGE SECURITY
AND
CRUD SUPPORTED DATA POSSESSION
WITH
INTEGRITY AUDITOR**

MAHMUT OFLAZ

APRIL 2016

**CLOUD STORAGE SECURITY AND CRUD SUPPORTED DATA
POSSESSION WITH INTEGRITY AUDITOR**

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY**

**BY
MAHMUT OFLAZ**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF
COMPUTER ENGINEERING**

APRIL 2016

Title of the Thesis: **Cloud Computing Security and CRUD Supported Data Possession with Integrity Auditor**

Submitted by **Mahmut OFLAZ**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.



Prof. Dr. Halil Tanyer EYYUBOGLU
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Müslim BOZYİĞİT
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.



Yrd.Doç. Dr. A. Nurdan SARAN
Supervisor

Examination Date: 05.05.2016

Examining Committee Members

Asst. Prof. Dr. A. Nurdan SARAN

(Çankaya Univ.)

Asst. Prof. Dr. Abdulkadir GÖRÜR

(Çankaya Univ.)

Asst. Prof. Dr. Tolga MEDENİ

(YBU)



STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Mahmut OFLAZ
Signature : 
Date : 30.04.2016

ABSTRACT

CLOUD STORAGE SECURITY AND CRUD SUPPORTED DATA POSSESSION WITH INTEGRITY AUDITOR

OFLAZ, Mahmut

M.Sc., Department of Computer Engineering

Supervisor: Yrd. Doç. Dr. A. Nurdan SARAN

April 2016, 54 pages

Cloud and cloud security are becoming more important while cloud technology is changing our lives in many ways. Because of data breaches, users and companies worry about migrating to cloud systems. When a client store data at untrusted server, verifying that the server possesses the original data is a challenging problem. Substantial portion of data possession offering focus on performance rather than security of the system. In this thesis, privacy focused data possession model has been offered; which is balanced form of increased privacy and performance. A trusted third party, an auditor, has been added to the system, which verifies the integrity of the data while preserving the privacy of the data on the cloud. Thereby, this auditor takes the burden from cloud user by delegating the user. In long term, integrity auditors can audit cloud systems by standardizing equivalent services which offered by clouds. In most cases, cloud users want to update/modify the stored data. The proposed data possession implementation supports modifying data, which has been stored on cloud. To measure the effects of the system parameters and general performance of the system, a test bed has been created which includes cloud user, cloud and integrity auditor. Experiments have been conducted to compare the proposal approach with the previous works to evaluate the cost of putting third party integrity auditor in the system, tests on encrypted data with different block and file sizes have been carried out.

Keywords: cloud security, data possession, secure storage, integrity auditor

ÖZ

BULUT BİLİŞİMDE DEPOLAMA GÜVENLİĞİ VE BÜTÜNLÜK DENETLEYİCİLİ GÜNCELLEME DESTEKLİ VERİ SAHİPLİĞİ

OFLAZ, Mahmut

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi: Yrd. Doç. Dr. A. Nurdan SARAN

Nisan 2016, 54 sayfa

Bulut bilişimin hayatımızdaki yeri arttıkça, bulut bilişim ve güvenliği önem kazanmaktadır. Veri sızıntıları nedeniyle, kullanıcılar ve şirketler sağlayıcılara güvenmemekte; veri güvenliği kaygılarından dolayı bulut bilişime geçmekten korkmaktadırlar. Bir kullanıcı, güvenilmeyen bir ortamda veri depoladığında, bu ortamda depolanmış olan verinin orjinal halini idame ettiriyor olmasının ispatlanabilmesi zorlu bir problemdir. Bulut bilişimde veri güvenliği ile ilgili yapılan çalışmaların çoğu güvenlik yerine performans odaklı olarak gerçekleştirilmiştir. Bu tez çalışmasında, güvenlik ve performans arasında denge kuran güvenli odaklı bir veri sahipliği modeli sunulmaktadır. Sisteme eklenen üçüncü parti güvenlik denetleyicisi, hem bulut sağlayıcılarını hem de bulut üzerindeki verilerin güvenliğini denetlemektedir. Bütünlük denetleyicileri, kullanıcıların vekaletlerini almak yoluyla; kullanıcıları, sürekli olarak verilerinin bütünlüğünü kontrol etmekten de kurtarmaktadır. Uzun vadede, bulut sağlayıcılarının, bütünlük denetleyicilerin kullanacağı ortak ve benzer servisleri sağlaması standart hale getirilerek; bulut sağlayıcıları da daha denetlenebilir hale getirilebileceği öngörülmektedir. Diğer bir taraftan, bu veri sahipliği modeli, buluta yüklenmiş olan bir verinin daha sonradan içeriğinin güncellenmesine de imkan vermektedir. Sistem parametrelerinin sistem performansı üzerindeki etkilerini ve sistemin genel performansını ölçmek için kullanıcıyı, bulutu ve bütünlük denetleyicisini içeren bir test uygulaması geliştirilmiştir. Bu tez çalışmasında önerilen modele ek olarak, daha önceki çalışmaların da performans testleri gerçekleştirilmiş ve bu tezde önerilen modelin performansı ile kıyaslanmıştır. Test metrikleri, bütünlük denetleyicisinin varlığını, paketlerin şifrelenmesini, paket ve blok boyutlarını içermekte; bu metriklerin sistem üzerindeki etkilerini incelemektedir.

Anahtar Kelimeler: bulut güvenliği, ispatlanabilir veri sahipliği, güvenli depolama, bütünlük denetleyicisi

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Yrd. Doç. Dr. A. Nurdan SARAN for her supervision, special guidance, suggestions, and encouragement through the development of this thesis.

It is a pleasure to express my special thanks to my fiancée for sharing my burden, to my family for their valuable support and to dear friend Murat for proof checking of the entire thesis.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM PAGE	iii
ABSTRACT	iv
ÖZ.....	v
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES.....	ix
LIST OF ALGORITHMS	x
LIST OF TABLES	xi
LIST OF EQUATIONS	xii
LIST OF ABBREVIATIONS	xiii

CHAPTERS

1.INTRODUCTION	1
1.1 Cloud Computing and Security Aspects.....	1
1.2 Literature Survey	13
1.3 Motivation.....	17
1.4 Outline	18
2.DESIGN of SCHEMES.....	19
2.1 Definition of Integrity Auditor and Purpose.....	19
2.2 Design of IIA	21
2.3 Design of IAaaS.....	21
2.4 Design of CIAaaS	23
3.IMPLEMENTATION of SCHEMES.....	25
3.1 Background.....	25
3.2 Implementation of IIA	28
3.3 Implementation of IAaaS.....	36
3.4 Implementation of CIAaaS	36
4.Test Bed	37
4.1 Test Bed Environment	37

4.2	Test Bed Restrictions	38
4.3	Application Details	38
4.4	Experimental Results	42
5.	Conclusion and Future Work.....	53
	REFERENCES	R1
	APPENDIX A: COMPARISON OF PREVIOUS SYSTEMS	A1
	APPENDIX B: DETAILED FLOW OF SYSTEM	B1
	APPENDIX C: DETAILED TEST RESULTS.....	C1
	APPENDIX D: PSEUDOCODES	D1

LIST OF FIGURES

Figure 1: General Cloud Architecture [3]	2
Figure 2: SaaS, PaaS and IaaS [3].....	3
Figure 3: Separation of Responsibilities on Cloud Models [5].....	4
Figure 4: Types of Cloud Computing aspect of deployment model [7]	5
Figure 5: Verification Agencies	17
Figure 6: IIA.....	21
Figure 7: IAaaS	22
Figure 8: CIAaaS	23
Figure 9: Merkle Hash Tree	25
Figure 10: Simple Skip List	26
Figure 11: Flows on System	30
Figure 12: Generating Q_{ID} by hashing ID	31
Figure 13: Generating T_i	31
Figure 14: Generating P_i	32
Figure 15: Obtaining P.....	32
Figure 16: IIA Components on Test Bed	39
Figure 17: Components of IA	40
Figure 18: Components of Cloud.....	41
Figure 19: PrepareFile Step: Encryption Enabled, with IA, SL.....	43
Figure 20: PrepareFile Step: Encryption Enabled, with IA, MHT	44
Figure 21: PrepareFile Step: Encryption Disabled, with IA, SL	45
Figure 22: PrepareFile Step: Encryption Disabled, with IA, MHT	46
Figure 23: Integrity Challenge Response Step, MHT	47
Figure 24: Integrity Challenge Response Step, SL	48
Figure 25: CheckIntegrity Step, MHT	49
Figure 26: CheckIntegrity Step, SL	50
Figure 27: PrepareFile Step (2048Mb File on IA Enabled System).....	51
Figure 28: Detailed Flow of System	B1

LIST OF ALGORITHMS

Algorithm 1: Pseudocode of Creation of V	D1
Algorithm 2: Pseudocode of PrepareFile and PrepareBlock	D3
Algorithm 3: Pseudocode of CheckIntegrity	D3

LIST OF TABLES

Table 1: HDD Benchmark Results.....	37
Table 2: Comparison of Previous Schemes	A1
Table 3: Test Results for PrepareFile Phase.....	C1
Table 4: Test Results for Integrity Challenge Response	C2
Table 5: Test Results for CheckIntegrity Step	C3

LIST OF EQUATIONS

Equation 1: Challenge with input file or block.....	13
Equation 2: Equations to set right and down nodes	28
Equation 3: Equations to determine high and low values of right and down nodes..	28
Equation 4: Generic Hash Function	28
Equation 5: Hash calculation of node.....	28
Equation 6: Definition of File.....	29
Equation 7: Multi-party Diffie-Helman Key Exchange	29

LIST OF ABBREVIATIONS

CRUD	Create, Read, Update, Delete
P2P	Peer to Peer
NFS	Network File Systems
CSP	Cloud Server Provider
CS	Cloud Server
IDE	Integrated Development Environment
PDP	Provable Data Possession
CHAL	Challenge produced by CU
CU	Cloud User
DPDP	Dynamic Provable Data Possession
CPDP	Cooperative Provable Data Possession
HomT	Homomorphic Tags
MHT	Merkle Hash Tree
SL	Skip List
IA	Integrity Auditor
VA	Validation Agencies
CHAL	Challenge
IIA	Independent Integrity Auditor
IAaaS	Integrity Auditor as a Service
CIAaaS	Compound Integrity Auditor as a Service
s_k	Secret/Private Key
p_k	Public Key
IAM	Integrity Auditor Manager
IAT	Integrated Auditor Thread
IADB	Integrity Auditor Database
IAFE	Integrity Auditor Front End
MFT	Master File Table
B_i	Block of file
T_i	Metadata of i th block
H_i	Hash of i th block

CHAPTER 1.INTRODUCTION

1.1 Cloud Computing and Security Aspects

Internet is getting more popular after people started to meet with Web 2.0. Web 2.0 is a technology that provides user to publish their content over internet without using HTML directly [1]. Nowadays, even 3 years old children are able to use tablets and computers. People got used to 3g and smartphones, it also centralizes internet in our lives. On one hand, increased usage of smart phones provides us mobility to access information. On the other hand, even that phones are getting stronger, smart phones have limited storage. In addition, since people may access internet by their mobile phones, they do not want to carry every needed information on the phone. Therefore, another technology "cloud computing" welcomes to our life. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be provisioned rapidly and released with minimal management effort or service provider interaction [2].

Cloud technology is welcomed by popularity. While social media occupies more block of life, people started to use cloud technology vaguely. Almost everyone has Facebook or Twitter account; iPhone users use iCloud to store images, people upload documents to GoogleDocs, share photos over Instagram, using Dropbox to access or transfer files between their devices and sharing videos over YouTube. People use that applications or websites with their personal computers, tablets and smartphones. Once the information is uploaded to an account, it is accessible over any device connected to internet just after being logged into that account. Even, friends, co-workers or family members can access those resources.

Companies work on projects to migrate cloud; companies looking for advisors to use cloud and hiring developers to publish cloud applications. By way of open

source cloud projects, famous companies like Amazon, Google and Microsoft makes investments for cloud technology to acquire corners of cloud technology.

Figure 1 [3] shows the general architecture of cloud. Cloud architecture is divided into two parts. Front end and Back End. Front End part includes whole infrastructure of clients. Actually, that part includes any device or medium that serves user to connect cloud over Internet. At the Back End side, there are many components, which shape body of the cloud. Management and security layers must cover rest of the layers since security is important aspect over all of the other layers. Security layers include tools and mechanisms to ensure security of cloud. Management layers are needed over rest of layers for high configurability and manageability. By using management tools, many things can be managed like users and services. Infrastructure layer is consisting hardware components, operating systems and middleware software that run behind cloud system. Storage is the layer, which provides the ability to store data over cloud. It can include disks and databases. Over infrastructure and storage layers, service and cloud runtime layers runs. Cloud runtime includes bones of the cloud. All the cloud business rules and systematic implemented under that layer. Service layer includes cloud services while application layers consists other applications that runs on the cloud.

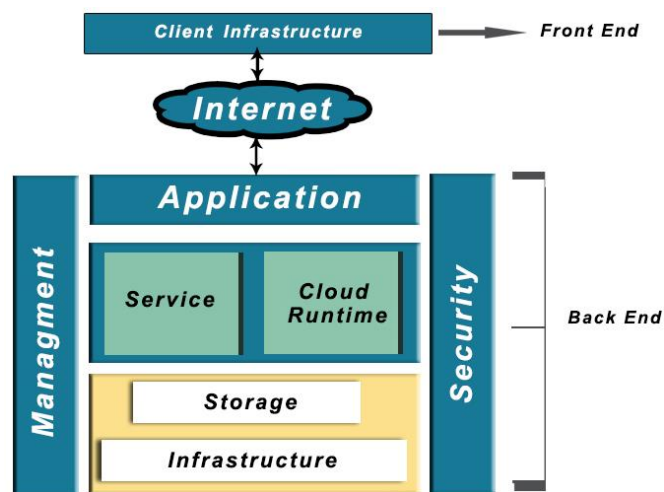


Figure 1: General Cloud Architecture [3]

Depending on requirements and funds, many cloud models have been introduced to satisfy needs of customers. These models are related to two perspectives: Service model and deployment model.

As can be seen in the Figure 2, regarding to service model, clouds can be categorized into three classes:

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS).

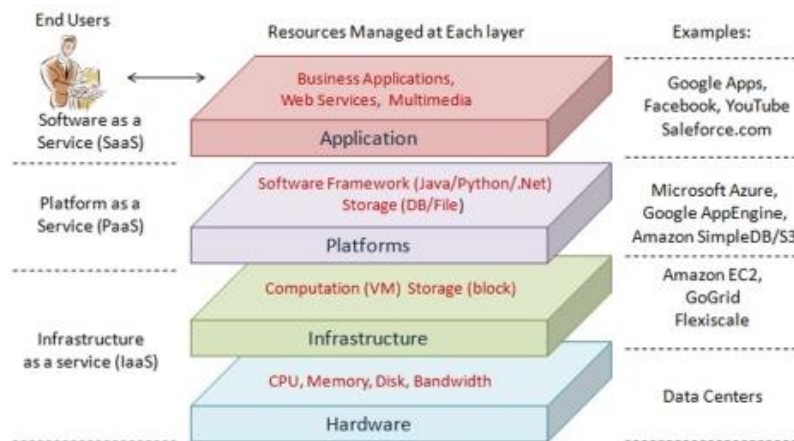


Figure 2: SaaS, PaaS and IaaS [3]

In the IaaS model, cloud consumers directly use infrastructure components (storage, firewalls, networks, and other computing resources) provided by the cloud provider. Virtualization is widely used in order to provide physical resources in an ad-hoc manner to meet current resource demand of cloud consumers. This cloud model, is generally being preferred by the companies in order to manage whole system. Generally, a virtual machine is being isolated from underlying resources which is offered to customer; and customer can use whole operation system to satisfy needs.

PaaS can be classified as the cloud model that offers application development. Its features include programming languages, integrated development environments (IDE), frameworks and programming models and various persistence layers [4]. Generally, IDE is isolated from operating system and other underlying resources

offered to cloud user. That IDE supports full of software development life cycle; also, application server, configuration tools and so forth.

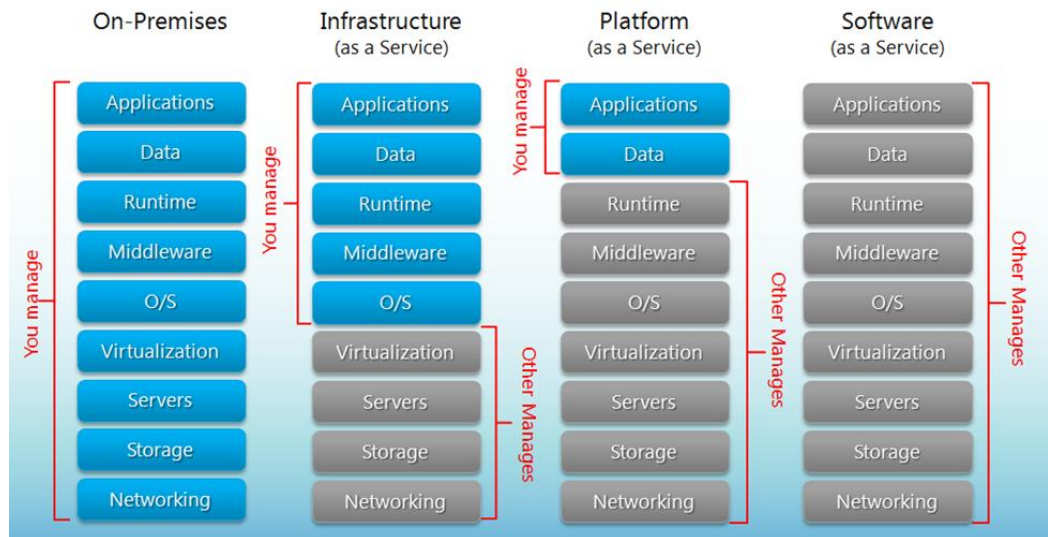


Figure 3: Separation of Responsibilities on Cloud Models [5]

Software as a Service (SaaS) is a platform that is various services deployed on hosting environment and offered to clients from various clients [6]. Only the endpoint for the service is offered to user and user accesses to end point using various clients (like web interface, smartphones etc.). Rest of the cloud system is fully isolated from cloud user and cloud user is only able to use that endpoint.

SaaS includes applications like GoogleApps, Facebook and Salesforce.com. PaaS includes platforms like Microsoft Azure, Google AppEngine, and Amazon Simple DB/S3. IaaS includes infrastructure and hardware like Amazon EC2, GoGrid, FlexiScale and data centers.

Responsibilities of both CU and CSP change related to cloud models, as seen on Figure 3. Whole of responsibilities are burden on user if subject is On-Premises. In IaaS, part of that burden switched on to the CSP. User manages applications, data, runtime, middleware even operating system while rest of responsibilities like virtualization, servers, storage and networking belongs to CSP. In PaaS, users are only responsible of applications and data while rest is burden of the CSP. In SaaS, user is

responsible of nothing. Namely, user only consumes services. Nevertheless, rest of responsibilities is being managed by CSP.

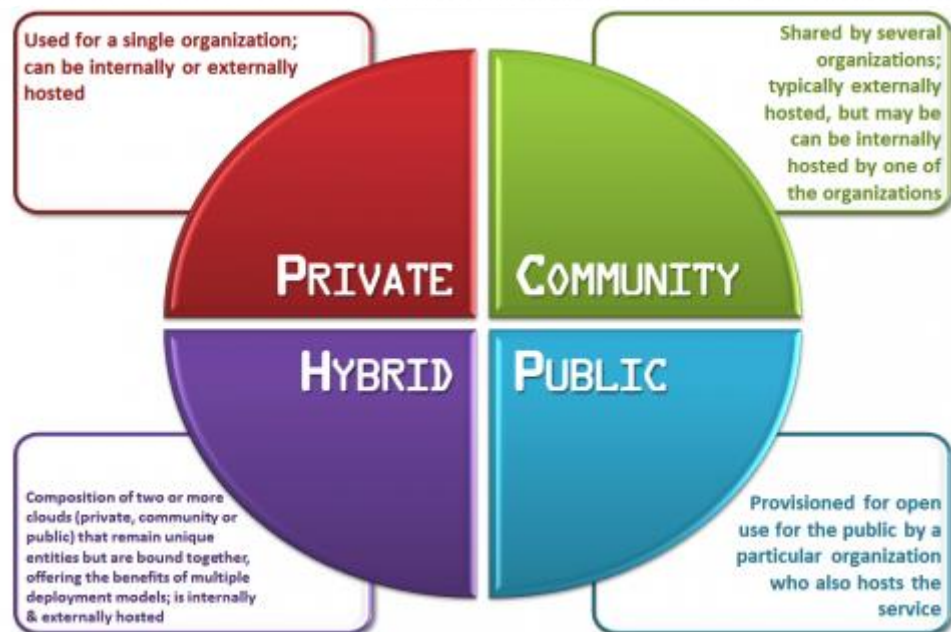


Figure 4: Types of Cloud Computing aspect of deployment model [7]

Aspect of deployment model, cloud systems are categorized into three classes:

- Private cloud
- Public cloud
- Hybrid cloud.

As can be seen on Figure 4, these three classes have differences. Private clouds are generally being used for single organization; but somehow can be served over internal network or external network. Community clouds are intended to be shared among several organizations. Even community clouds are served on external networks, one of that organizations can serve cloud in its internal network. Public clouds are served for open use for the public by a particular organization. Generally, public cloud is hosted externally by that particular organization. Hybrid clouds are mixed flavor of both private, public and community clouds. That flavor is achieved by composing private clouds, public clouds and community clouds; however, all of composed clouds are unique entities itself but bound together.

Even cloud systems are having widely usage area on our life; there are many inquiries about security of cloud systems. Even some standards have been defined related to cloud security, this subject is still open to question. The nature of cloud systems already makes the systems vulnerable to most of attacks. At this point, while evaluating the scale of security of cloud system many points must be taken into account. If the weakness of informatics laws reckons too; lots of users and companies that are eager to use or migrate into cloud, starts to be chafed. Lastly, also more important than the others: Since black-market trends (popular viruses and malwares) are related to popular systems, lots of black hat hackers are wringing their hands to breach the opaque but not that strong walls of cloud systems.

Before mentioning about storage security, importance of information must be understood. Information security is important for any platform. It must be planned and applied for various cases and platforms. Information is valuable thing and transformation between data to information needs processes. Information is money. Both data and information must be stored securely and must be protected being beware of unauthorized hands. If a company's secret product details have been revealed to rival company, that company loses both prestige and lots of money. If a bank's credit card archive has been stolen, that means bankruptcy of that bank. If a government's intelligence team have been revealed in another country while on operation, it becomes catastrophic situation. Information is valuable thing. That is why personal details of people being sold to various customers in return for decent prices. That is why information has secrecy levels.

There are few key concepts of information security. According to IAS-octave that has been proposed (also evaluated by lots of authorities) as extension of CIA-triad [8]. These concepts include

- Confidentiality
- Integrity
- Availability
- Accountability
- Auditability
- Authenticity

- Non-repudiation
- Privacy

Most important three concepts of that list will be discussed briefly, one by one.

Confidentiality is a concept that any information is being available to only authorized users or parties. To provide confidentiality, authentication and authorization are required steps and these steps will be mentioned. Unauthorized access to any information must be prevented to provide confidentiality. Any party which can able to access data must be authenticated and authorized for that data. Confidentiality is like read permissions of files. Only authorized parties or users must able to access that information. To provide confidentiality, data encryption methods can be used.

Integrity is a concept that any information is being accurate and complete over its life-cycle. Likewise, integrity and confidentiality must not be dizzied. Same user or party can have different read and write permissions on same file. On the other hand, same information can have different permissions for different users or parties.

Availability is a concept that any information is being available when its needed to authorized parties. Any unavailability must be totally avoided. To provide this availability; storing systems must work as intended; communication channels must be working and secure; and security controls that protects information on any level must be working.

Cloud security is another important subset of information security. Since data has been stored on various servers separately; extra cases for security of the data must be taken into account. One of those servers can be malfunctioned, disabled by attacks or dirty hands can obtain whole information on server. This makes cloud data security more noteworthy. At this point, all IAS-octave concepts must be taken into consideration. Confidentiality case is very important; since, many users are using cloud system and every user's data hold on cloud separated over servers. This situation makes access control privileges crucial, since any user does not want any other user to access his/her information unless he/she purposely intended to share that information. Integrity case is another important subject. Availability is another important point.

These three cases are most important point of view for cloud security. Moreover, fragility of cloud security makes these cases more important. That is why replication or backup technologies are having more important role on cloud to keep data integral and available; also authentication mechanism to keep data confident.

While storing data on untrusted servers, it is very important to verify authenticity of data. Since servers are untrusted, it is mandatory to verify the authenticity of data. That problem is being conspicuous while discussing P2P storage systems [9], NFSs [10], object stores services work on web and databases [11]. Integrity problem becomes more obvious while size of data is large. Even for small size of data, it is soluble; while size of data increases, cost of accessing data and transmit cost over network makes that problem serious [12]. To handle this problem, a waypoint must be implemented in life cycle of integrity check of big files. In this respect, a solution without transmitting the big data and without accessing entire data must be applied. Thus, a solution must be applied which is not dependent to size of data.

On the other hand, both transmitting big data over network and storing as one block can be thorny because of interruptions and corruptions. Besides, due to limited storage size, it would be beneficial to split data. Also storing data blocks at various servers is nice practice. However, splitting data into blocks and transmitting those blocks is desirable on account of bandwidth limitations too [13].

In a cloud system, distributed file system is used for allowing many clients to access on same file by statutory operations like accessing, creating, updating and deleting files. Each file uploaded to cloud can be split into files and can be stored on different cloud servers. Since those clients must able to operate on the files, which uploaded to cloud, there is massive access cost in a cloud for accessing to disk [14]. Even, this situation can be handled by lots of methods, the architecture of that system is critical; since, architecture heavily influences performance and throughput of the system [15]. Besides, any method also must ensure both security of system and security of files.

In 2010, CSA (Cloud Security Alliance) listed “Data Loss or Leakage” as 5th important threat [16] from 2010 to 2013. Further, “Data Loss or Leakage” is being split into two new items in Notorious Nine [17]. Even CSA did not release newer version of this document, these threats are still must be considered; since these threats are still impends cloud systems. According to that document, top two of nine threats - which mentioned above- are related to data security. Both of threats are relevant to all models of cloud. “Data Breaches” includes unauthorized hands to access confidential data in the cloud. This situation causes to information disclosure. Second serious threats are “Data Loss”. Even this threat includes any type of data loss (like destroying data, accidental deletions, disasters, forgotten encryption keys), destruction of the data by unauthorized hands is security aspect of the threat. “Data Loss” is about data’s availability and non-repudiation.

Even cloud providers are claiming to provide key concepts; many people still hold suspicious overlook at storing important data over cloud. This mistrust forces people to prefer external hard drives to cloud storages. By this way, people feels more comfortable and secure. Because, data stored inside external hard drive is physically secure; since owner can carry or hide that hard drive. In addition, if hard drive is not plugged into any computer, it cannot be read or modified by anyone. On the other hand, owner can easily plug that hard drive into computer and access those files. By this way, confidentiality, integrity and availability is provided by the owner of data. Nevertheless, this choice has lack of portability and mobility. To provide more users to courage about using cloud, users must feel more comfortable than using their external hard drive.

On a cloud or distributed storage system, data are split into blocks and scattered over whole storage servers or cloud servers. It makes imperious to check integrity of every block of data to ensure whole data is integral. This operation can be done by both CSP (Cloud Server Provider) and CSs (Cloud Server) by sending feedback to CSP about integrity of possessed data by each CS. Implementing second way means being wiser. Because system does not spend resources for transmission, which to transmit data between all CSs and CSP.

Integrity is a property that is independent of size of the data. The method used to check integrity of a block is same with the method used to check integrity of a file. Hash algorithms verify integrity; and time needed to calculate hash of a data is only relevant to size of data. However, even data split into blocks, every block must be integral itself to prove integrity of whole file. In other words, if all blocks of data are integral itself, it means file is integral too. Hash algorithms are smart executions that provide output by accessing block of a file or entire file. First hashing must be done while file is integral; to provide base result for integrity checks. Since there is base result of hashing, other hashing operations can be compared with base hash results. If no difference between hash output and initial hash result, it means file is not changed. However, hashing algorithms' output will change even if single bit of block of file or file has changed; even if two bits swapped on block or on file. Thus, hashing algorithms are very suitable for integrity checking operations. To check integrity of folder or file (which consists blocks), few methods can be used. Obviously, as a first step, hash of each file (or part) must be calculated. The difference reveals at second step. As a second step, there are two different methods. Hashes of files (or blocks) can be combined into one hash by appending or using XOR for all hashes. Appending many hashes is not smart solution even this method is aware of integrity. By using this way, fixed sizes of hashes are changed; also, it is size dependent solution, which is not ideal for files (or blocks). Besides, this system is only aware of monolithic integrity. Alternatively, hashes of files (or blocks) can be used to create more complex structures like array lists or linked lists. This method can provide corrupted file to be determined. However, it is costly to check all files' or blocks' hashes. At this point, another method that can be assumed as hybrid of that two methods. One monolithic hash is calculated with hashes of each blocks or files. After general integrity of folder (or files) is checked, if any integral problem has been determined, each block's or file's hashes can be checked. That method is the right thing for checking integrity of folders (or files formed by blocks). To achieve this method, multi-level lists or trees are best-suited data structures. Multi-level lists and trees will be compared and discussed respect of checking data integrity in CHAPTER 3.

Provable data possession is a term about integrity of stored data on remote servers. When user uploads a data to remote server, it means that server possesses that data. While CU wants to be sure about integrity of the possessed data, CSP must ensure about possessed data integrity. At this point, user challenges server to control possessed data at server side and server replies. By checking reply sent by server, user decides if data is integral or not. That entire cycle is named as PDP [18].

At most basic level, CU is creating some flags and adds that flags particularly to file while uploading the data. Few of flags are being stored at CU side to check integrity of the file by comparing those flags with response is being sent from CSP. Flags sent to CSP by CU must be enough to generate flags. CU must able to check anytime, that if data is integral or not. When user initiated a process to check integrity of the file, CSP starts an execution to create response. CSP creates metadata from scratch by accessing file. After CSP sends generated metadata as a response for CU's request, user checks equality of results sent by server and metadata, which is stored at CU's local. If the result sent by CSP equals to metadata stored locally, it means data is integral on cloud.

For the single file, only one hash is needed to create metadata. For blocks of file or a folder, initial metadata must be created for every blocks of file or every file.

Simple implementation of that method [18] is nice and naive but there is probability for unauthorized hand to access file or block. First of all, this probability must be minimized. Unless user exclusively states that file or block was public; that data must stay private. Somehow, access of any other parties except exclusively permitted to access that file or block must be hindered. Furthermore, files or blocks can be stored after files become encrypted. By way of addition, a control mechanism can be established as a layer between CU and CSP or secure tunnel can be used to transmit file and responses. Otherwise, an eavesdropper has ability to create hash of file after obtained original file. If CSP does not generate response from scratch by reading file, unauthorized access can cause problems that are more serious. This problem includes modifying file (or block) and hash of that file (and block). Modification on a data is an operation that changes integrity of the data. At this point,

if metadata is not generated from scratch every time, CSP cannot realize that if file or block has been changed. This is serious problem; and, if CSP is not cognizant about changed content of file, CSP sends responses malfunctioned. Worse, malicious code can be integrated into data. Especially, if the data includes executable blocks, problem becomes fatal. Attacker can grant unwarranted permissions over that server or CU's computer to get more control. This unexpected power can cause complete data wipe or more sensitive data being obtained by attacker. Attacker can delete that file by modifying metadata sent to CSP and cause data loss. To provide more secure implementation, a control mechanism to manage access rights over files on cloud must be exists. As an addition, to provide CSP to build response from scratch for every request, some kind of unique data like session keys must be added into request sent by CU. CSP must serve the response related to that unique data [19]. To provide singularity, this unique data will be mentioned as CHAL (challenge) at rest of the document.

It is regular situation people to upload text files or pictures to cloud. It is very common necessity people to change file, which is uploaded to cloud by themselves. Significant part of schemes offered to check integrity of cloud is not dynamic. Those solutions are not taking into consideration if files are changed or not. Those simpler and older solutions are generating hash of uploaded data on CU side, and sending challenge to obtain response from CSP. If user wants to modify files, which are uploaded to cloud, these schemes do not have enough flexibility. Before file is uploaded to cloud, initial hash at client side is calculated and CU only be able to check if uploaded file is integral on cloud.

At this point, another concept is greeting. This concept allows CSP to ensure integrity of the data, even data is being changed by CU. Moreover, it gives a capability to CU: Updating or deleting file. CU also has ability to update the file partially.

PDP schemes generally include few steps to store data and few steps to challenge and ensure data is integral and safe. Some of schemes provide extra or extended steps to ensure getting something valuable like, decreased computation on

CU side, minimizing communication complexity, providing better reliability and higher probability of detection.

Even schemes are different from others, main steps used to store and check integrity of data can be characterized like these:

- System Setup: Generally, in this step, CU generates necessary keys, creates hash for each block of file.
- Sending Data: Generally, in this step, file, keys and hashes are sent to CSP by CU.
- Storing Data: In this step, received file is stored on cloud.

Challenge: In this step, integrity of file, which stored on cloud, has been checked.

1.2 Literature Survey

Many provable data possession schemes have been offered until today. Few of them focused on client performance, few of them focused on reliability of PDP scheme, few of them tried to minimize overheads. Some of them based scheme on HomT (Homomorphic Tags), some of them based on SL (Skip Lists) and some of them used MHTs (Merkle Hash Tree).

Diffie-Helman based scheme is firstly proposed by Golle *et al* [20] which is based on a n-Power Decisional Diffie-Helman. In that scheme, Server restrain a data which has at least size of original file. Somehow, it is not necessary this data to be an original file which user stored at cloud. It also refers to Merkle Hash Tree scheme suggested by Wagner *et al*. [21] .

Deswarte *et al*. [19] have offered RSA based implementation for verifying data integrity stored at untrusted server. It is a solution, which still creates a base for other PDP solutions. This implementation is using two functions and keeping at least one of those functions secret. F is defined as a function with one-way hash H'. C is the challenge sent by user while R is the response sent back to CU by CSP.

$$f(C, H'(Input)) = H(C|Input) = R$$

Equation 1: Challenge with input file or block

Krohn *et al.* [22] have offered a scheme which uses homomorphic hash functions. Filho *et al.* [23] have offered another RSA based solution for verifying integrity of files stored at untrusted remote servers. This method is based on scheme of Krohn *et al.* and Filho *et al.* improved implementation of Krohn *et al.* by extending algorithm to support different sized message blocks. Filho *et al.* also consider that both uploaders and downloaders are untrusted. However, Golle *et al.* [20] assume that uploaders are trusted and downloaders are untrusted.

Schwarz and Miller have proposed another scheme [24], even the security proofs of scheme is not provided. This scheme verifies data between multiple sites over erasure-coded data. Erasure code is a method for data protection that data split into blocks, enlarged and encoded with duplicate blocks and stored along various storages. Both server side operations' complexity and communication complexity (between CU and CSP) of that scheme is linear. In addition, it makes that scheme less relevant to implement on real system because of bringing forth low performance on big files.

Sebe *et al.* [25] express a solution for integrity checking which sticks into Diffie-Helman problem over Z_N . Storage needed on client is $O(n)$ since N (size of RSA modulus) bits per block must be stored at client side. It makes this solution useful for block size larger than N . However, that needs server to access whole file to check integrity of file.

Batch verification adapted to use homomorphic hash functions for source authentication is offered by Yamamoto [26].

Another authentication scheme that is using homomorphic hashing in which generate single hash value is used for multiple blocks, is offered by Krohn *et al.* [27].

Another solution is offered by Oprea *et al.* [28]; which relies on tweakable block ciphers that allows client to determine if file block has been modified on untrusted server. That solution does not need any extra storage at server and client needs to store very low amount of data if entropy of data is low. Nevertheless, entire file must be retrieved in verification phase. Retrieving entire file on verification phase makes server file access cost and communication cost linear to size of file for any

challenge. Actually, that solution is invented for data retrieval and not useful for data possession verifying.

Erway *et al.* [29] and Ateniese *et al.* [12] simultaneously have offered PDP models that support update operations.

First sampling model for PDP is defined by Ateniese *et al* [18]. It can verify authenticity of stored file or blocks without needing server to retrieve and to access entire data. Furthermore, this scheme has improved PDP model to support CRUD (Create, Read, Update, Delete) operations. This model comes up with handling all future challenges at setup phase and populating metadata that includes pre-computed answers. However, model of Ateniese *et al.* is based on RSA scheme and it causes exponential calculate operations. On the other hand, inserting block into blocks cannot be handled with that scheme while only appending blocks are supported.

Erway *et al.* [29] have offered DPDP method which based on scheme offered by Ateniese *et al.* [12]. Erway *et al.* compared those two method with previous schemes. This model has been proposed closely related to memory checking; that supports update operations over stored data using skip lists, which has decent performance. They offer extra few procedures to prepare data for update, perform update on the data, verifying updated data. Different from previous work, they implement fully dynamic PDP first time. By using, rank based authenticated dictionary, logarithmic performance has been satisfied and that scheme named as DPDP I (Dynamic Provable Data Possession I). Other alternative scheme that named DPDP II has been constructed with RSA tree using rank based authenticated dictionary. Even DPDP II's computation cost at server side is high; it provides higher probability of detection. These models have been offered for a file, which stored on cloud server as n blocks. Dynamic PDP supports insertion of new block, updating block of stored file and deleting block of a file that stored on cloud.

Chen and Curtmola [30] have proposed a model based on spot-checking for both PDP and PDP supports CRUD operations.

Y. Zhu *et al.* [31] [32] have offered a CPDP (Cooperative PDP) hash index hierarchy and homomorphic verifiable response to provide better scalability of service and data migration between clouds.

L. Wei *et al.* [33] have studied to compose storage and computation aspects of security and privacy. By using verification agencies, storage and computation operations have been audited. MHTs (Merkle Hash Tree) has been used to hold hash values and they extend HDFS to compute tests. As their opinion, VAs are other party that trusted by CUs. In addition, VAs have far better computation power compared to CUs. Many complex operations that need high computation load and power needed to validate CHAL (challenge) results. Wei *et al.*'s solution needs CU to evaluate MHT (Merkle Hash Tree) and compares CHAL response with that tree. They introduce this idea to lower computation weight on CU side. Their method offers both parallel storage and computation over clouds servers. CU uploads their data to cloud and asks for computation. After data stored on CSs separately, each CS orders with some function to evaluate own data stored on CS. Various hashing operations and flags have been used and hashes of stored data have been used to populate MHT. After that functions run on each server with data, MHT has been calculated. CU gives delegation to VA and VA asks about data integrity to CSP. When challenge is ordered by VA, partial MHT is sent to VA as response. VA creates same MHT with various keys and controls if those partial MHTs have equal content or not. Furthermore, challenge response is sent back to CS.

Illustration of model can be seen on Figure 5. Even lots of components exist in cloud, related components are CSP and CSs. CSP is a front gate of the cloud. Any storage and computation requests sent to CSP from CUs and storage and computation responses sent to CU from CSP. Cloud users can use any device that can connect internet through related protocol. VA (Validation Agency) has role related to verification operations. After delegation of CU assigned to VA, verification request sent to CSP by VA. Results of the verification request received by VA and necessary feedback sent to CU by VA.

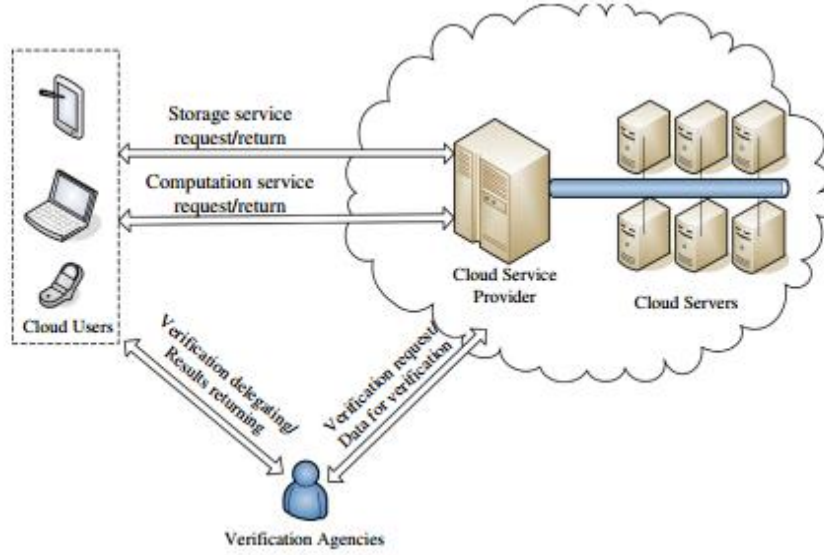


Figure 5: Verification Agencies

1.3 Motivation

There are many studies related to cloud storage security; however, very small part of those workings have focused on security aspect. At this point, this thesis mostly focuses on security aspect of cloud storage without losing a focus on performance aspect. Besides, IA (the third party auditor that audits the integrity of files which stored at cloud) is offered for making users' life easier and providing extra security.

At this point, this thesis focuses on three different implementation aspect of various approaches. Our model relies on breakthrough schemes offered by Ateniese *et al.* [12], Erway *et al.* [29] and Wei *et al.* [33].

Scheme offered by Ateniese *et al.* is a state of art article, which is the base idea for many of other schemes and studies. Erway *et al.* have improved Ateniese *et al.*'s work and offered PDP scheme, which supports CRUD operations. Wei *et al.* offers another scheme focused on secure computation and storage over cloud using verification agencies.

At this point, this thesis aims to offer a scheme that both includes verification agencies and fully supports CRUD operations. Besides, this scheme aims to offer

various improvements for PDP scheme that supports CRUD operations. Further, third party auditor inside PDP scheme is aimed to be integrated into scheme. Furthermore, test results of adding verification agencies, using Merkle Hash Trees or Skip Lists; also, effects of different file and block sizes over PDP scheme is aimed to be tested, analyzed and analyzed.

1.4 Outline

In CHAPTER 2, different approaches of PDP will be presented related to design aspect. Furthermore, in CHAPTER 3, implementation details will be discussed. In CHAPTER 4, experiments will be analyzed with test bed and experiment results. Also, test results will be introduced, imposed and evaluated. In CHAPTER 5 conclusion and future work will be mentioned.

CHAPTER 2. DESIGN of SCHEMES

Since cloud is living structure that users share, modify and delete their files on; supporting these operations on cloud is crucial. In addition, integrity of data must be ensured while these operations are being processed and after these operations being completed. We aim to provide a service interface and parallel data integrity check done by each of CS over the data that servers possess. Further, we aim to take data integrity checking burden from CU to another third party. Also, we have an idea to offer common interfaces for services which checks integrity of data and making their scheme valid for multi cloud environment. If these common interface become supported by standards, it can be useful hence increased privacy for users. On the other hand, cloud providers can be audited about privacy performances related to integrity aspect.

2.1 Definition of Integrity Auditor and Purpose

Integrity auditor is a mechanism to check integrity of files uploaded on cloud. First, user must not be busy with such a technical security problem. Regarding to user, if security and integrity of the data uploaded to cloud is provided by CSP; it is enough. If security and integrity has been provided, user would not concern about their data. On the other hand, it must not be necessity that user has technical knowledge to understand integrity of data. Some curious users like computer engineers or technology gurus or cybersecurity specialists keen on technical terms but significant part of cloud users don't have technical knowledge. If the user doesn't have technical knowledge or if the user isn't familiar with technology; it's a burden for him to check and understand if data is integral and secure. User wants to ensure about security of data; not more. Another aspect of view is about intervals between that checks. If any critical information is stored on cloud, it may be needed to check integrity and security of the data more often. In addition, user does not have to lose time for checking integrity and security of the data that often.

Another aspect is about power needed to make a computation for checking. As stated earlier, if complex computations are needed at CU side. It would be logical CU to delegate this duty to another powerful party that can evaluate those computations much faster.

Another important thing to be considered is that if cloud providers forced to check integrity by third party, this will cause cloud providers to yield this subject more tenderly. At this point, some standards are needed and these standards can be invented inside ISO standards. By doing this, standards may force cloud providers to use integrity auditors. By this way, every single cloud provider will be forced to focus on keeping integrity of users' data. If some standards have been adapted about using this kind of mechanism, legal sanctions can be applied to cloud providers that don't focus on integrity as intended. Furthermore, this may provide integrity auditors to work centralized. By this way, users can delegate to integrity auditor mechanism, and integrity auditors do complete job while users will not be busy with checking or thinking about data integrity.

We offer three model for integrity auditor:

- Independent Integrity Auditor outside cloud (IIA)
- Integrity Auditor as service inside cloud which is controlled by third party agency (IAaaS)
- Compound Integrity Standardized mandatory IAaaS for cloud providers which have same interface as a service (CIAaaS)

Three figures are given below to show general architecture for offered models. All of these models include IA at different and improved approaches. First one is including only external IA to check integrity. Second one is a service deployed inside cloud to make integrity controls. Third one is standardized model of second model that makes possible to check integrity controls over few clouds.

2.2 Design of IIA

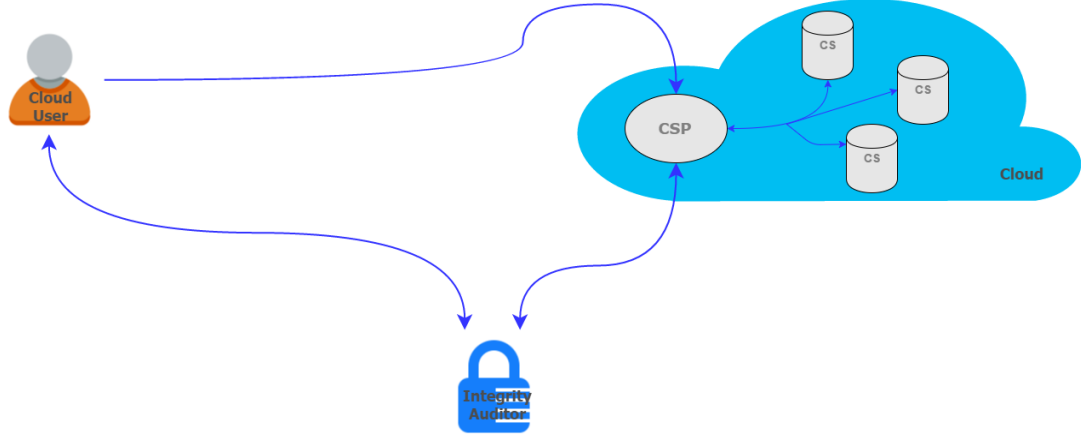


Figure 6: IIA

This model is simplest IA model that runs outside of cloud. IA is a third party application that acts like a bridge layer between CU and CSP. CU delegates to IA and IA ask CSP about data integrity. After some evaluations and calculations, IA controls CHAL response sent by CSP and determines if integrity is ensured or not. In this model, CSP checks integrity of the data directly; without giving an endpoint as a service. As foresaid, it is not needed to deploy extra service on cloud; but a business flow must be implemented inside CSP business flow. This flow sends a response to CHAL by performing integrity check on data, which is uploaded to cloud. Main disadvantage of this approach is having no available service for third party users outside of cloud. To put in another way, it is like internal flow between CSP and IA to validate data integrity. Detailed flow of this model can be seen on Figure 6. Data storage request and responses occur between CU and CSP directly. Data validation request is sent to VA, which is delegated by CU. Data validation request is directed to CSP and the response is received from CSP directed to CU.

2.3 Design of IAaaS

IIA is the simplest version for IA. It is noticeable that IIA has imperfections and IIA may be improved further. Since, CSP is already responsible for many things to manage. Processing high cost operations like integrity check would affect

scalability, availability and performance of CSP. So, the burden of CSP about checking integrity can be switched to another layer, which is only dedicated for checking integrity. General architecture of that idea can be seen on Figure 7. Data storage request and responses occurs between CU and CSP directly. Data validation request is sent to VA, which is delegated by CU. Data validation request is directed to IA by consuming service and the response is received from IAaaS directed to CU.

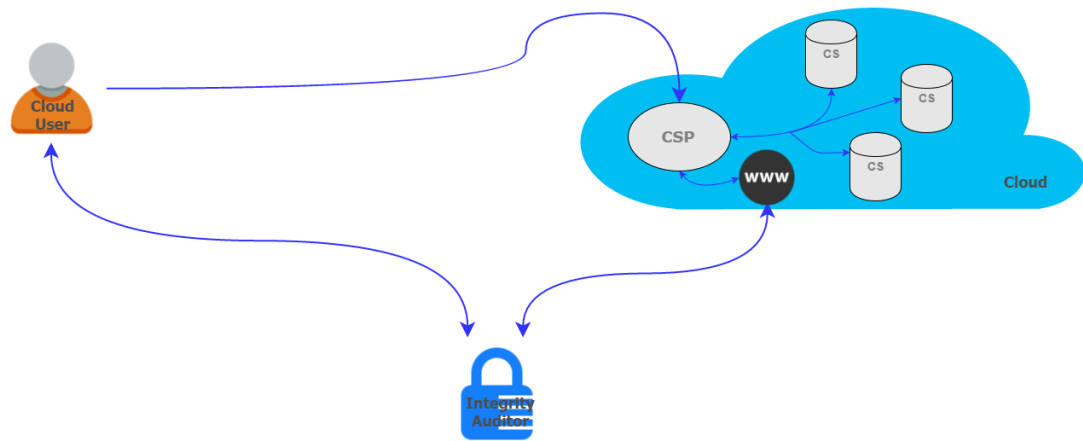


Figure 7: IAaaS

On the other hand, implementation of integrity check has low scalability since it is working as a part of CSP. It can be improved and obtain better usability by developing a service in. It would be easier to manage that service. If updates and maintenances of integrity check implementation taken into account, independent service approach is better. It is needed cloud to go offline if integrity check operations are implemented as a part of the CSP. While considering perspective related to maintenance aspect, service can be switched off to maintain and can be switched on again after necessary updates have been applied.

Furthermore, IIA is not generic solution since it is not same on all clouds; since it is a business inside CSP. Different programming languages, different designs, different architectures for different clouds. Cloud by cloud, business flow of integrity check is changed, CHAL requests and responses are being changed. This prevents the development of the suitable business workflow for integrity check for each cloud.

Somehow, if CSP is possessed by dirty hands; even, it seems like all cloud have problems; a lock mechanism can be developed to encrypt very critical data on cloud. By this way, data accessibility and integrity can be kept.

2.4 Design of CIAaaS

Last but not the least; after IAaaS become standardized; it means inputs and outputs of IAaaS can be served with same interface over all clouds. This favor provides an external auditor to manage data possession over more than one cloud. Another application or party can be developed that undertakes controlling data integrity of lots of data over lots of clouds. CUs can join that application or party by delegating their integrity check rights to IAs.

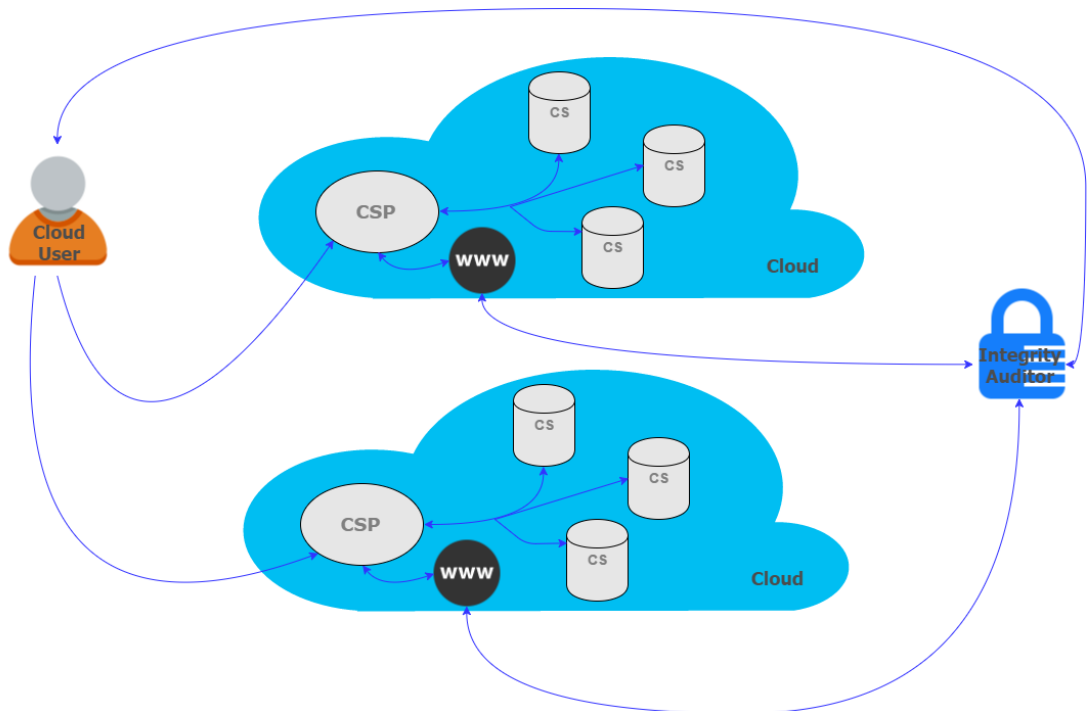


Figure 8: CIAaaS

As seen on Figure 8, data storage request and responses occur between CU and CSP directly. Data validation requests are sent to VA, which is delegated by CU. Data validation request is directed to IA by consuming service, and the response received

from IAaaS is directed to CU. The difference between IAaaS and CIAaaS is that all clouds may use same interface for data integrity checking services.

This standardization also makes easier to monitor integrity and security quality of the cloud providers. This monitoring mechanism enables some legal actions against cloud providers like, punishments and promotions. Some cloud providers that can't maintain integrity of definite ratio can be punished. On the other hand, some cloud providers that can maintain integrity of definite ratio can be rewarded.

CHAPTER 3. IMPLEMENTATION of SCHEMES

3.1 Background

To provide better understanding on validating integrity of block of files, Merkle Hash Trees vs Skip Lists will be discussed further.

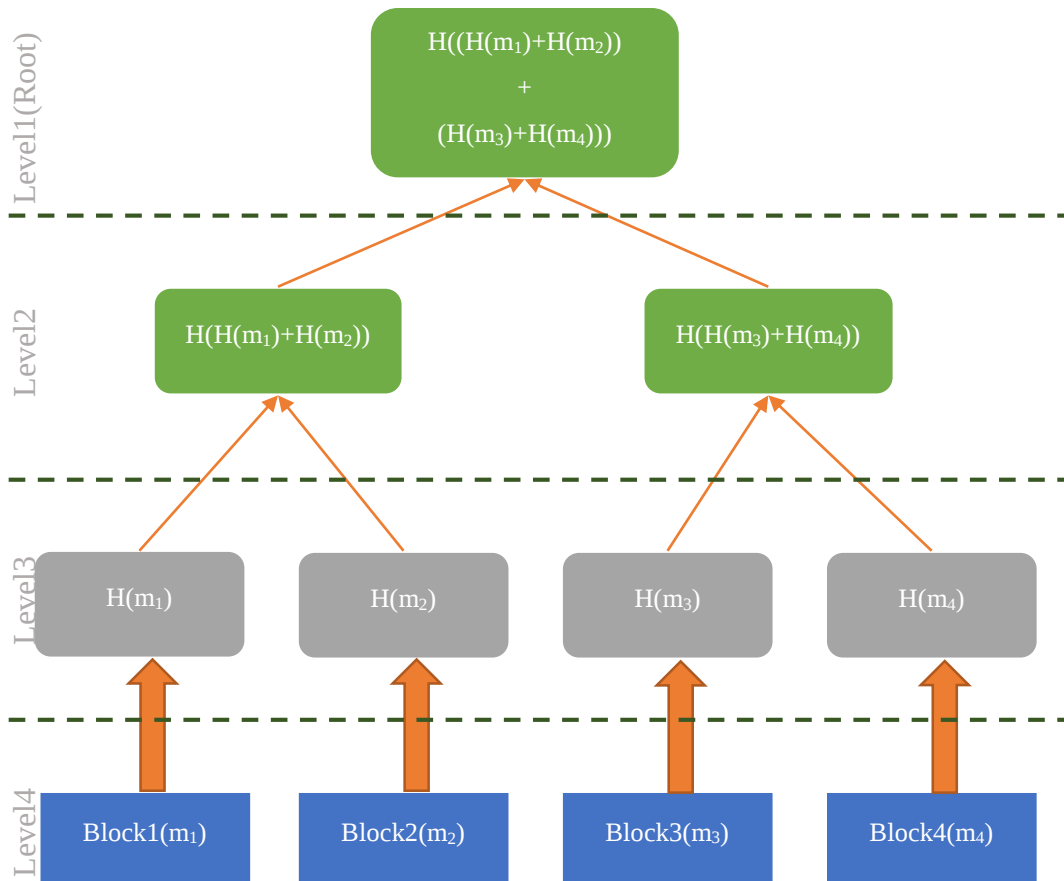


Figure 9: Merkle Hash Tree

Merkle hash tree is a structure that has hashes on nodes except leaf nodes as seen on Figure 9. MHTs provide secure and efficient way to check integrity of large files.

MHT has bottom to top approach while being constructed. To construct a MHT, as a first step, leaves must be constructed. Leaves include many blocks of a file. These nodes are constructed using hashes of each block and can be seen as Level4 nodes on Figure 9. As a second step, hashes of the blocks must be generated using $NodeValue_{leave} = m$ and $NodeValue = \bigvee_1^n ChildValue$.

Generated hashes of blocks can be seen on Figure 9 as Level3 nodes. To create higher level nodes, for every node, values at child nodes must be concatenated before hashing as seen as Level2 nodes on Figure 9. This operation finishes when root node has been generated.

To evaluate an integrity of a file by MHTs; MHT for integral file must be constructed using blocks. This MHT creates a base for integrity checks and it must be stored as metadata. When integrity check process starts, up-to-date MHT must be constructed using up-to-date file. Equality of root nodes determines if file is still integral or not. If root nodes are not equal, it points that; file was not integral. The advantage of MHT is that; it supports results with corrupted data. In other words, corrupted block can be determined by chasing MHT from root to bottom nodes. If node has same value with the node, which is at same location on base MHT; it means that blocks under that node were integral. On the other hand, child nodes must be checked until unequal value being found. If no inequality has been found until reaching to leaves, it means file is integral.

Security of this scheme is related to security of hashing functions. In best case, transmitting root node is enough for integrity check. If further controls needed, bottom of the tree can be transmitted. To check integrity of a single block, $\log_2 n$ of hashes needed [34].

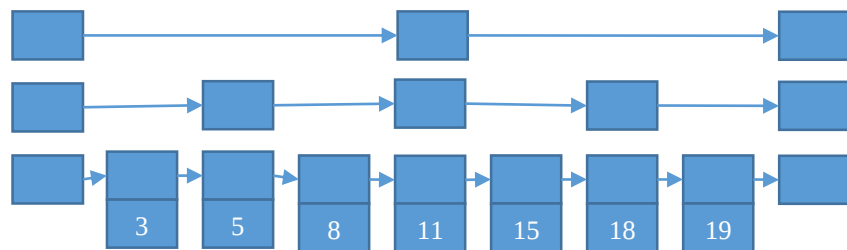


Figure 10: Simple Skip List

Skip list is another option to check integrity of blocks of files [29]. A simple version, which includes few numbers, can be seen on Figure 10. Skip lists can be defined as optimized version of linked lists. Using linked list, insertion/deletion operations are easy, however searching in less than $O(n)$ is not possible. Skip lists have $\log n$ complexity for basic operations like inserting, removing and searching [35] since every non-leaf node have two pointers that points another two nodes. It also seems like garbled sorted multi-level list. Basic operations like insert, update and delete are affecting only nodes on the search path. After these operations, ranks are re-computed bottom nodes to top nodes. Ranks will be mentioned further.

For better understanding the structure of rank based skip list, available pointers on structure (right and down), rank of node, low/high nodes of any node must be discussed.

Each node, n , has two pointers for search operations as seen on Figure 10, which points right ($rgt(n)$) and down ($dwn(n)$) nodes. The hash value of each node is calculated using these two nodes.

Rank is the number of nodes at bottom levels that can be accessed from any node. Let $low(n)$ represents the left most node at bottom level can be reached by node and $high(n)$ represents the right most node at bottom level can be reached by node. To clarify low/high nodes, an example using start node can be mentioned. Since all nodes must be accessible from start node; $high$ value of start node must be equal to the number of bottom level nodes. However, low value of start node must be equal to one.

High/low values determine traverse path over skip list. While traversing, low and high values of current node are checked and decided to traverse though left pointer or bottom pointer. To decide which node to traverse, high and low values of right and down nodes must be determined (Equation 3).

Traversing a skip list is iterative process. In each step, there are current node and next node. Current node (v) is the node which is pointed by pointer at time t while next node (n) is the node which will be pointed by pointer at time $t+1$.

$$w = rgt(n)$$

$$z = dwn(n)$$

Equation 2: Set equations to set right and down nodes

$$high(w) = high(v)$$

$$low(w) = high(v) - r(w) + 1$$

$$high(z) = low(v) + r(z) - 1$$

$$low(z) = low(v)$$

Equation 3: Set equations to determine high and low values of right and down nodes

According to Equation 2, next node to be traversed is being determined. If Equation 3 being satisfied, right pointer must be followed to determine next node; otherwise down pointer must be followed to determine next node.

To calculate hashes, more generic hash function is defined as Equation 4.

$$h(x_1 \cdots x_k) = h(h(x_1) || \cdots h(x_k))$$

Equation 4: Generic Hash Function

After generic hash function defined, hashing can be discussed. The node belongs to m_i (i-th message) stores $x(v) = T(m_i)$. $l(v)$ is the level of v node. $F(v)$ is the value of v node. This operation is marked as $F(v)$ and determined by using Equation 5. T is the path which is generated to prove integrity of a block. For further information, suggestion of Erway *et al.* [29] can be reviewed.

$$f(v) = \begin{cases} 0, & \text{if } v = null \\ h(l(v), r(v), f(dwn(v)), f(rgt(v))), & \text{if } l(v) > 0 \\ h(l(v), r(v), x(v), f(rgt(v))), & \text{if } l(v) = 0 \end{cases}$$

Equation 5: Hash calculation of node

3.2 Implementation of IIA

CU wants to upload file (F) which stated on Equation 6 into cloud. Block size is not generic and read from XML file for our implementation. A suggestion about dynamic block size will be mentioned in last chapter. CU sends that file to CSP and

informs IA about this situation by delegating to IA. IA checks the integrity of the file with intervals and informs user if any problem occurs. In addition, user can see IA's reports by client application installed on.

$$F = \{B_1, B_2, \dots, B_n\}$$

Equation 6: Definition of File

In the entire lifecycle of data, confidentiality, integrity and other important aspects must be considered and provided. In this scheme, integrity has highest priority. Even main aspect and highest priority is integrity; for better protection, other aspects must be considered while developing system. With other words, even goal is integrity, other aspects of data security is considered and evaluated while developing this system.

Before steps to begin, some agreements and exchanges must be executed. These steps will be named as initialization steps.

CU must be registered to CSP. When CU requests CSP to register, CSP creates ID for CU, which will be used as CU's identity. ID has length of 256 bits. This ID is proof of being owner of a file on cloud. On the other hand, this ID is integrated into packet which will be sent to CSP. To put it another way, whole blocks of file being signed by ID. Ownership of any file is determined by this ID; since CSP controls metadata of each block while decrypting file and preparing file to store on cloud.

CU must be registered to IA. When CU requests IA to register, IA creates A (ID of CU on IA) for CU. A has length of 256 bits.

After CU registers CSP and IA; three sides agree on a key. This agreement is done by multi-party Diffie-Helman key exchange (To protect against MiM attack, a PKI may be used). This key is named as K on entire flow. To extend Diffie-Helman key exchange between two parties to three parties; generalization of discrete logarithm problem must be utilized [36]. To achieve S through Equation 7, any party must send generated data to two parties, rather than just sending to other and only one party.

$$K = ((g^a)^b)^c = ((g^a)^c)^b = ((g^b)^a)^c = ((g^b)^c)^a = ((g^c)^a)^b = ((g^c)^b)^a$$

Equation 7: Multi-party Diffie-Helman Key Exchange

After CU becomes registered on IA, CU shares ID using K. However, CU shares his/her ID with IA for delegating his/her rights to IA. This provides IA to challenge CSP over files which uploaded to cloud by CU. IA also adds this ID to packet while sending challenge to CSP, to prove that possessing delegation is provided by CU.

Before CU to prepare packet which will be sent to CSP he/she requests to upload the file.

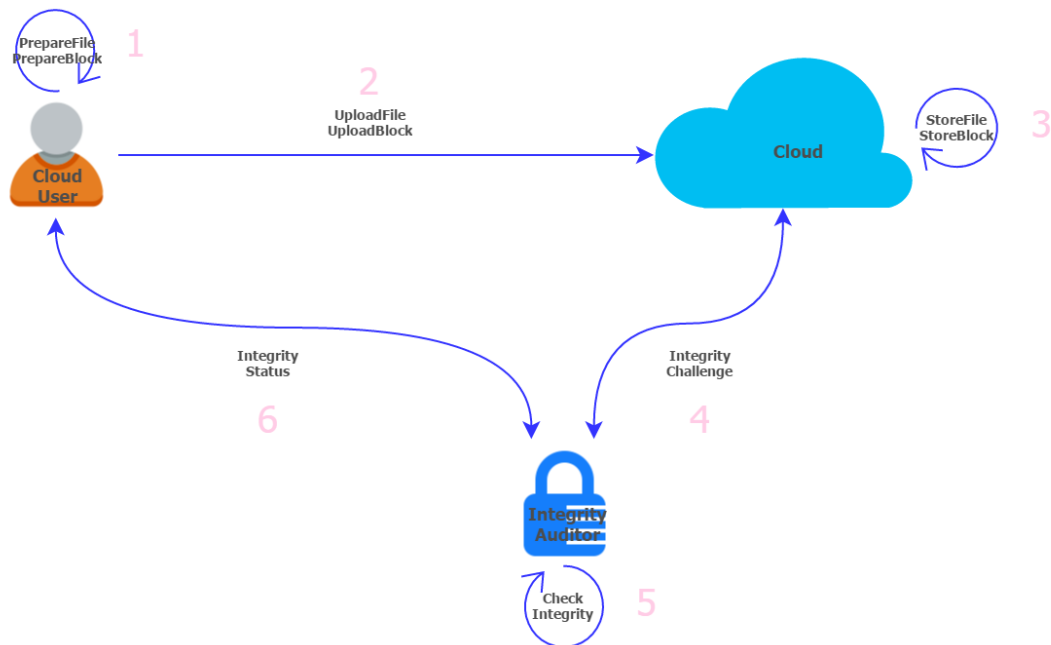


Figure 11: Flows on System

CSP generates S which is session key; and sends it to CU after encrypts S with K. S is also used as unique id of the file which proves the file is uploaded via this session.

PrepareFile: In this step, user creates packet by processing file. Hashes of blocks are generated and necessary parts are encrypted using K. Steps of this process are mentioned below.

Q_{ID} is generated by hashing ID by cryptographic hash algorithm such as SHA2. This data is appended to metadata of each block. Q_{ID} is the sign of the user which will be checked by CSP to ensure packet is totally integral and sent by CU.



Figure 12: Generating Q_{ID} by hashing ID

CU generates H_i (hash for each block of file, B_i) values. H_i values are obtained by using SHA2 algorithm. H_i values will be included inside T_i (metadata); also used to generate V (Validation data). V is obtained by storing whole H_i values on MHT or SL. Both MHT and SL structures are implemented in test bed to compare results.

T_i values are obtained by encrypting data by K which is obtained by appending i (order of block and operation type: 2 bits for operation type, 30 bits for block index), H_i values, Q_{ID} and S (session key). This metadata proves integrity and ownership of the data. After T_i is processed by CSP, CSP ensures if received block is integral and sent by specific user. Since S is generated for that session and file; also each block includes r (random number); replay attacks are being ensured.

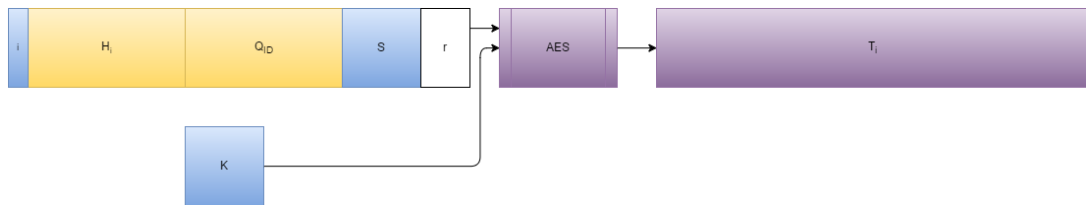


Figure 13: Generating T_i

P_i (Block of packet) is generated after T_i is obtained. P_i consists B_i and T_i . It is obtained by appending T_i to B_i .

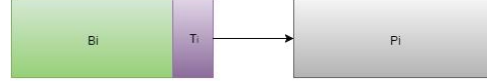


Figure 14: Generating P_i

To obtain P (Packet that will be sent to CSP), each P_i is appended after metadata and packet is encrypted using K . M (Metadata of packet) includes V , original file name and block size used in PrepareFile step. Metadata provides CSP to obtain block size and split packet to obtain P_i data. On the other hand, it provides file to be saved with proper name as intended user to upload with.

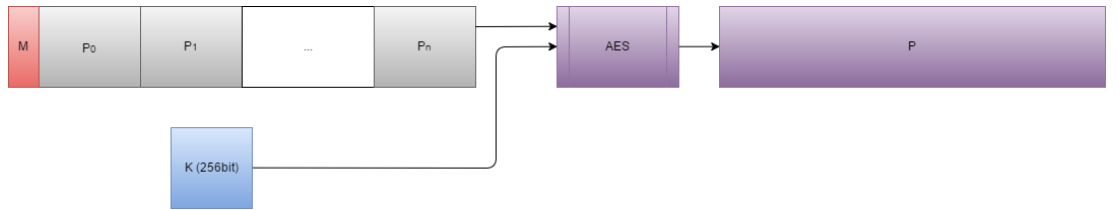


Figure 15: Obtaining P

B (Block size) has very important input about system performance. If B is being kept small, number of blocks is being increased; and, it causes ratio of metadata in packet to increase. On the other hand, cost of creating metadata for blocks are getting expensive.

More details about this situation will be mentioned in CHAPTER 4.

UploadFile/UploadBlock: In this step, the packet prepared by CU is sent to CSP. The reason behind both encrypting metadata and whole packet is that; if data content is not sensitive, only metadata can be encrypted.

StoreFile: StoreFile step is reverse of the PrepareFile step. In this step, CSP opens packet, controls hash of blocks, creates V and stores file on cloud. As first operation, whole packet is decrypted using shared AES key. Since any incremental schema is not being used, this operation doesn't take so long time. After packet is

decrypted by CSP, metadata of packet file and packets of blocks (as single part) are obtained. To obtain packets of blocks, single part must be split into blocks, using block size which included inside metadata.

Before file is stored on cloud servers, CSP re-calculates hash values for each file blocks. If any integrity issues are detected, CSP informs user about situation. If there are not any issues related to integrity of the file, CSP informs user. After response is received by user; if upload is successful, user sends a packet that includes V and S. This packet is encrypted by K using AES. If file will not be changed, user can delete V and F. If system supports modification and if any modification is planned, CU continues to hold V and F.

IntegrityChallenge: As mentioned previously, IA delegates CU. Since IA has ID and S, IA have sufficient rights and data to send C (Integrity challenge) to CSP. IA prepares a packet for this step. This packet is encrypted with K. C includes ID and S that encrypted with K by AES algorithm.

When CSP receives C, C is decrypted by CSP using K; then, ID and S pair is confirmed. If this pair is meaningful for CSP, F is checked and V_C (V that belongs to F stored on cloud) is populated. R_C (Result of challenge) is obtained by encrypting V_C with K using AES algorithm.

This step is executed by IA in custom intervals. File can be checked so often and so rare. This interval must be chosen wisely; since there is a trade of between network traffic and security. If file integrity is checked in shorter intervals, it may cause overhead on both CSP, network and IA. If file integrity is checked in longer intervals, corruption of file could be realized so lately.

CheckIntegrity: In this step, V stored on IA that belongs to F is compared with V which is received as a response for challenge request. A Boolean value R_I is obtained by comparing V with V_C . If content of V and V_C is equivalent, it means file stored integral at cloud servers ($R_I = \text{true}$); otherwise not. If file is integral, this

information is stored on IA with a timestamp. Otherwise ($R_I = \text{false}$), CU and CSP is somehow being informed.

IntegrityStatus: This step is initiated by user. When user wants to check integrity of the F; it creates a packet that includes A (Id of CU on IA system) S and N (Currentness flag determines if IA to request latest R_I or IA to send challenge to CSP). After this packet is encrypted by K using AES algorithm, I (Integrity check request of F) is obtained and sent to IA. When I is received by IA, it is decrypted by K using AES algorithm. IF A-S pair is meaningful on IA data, IA starts execution. At this point, N determines if latest CI data is sent to CU or if new C must be sent to CSP. If N equals to “Latest”, IA obtains I_R (Integrity check result) by encrypting latest R_I with K using AES algorithm. If N equals to “Actual”; CSP executes IntegrityChallenge step. In this situation, I_R is obtained by encrypting new R_I (just received from CSP) by K using AES algorithm. As a last procedure, I_R is being sent to CU.

PrepareBlock: Nowadays, lots of cloud storages are integrated with various editors. GoogleDrive and OneDrive offers office applications to edit files online. At this point, updating a file on cloud is not unordinary operation.

However, IA is implemented the way to modify files over cloud. Modifying file rather than re-uploading both saves time and cost. Even implementation with MHT supports modification of file by re-uploading, it is not best practice and it is not intended situation. V must be totally reconstructed because of structural design of MHT; skip lists are more dynamic structure to get down with that problem. Skip lists allows adding, modifying and deleting node operations.

In this step, a file that exists on cloud is being updated by adding, updating or deleting a block. The size of new block doesn't have to be equal with the previous uploaded blocks. Block size is important when storing a new file, since CSP splits blocks by using block size argument. Adding a new block and updating block operations are done independent from block size. CU prepares a packet which includes

same content similar to packet that prepared in PrepareFile step. This step can be called as PrepareBlock; since, it prepares only block of a file, rather than a file.

T_i (Metadata of new block) has been created with the same way in PrepareFile step. H_i (Hash of new block) is created using SHA2 hash algorithm. i , Q_{ID} , S and r are appended to H_i . To obtain T_i , this content is encrypted using AES algorithm with K as key.

To obtain P_n , T_n is appended to B_n for update and addition operations. For delete operation; different from other operations, B_n content is set as empty string.

Content of final package is near to be ready before sent to cloud. If content is not sensitive, CU can send P_n as final packet. If B_n includes sensitive information or CU wants to send packet encrypted; P_n is encrypted. To avoid replay attacks, metadata includes r . Even whole packet is sent without encryption, keeping T_n encrypted makes metadata hidden and provides security for unauthorized manipulation of metadata. On the other hand, if unauthorized party modifies the content of block; since metadata is not changed, CSP can alert the CU.

After hash of block has been created; V is composed again. If, the structure behind V is MHT; all tree must be re-composed. However, if the structure behind V is skip list, the nature of skip list allows that node to be added inside V , without re-compose.

StoreBlock: After packet is received by CSP; CSP decrypts the packet and obtains B_n and T_n . (as encrypted) CSP decrypts T_n ; then, obtains H_n , Q_{ID} and S . CSP checks if H_i is consistent with the B_n and if Q_{ID} is proper. If values are consistent, CSP starts to store block as a part of file which stored with S id. Indexes which include block storage addresses belongs to file are updated and file is stored with new form CS. When store phase is completed successfully, CSP informs the CU about situation. CU sends updated V to IA. Up to date V which belongs to S is updated on database.

3.3 Implementation of IAaaS

The difference between IIA and IAaaS is that IA communicates with service, which provides integrity results of files; rather than directly communicating with CSP.

Rather than CSP thread, another layer is needed to handle only integration related operations. General flow is same in this model, except IntegrityChallenge. IntegrityChallenge must be sent to servlet which undertakes checking the integrity of files and generating response for the challenge. Also, this model creates space to CSP for other management tasks; by reducing traffic over servlet.

3.4 Implementation of CIAaaS

Main focus of this implementation is generalizing service interface and providing IA to be informed about integrity of data, which uploaded into various clouds.

CHAPTER 4. Test Bed

This chapter summarizes the test environment, data collected and evaluation and comparison of algorithms.

4.1 Test Bed Environment

Computer Specs:

Operating System: Windows 10 Professional

Processor: Intel Core i7 3630QM (2.4Ghz, 8 cores)

RAM: 2 x 4Gb (DDR5)

Hard disk: Hitachi HTS72755 (500Gb)

Min	Avg	Max	Min	Avg	Max
72.4	Read 85.8	110	0.12	4K Read 0.41	0.58
69	Write 81.4	94.1	0.36	4K Write 0.91	1.27
68.4	Mixed 83.1	93.1	0.14	4K Mixed 0.18	0.22
62.9% 83.4 MB/s			53.6% 0.5 MB/s		
Sequential			Random 4k		
Read 79.4			4K Read 0.12		
Write 77.9			4K Write 0.36		
Mixed 68.4			4K Mixed 0.15		
56% 75.3 MB/s			28% 0.21 MB/s		

Table 1: HDD Benchmark Results

Read/Write speed of used HDD can be seen on Table 1. Values above headers include average benchmark results along other testers who tested same HDD. Values below headers are benchmark results of the HDD used for test bed.

4.2 Test Bed Restrictions

The test bed used to produce test results is not perfect to obtain best results of this architecture. Some restrictions which affects results of the test bed must be mentioned to make results more clear.

These test results are obtained using Windows 10. Windows is not highly configurable operating system that runs lots of processes and services at background. Various processes and services have been stopped while tests are conducted but not all.

All threads run on same PC. This means, all threads shares same resources. These resources include RAM, CPU and HDD. This is double-edged situation. Sharing these resources along threads cause lower performance when overlaps occur. On the other hand, by creating a test bed in one machine removes the effect of network and bandwidth.

As mentioned above, test bed is not perfect for ideal results. Even test scenarios have run 10 times to minimize instantaneous event effects, continual events' effects cannot be eliminated in that test bed. In addition, the computer is typical home laptop which is enough for daily use. To obtain ideal results of this architecture, various improvements can be realized.

To calculate results, the timer used inside code which starts and stops when first and last lines of related function have been executed.

4.3 Application Details

In experiments, a test bed has been conducted using Java. As a JDK, 8u77 version have been used IntelliJ IDEA has been used for development on Windows 10 Professional operating system. Ant scripts have been used to compile code, create jars and run the threads.

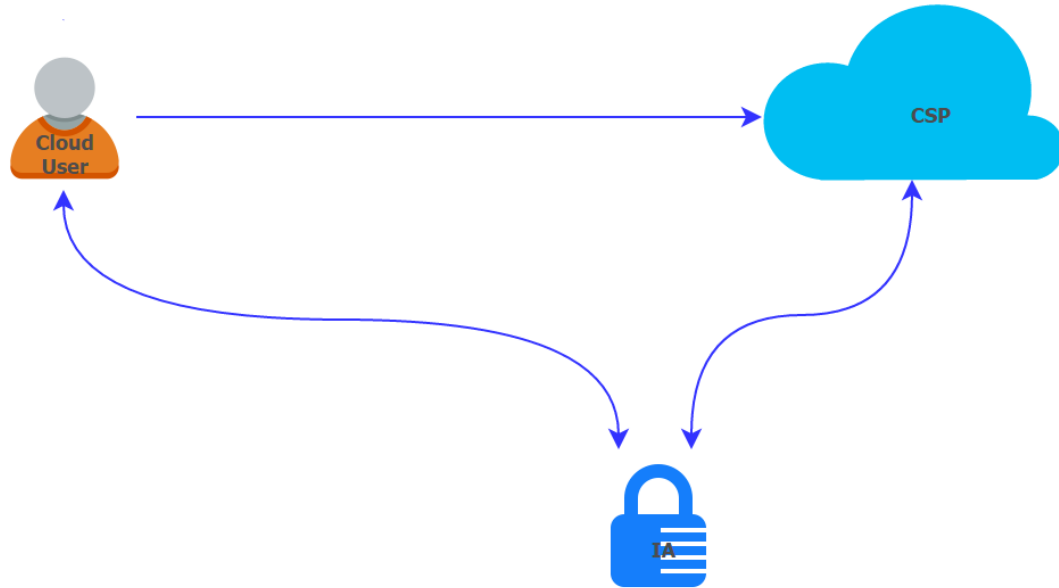


Figure 16: IIA Components on Test Bed

To compile test bed applications and produce JAR, two ant tasks are created. First ant task takes module path and compiles specific module. After this operation is completed, it moves batch files, configuration files and jars to specific folder, which all application outputs moved to. Second ant task calls first task with all modules' paths. After second ant task is called, all necessary batch files, configuration files and jars to same folder. Except all components drawn in Figure 16, there is Utility jar to hold common tasks shared between all modules. Besides, lib folder exists to store various third party JARs.

To hold various data (like user IDs, file IDs, keys) PGSQL has been used behind CSP and IA applications.

Test bed for the IIA includes these components as seen on Figure 16:

CU as a console application: CU has few operations. First operation is to register on CSP that assigns ID on cloud. Second operation is to register on IA that assigns A on IA. Third operation is to initiate key exchange that communicates with IA and CSP.

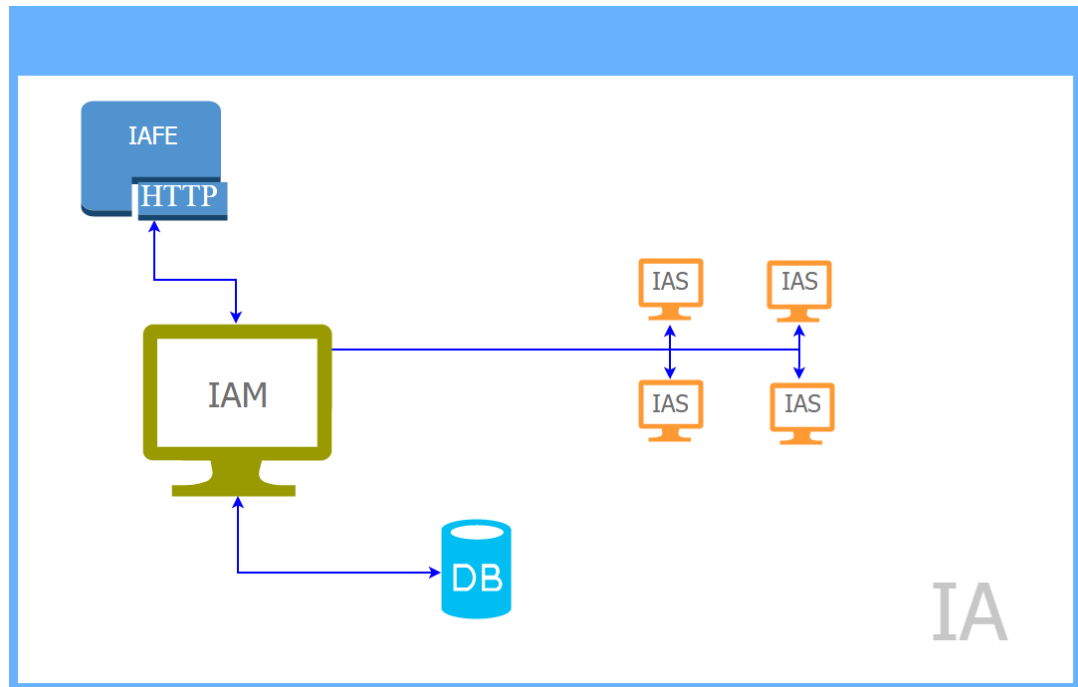


Figure 17: Components of IA

IA as console application: IA starts to run with parameters which included inside configuration file related to itself. This configuration file includes few parameters. IP and port parameters have been used for IA to listen specific port. Temporary file path has been used to store temporary files. DBConnectionParams has been used to store database connection parameters. Database tables include IDs of CUs and their files' keys, shared keys, V of files as blob, recent integrity and port parameters of IASs registered to IA.

As seen on Figure 17, IA includes IAM, IAFE, IASs and DB. IAM is main thread that communicated via socket communication. It manages all other components and the heart of IA. IAFE is used to offer front end via servlet for users to check integrity of their files over IA. IASs are other slave threads that simulates the server which sends IntegrityChallenges to cloud.

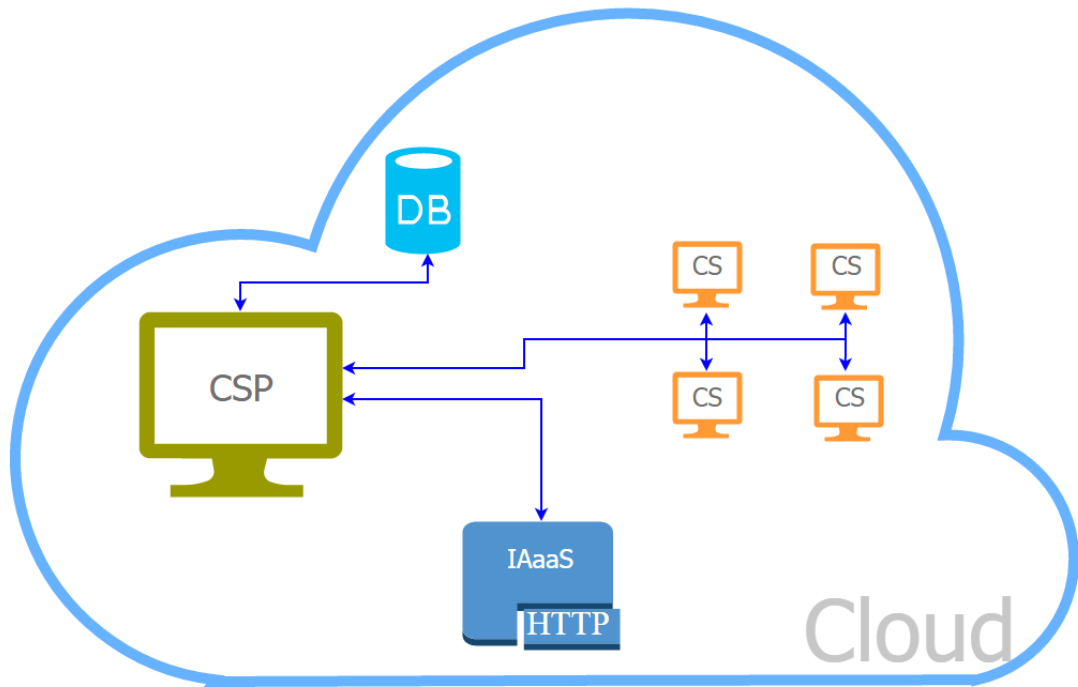


Figure 18: Components of Cloud

CSP as console application: CSP starts to run with parameters which included inside configuration file related to itself. This configuration file includes IP and port parameters that CSP starts to listen. Also, temporary folder path that is used for operation that needs temporary files in progress. To access database, connection parameters are defined inside XML file. CSP connects to database that has tables includes data (like CU IDs, CS parameters, etc.).

As seen on Figure 18, Cloud includes CSP, CSs, DB as main components. IAaaS servlet is available for IAaaS and CIAaaS models only. DB tables includes information about IDs of CUs, files of CUs, file indexes, registered active CSs, file block indexes of files over CSs.

IIA as an application includes Integrity Auditor Manager(IAM) as thread, Integrity Auditor Front End(IAFE) as a servlet, Integrity Auditor Threads (IATs) as threads and Integrity Auditor DB (IADB) as a PGSQL.

Simple Cloud Factory application communicates with cloud server application.

Simple cloud application includes CSP thread and CSP threads.

All components run on system as threads.

First application starts to run on test bed is CSP thread. By using reactor pattern, CSs is created. Communication between CSP and CS is implemented with interthread communication.

Second application starts to run on test bed is IAM thread. By using reactor pattern, IATs is created. Communication between IAM and IA components are implemented with interthread communication.

Third application starts to run on test bed is CU. CU is a simple application that reads commands and file paths through console.

Communication between CU and CSP is implemented using socket communication. CU communicates with IAFE over HTTP tunnel. IATs communicates with CSP over socket communication and IAaaS over HTTP tunnel.

Bash scripts are used to initialize applications.

To run multiple CU, executing the script, which starts CU application is enough. Cloud Factory can be implemented to run multiple clouds. Actually, CSP initializes a cloud system by running CSs and writes process id (and if necessary, servlet port) on a permanent path. To create clouds on CIAaaS test bed, Cloud Factory can run scripts to initialize CSPs. This scripts are called with parameters (like servlet port). Cloud Factory can read a path at regular intervals, which CSPs write process ids and servlet ports. When new file of CSP is added to that path, Cloud Factory can determine process id of a CSP and can register that CSP to itself.

4.4 Experimental Results

To test IA, several conditions has been determined. To provide this bed, various operations have different implementations. To achieve this; few implementations have been coded using factory patterns.

To determine effect of encryption over system, PrepareFile operation's related interfaces of related steps have two different implementations. One of this implementation offers encryption with AES and second implementation returns input as output which does not offer encryption. Different implementations of same interface are chosen by reading related values in XML files.

To determine effect of MHT and skip lists; interface of classes, which used to produce V, has two different implementation classes. These classes are inside Utility JAR. To switch between MHT and skip list, related XML values can be changed.

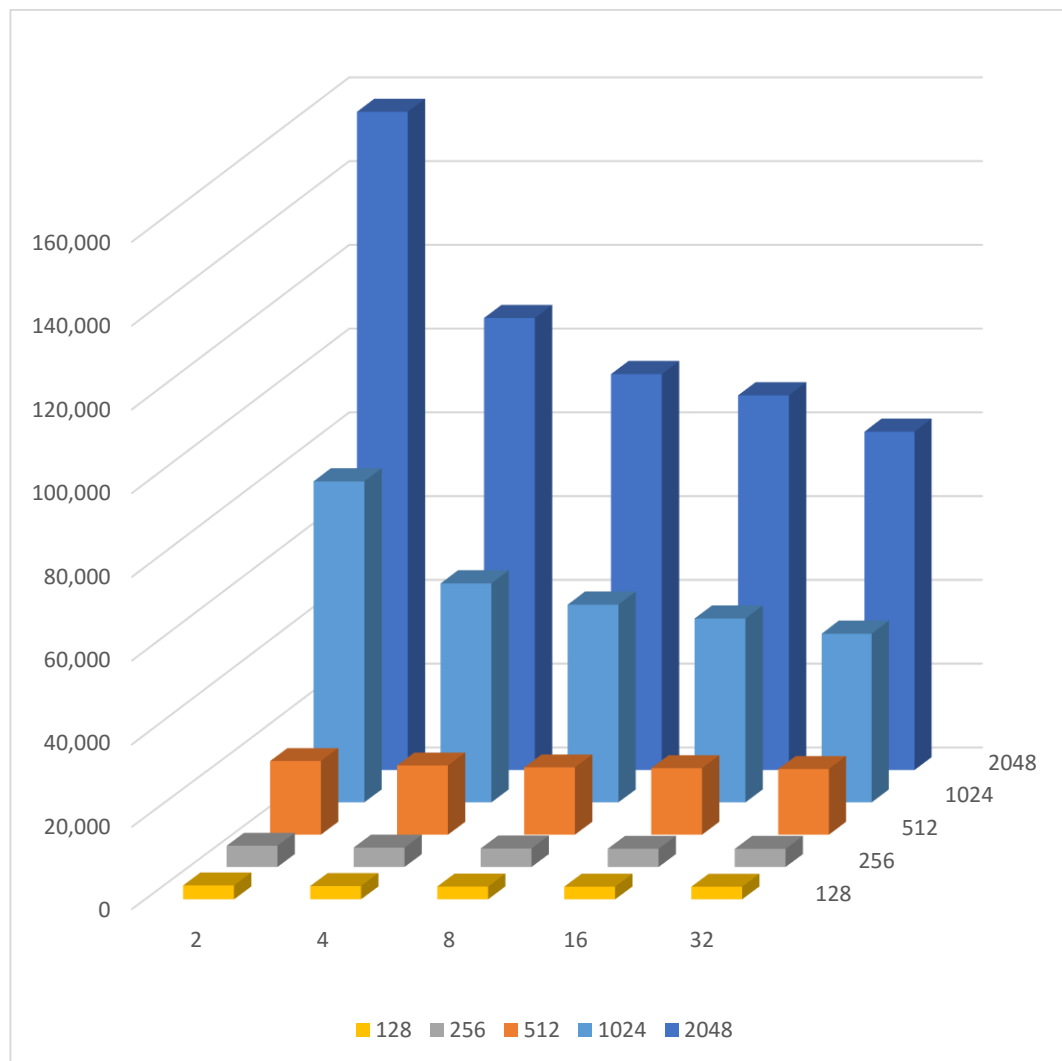


Figure 19: PrepareFile Step: Encryption Enabled, with IA, SL

To determine effect of block size over system, block size of system offered as configurable on common configuration XML file. By changing values at common configuration file for fixed file length, effect of block size over system is observed. Values of block sizes includes 2mb, 4mb, 8mb, 16mb, 32mb.

To determine effect of file size, five different files have been prepared by appending various big text files over internet. These files include 128mb, 256mb,

512mb, 1024mb and 2048mb length of files as seen on graphs created to show test results. Results of tests are time intervals which has unit of milliseconds.

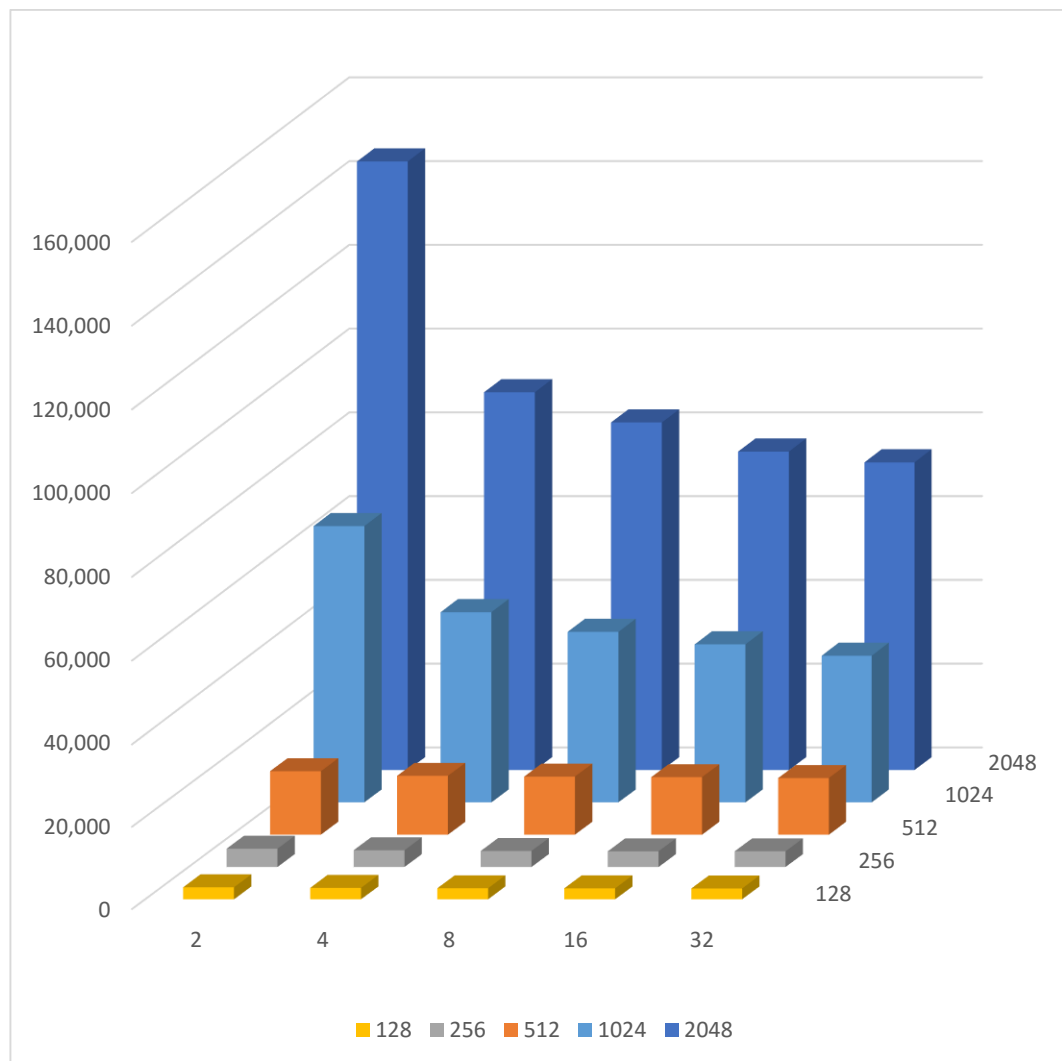


Figure 20: PrepareFile Step: Encryption Enabled, with IA, MHT

Changing block size does not affect the time needed to prepare file much except using smaller block size than 2Mb. Even, little changes can be observable on results; because of hard IO operations over disk, the main percentage of PrepareFile step is IO operations.

On Figure 19 SL used as structure to observe effects of file size and block size over PrepareFile step. On the other hand, this graph also shows the effect of number

of blocks. Figure 20 includes similar results and this results obtained by using MHT. In both cases, encryption is enabled over system and IA is activated.

On Figure 21 and Figure 22, effect of file size and block size can be seen. Different from result sets of Figure 19 and Figure 20, to obtain these results sets, encryption is disabled over system.

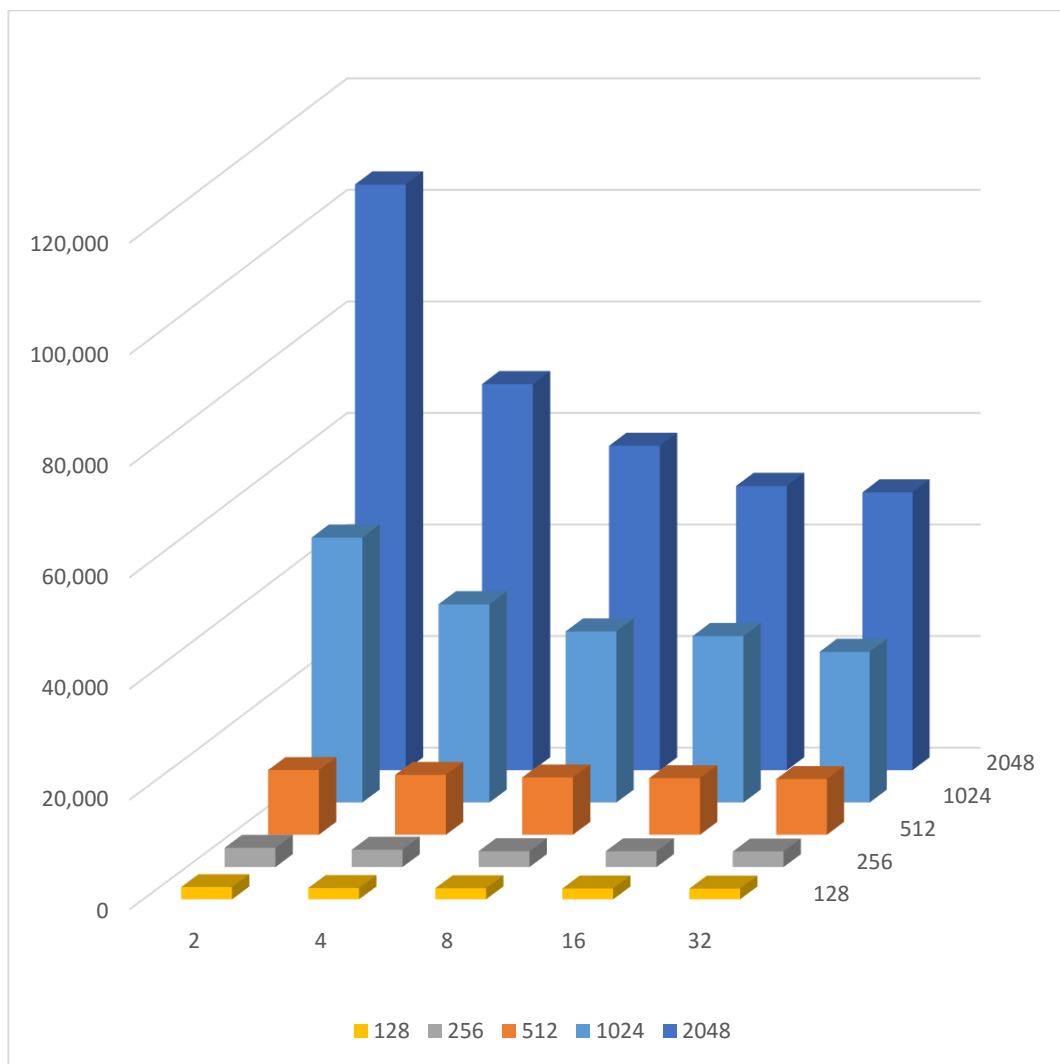


Figure 21: PrepareFile Step: Encryption Disabled, with IA, SL

Block size does not affect IntegrityChallenge and IntegrityStatus steps directly. Since number of nodes in MHT/SL affects time required to complete these steps, Block Size affects these steps indirectly.

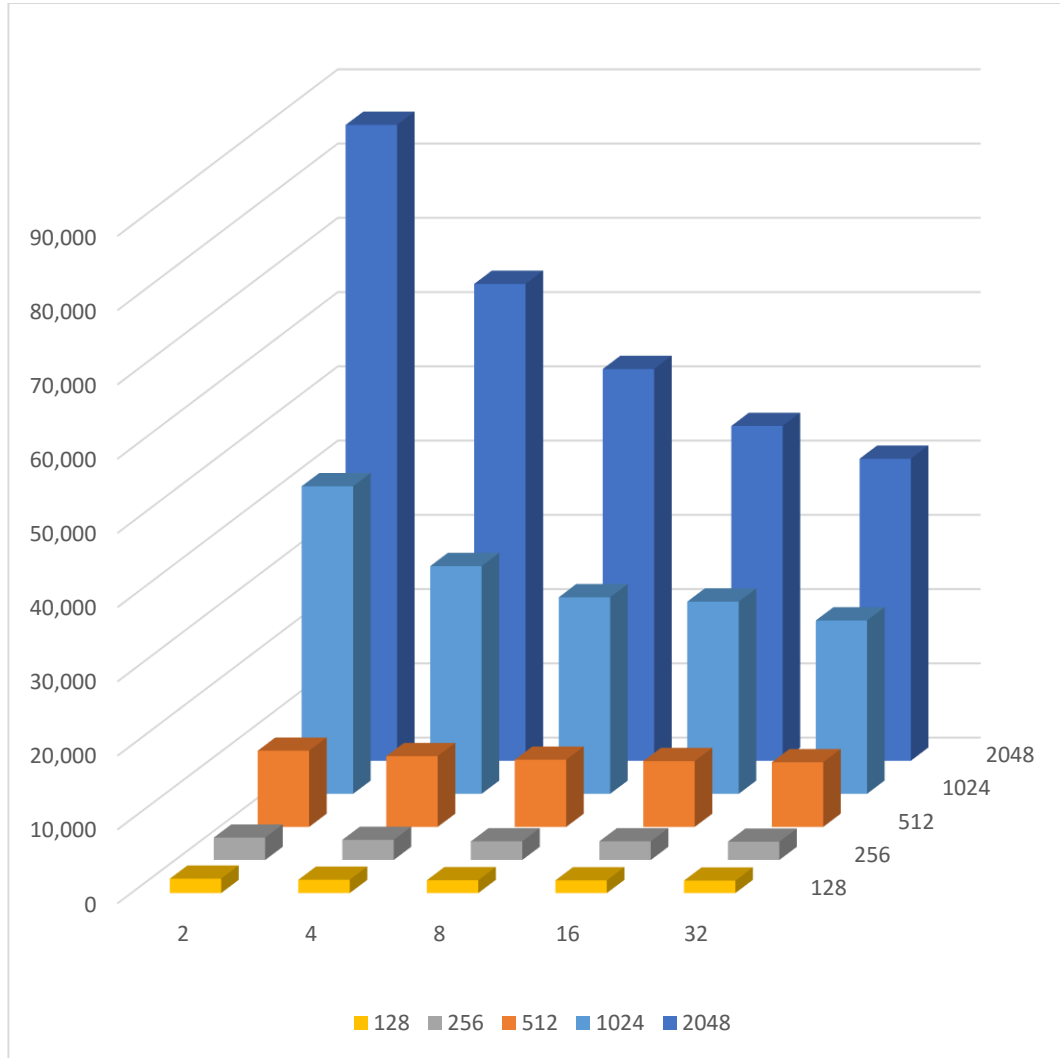


Figure 22: PrepareFile Step: Encryption Disabled, with IA, MHT

Making IA available on system does not change time needed for PrepareFile and IntegrityChallenge steps. Thus, graphs that includes result sets that IA disabled over system, is not put on to avoid dense of graphs. Even IntegrityStatus step is not affected by having active IA on system, a powerful server can speed up this step. Since, comparison operation of two validation data does not take long time, preferring powerful server does not change results a lot. On the other hand, if IA deals out with massive number of users, a powerful server is better choice. Preferring encryption at PrepareFile affects performance of that step around %30~%40. It is the tradeoff

between security and performance. If CU prefers to encrypt whole packet, even performance is getting lower, increased security is a benefit.

Active encryption on messages that transmitted between parties slightly affect the system since CBC mode is not used with AES. For increased security, it is usable for very sensitive data.

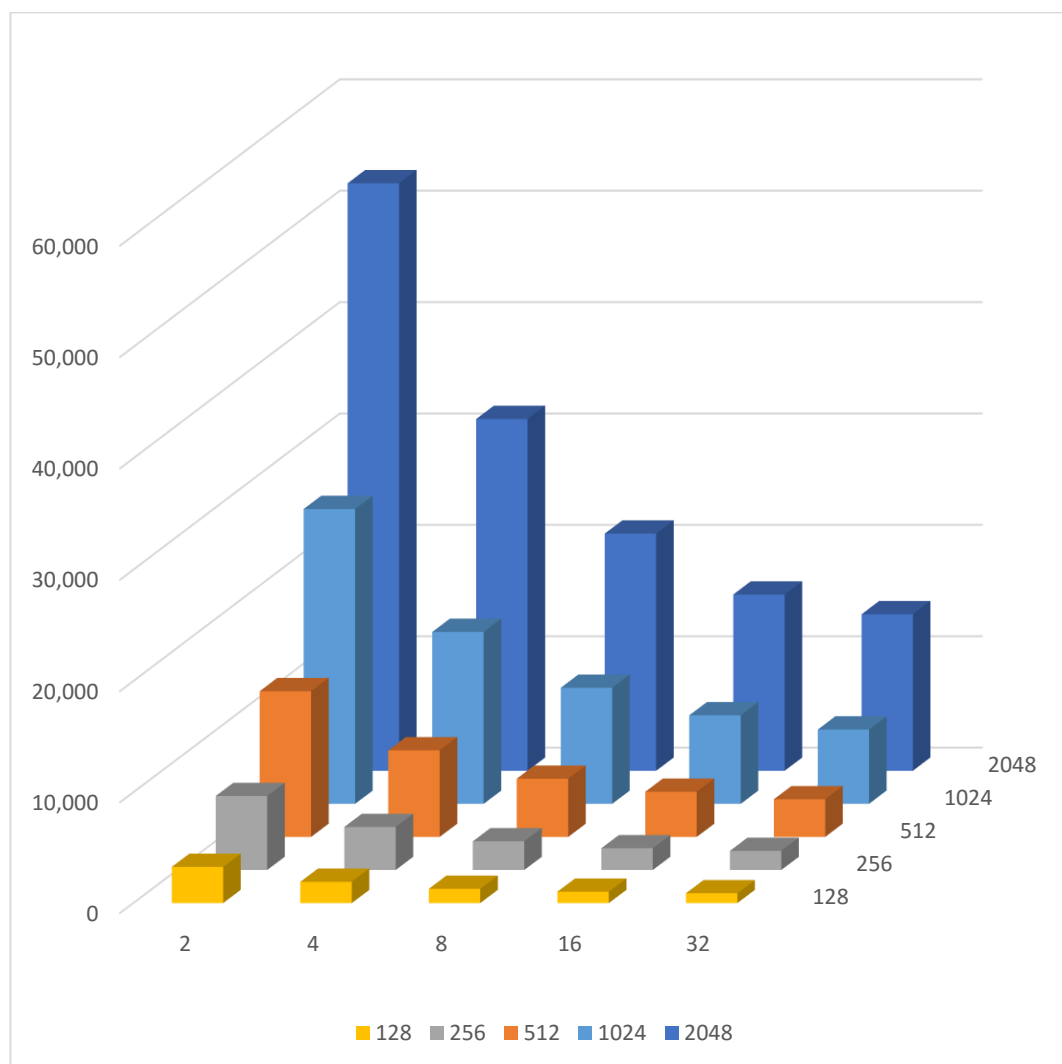


Figure 23: Integrity Challenge Response Step, MHT

On Figure 23 and Figure 24, time spent on IntegrityChallengeResponse step relevant to data structure is illustrated. Effect of file size and block size over producing

response (milliseconds as time) for Integrity Challenge step using these two data structures can be seen on these graphs.

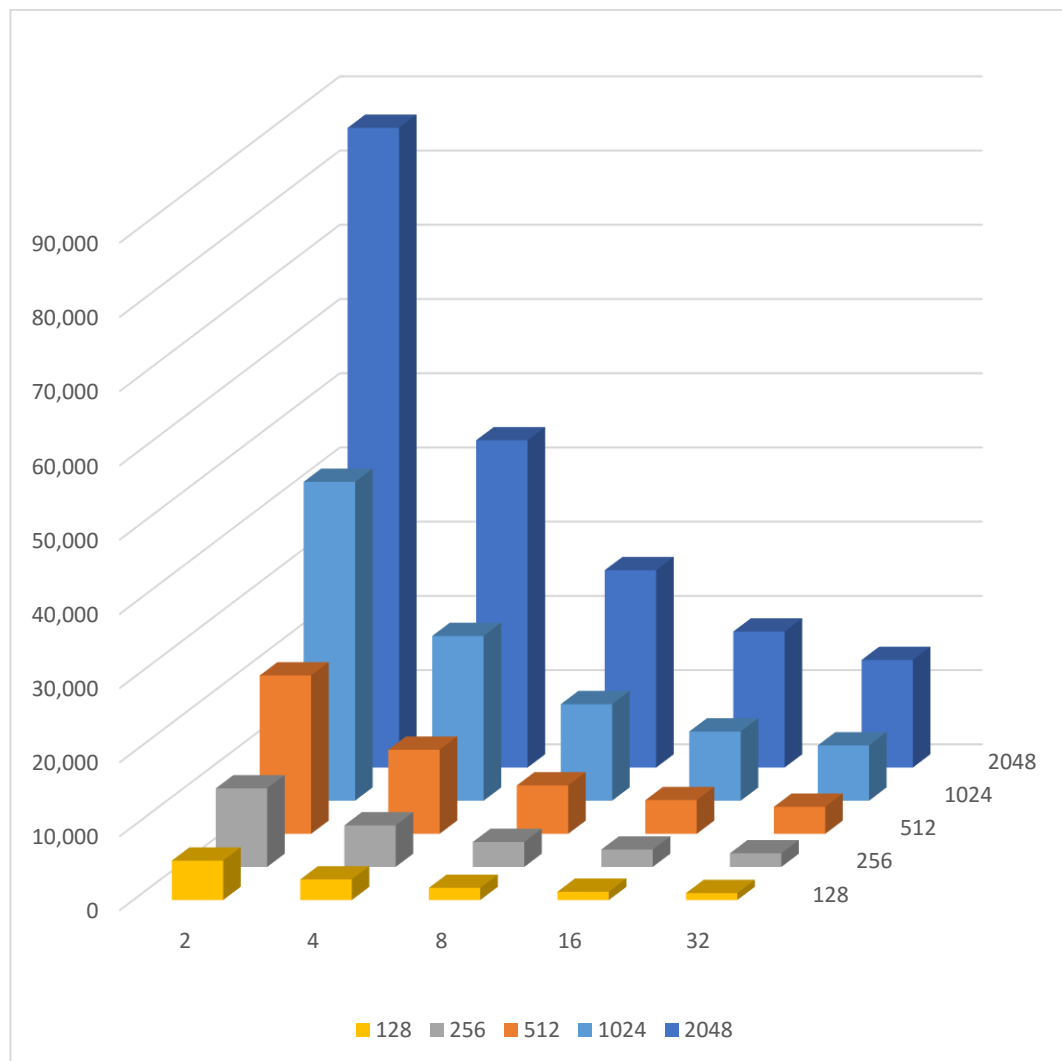


Figure 24: Integrity Challenge Response Step, SL

On Figure 25 and Figure 26, time spent on CheckIntegrity step relevant to data structure is illustrated. Effect of file size and block size over checking integrity of two data structures can be seen on these graphs.

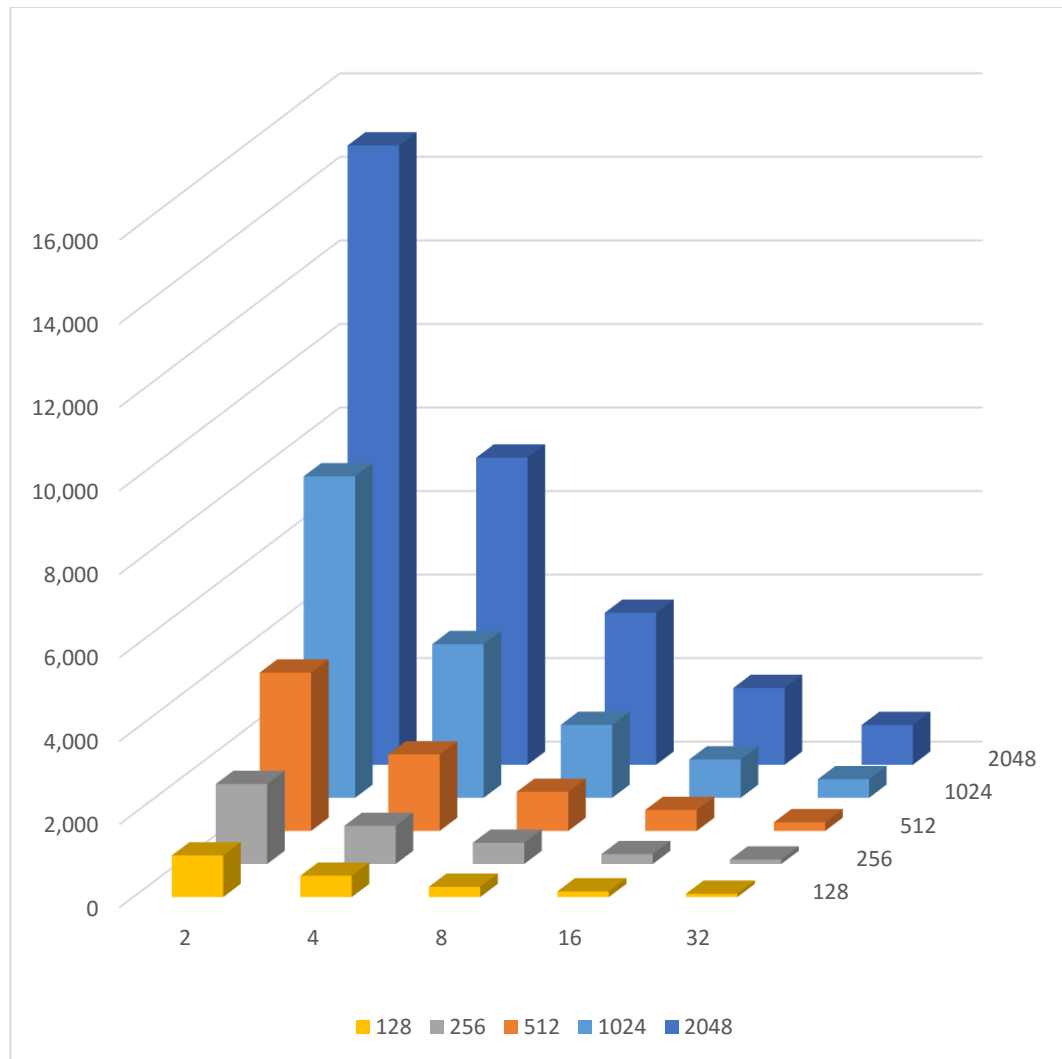


Figure 25: CheckIntegrity Step, MHT

If MHT is compared with SL, it's faster. It's because of nature of trees. On the other hand, as mentioned earlier; MHTs can be totally re-constructed for update and delete operations. This decision negatively affects system performance; so, only creation of V in PrepareFile step is affected by MHT/SL preference. And it is slightly affecting general performance of the system.

To bring out effect of data structure used to check integrity and encryption over PrepareFile step, another graph is created. As can be seen on Figure 27, encryption has solid effect over system. As mentioned above, this effect is around %30~%40.

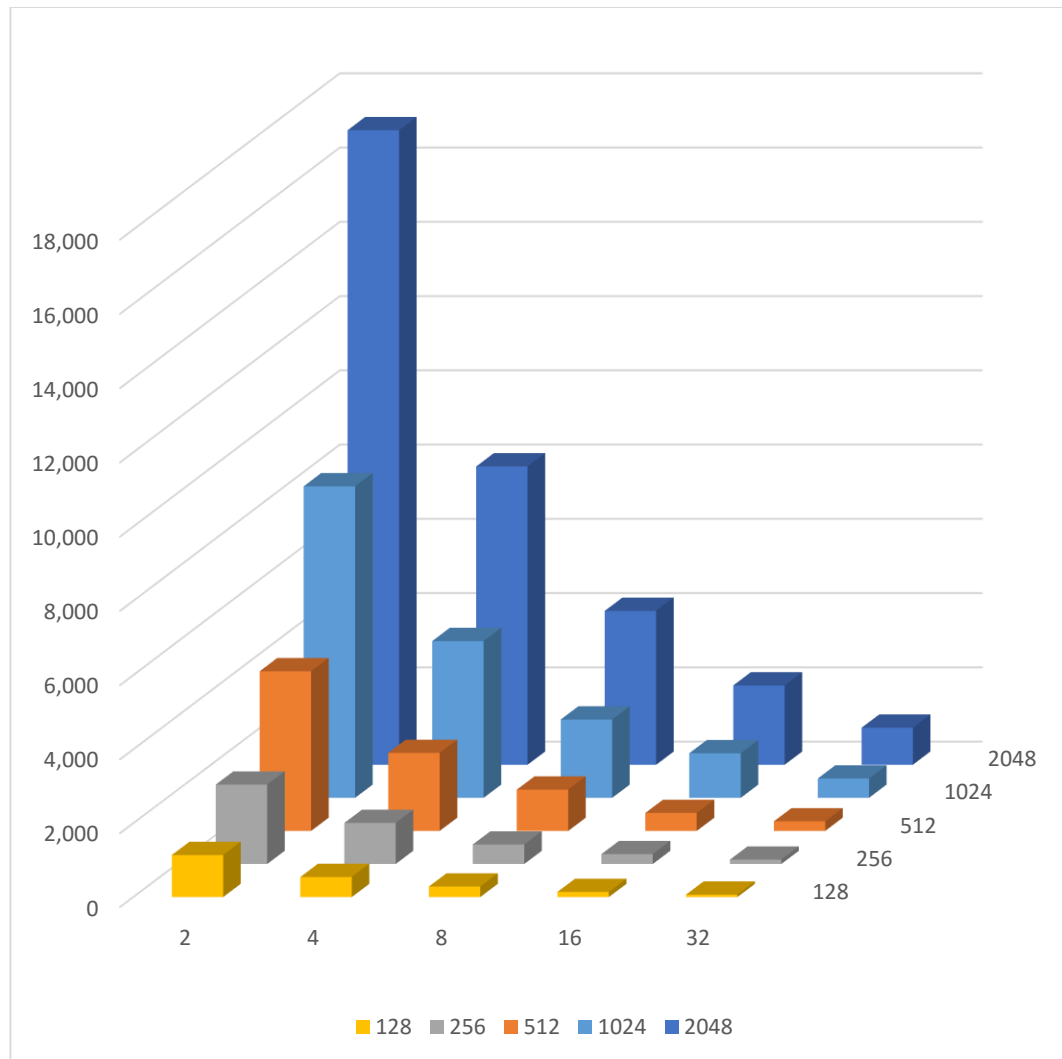


Figure 26: CheckIntegrity Step, SL

While block size is under two megabytes, it causes reduced performance. Because lower block size causes to have more blocks. Having much blocks on system increases number of transactions needed for any step. Higher block size provides small amount of blocks and it provides to have lesser transactions. As a result, if block size is more than two megabytes, it has obvious effect on system performance.

File size has most impact on system performance. For fixed block size, bigger files lead to have more blocks. Number of blocks directly effects number of transactions needed on system.

Flexible block size for different files (dividing files which have different sizes by fixed block size) is not a good idea. While operating system creates a file on disk,

it initially creates an entry on MFT (Master File Table which holds entries for all of disk files) and binds this MFT entry to physical address on disk. For read/write operations, a handle must be created for reading or writing into that file. However, this handle must be disposed when operation is completed. All these operations have high costs. To read, write or copy a file as one part is faster than doing same processes many times. At this point, minimizing the number of blocks as much as possible is a good idea. For example, the time required for processing one file that has million megabytes and million files that has one-megabyte size are not equals.

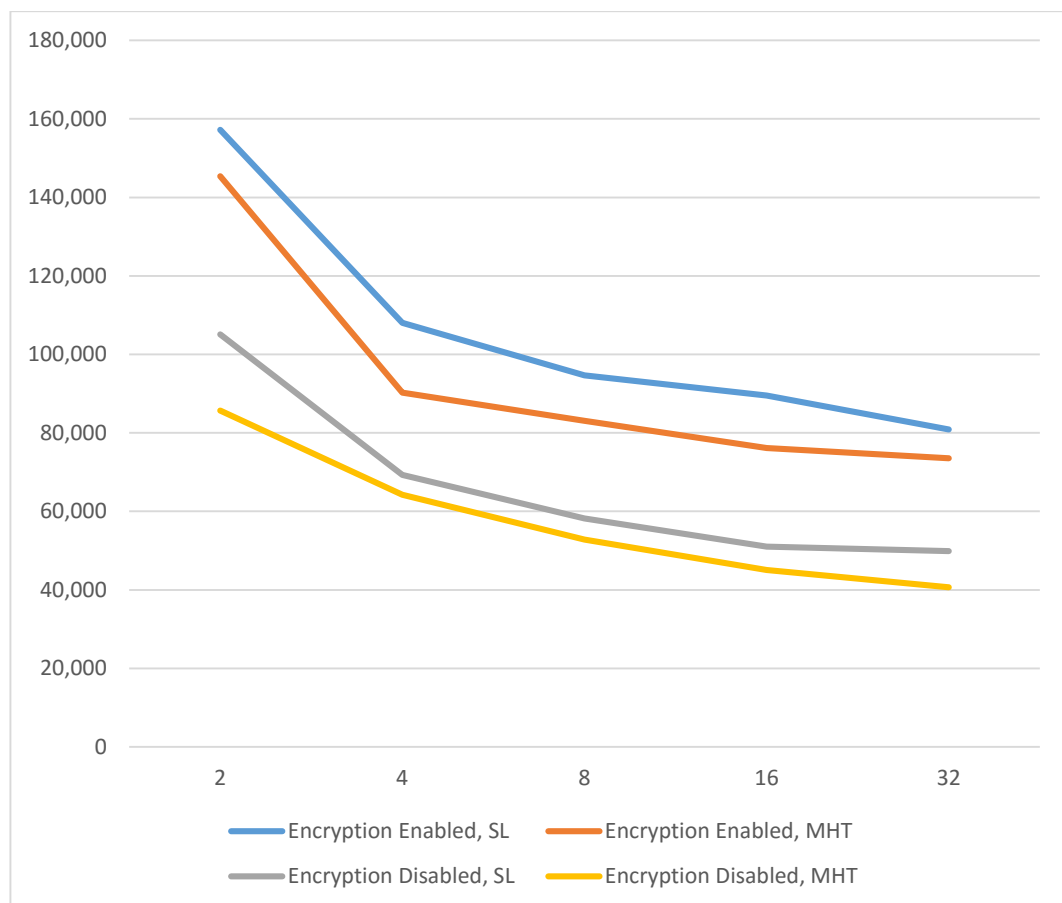


Figure 27: PrepareFile Step (2048Mb File on IA Enabled System)

*Block Size(Mb) x Results: Intervals (ms)

To obtain results that are more close to real life scenario, different test bed can be created.

First of all, rather than Windows 10, a Linux distribution which include only core packages of Linux and Java leads better results. Besides, rather than graphical user interface based operating system, command line interface oriented operating system minimize resource usage of operating system. This provides more resource pool for IA threads.

Rather than using all threads on one machine, each thread can run on different machines. This changes this results to more realistic results. At this point, network structure and connections must be chosen carefully to minimize effect of network and bandwidth.

Rather than using a computer, which caters daily usage, powerful servers must be used for test bed.

These implementation and tests achieved by writing temporary files to disk. This cause worse performance than in memory operations. At this point, by checking memory status and number of files needed to process, CSP can use memory as much as possible.

CHAPTER 5. Conclusion and Future Work

In the cloud environment, data owners outsource their data by storing remotely. However, cloud server may not be fully trusted when a sensitive data is considered. Therefore, data privacy and integrity are equally important. In this study, a scheme has been presented for verifying the data integrity and privacy by integrating integrity auditor into data possession scheme.

In [29], Dynamic provable data possession (DPDP) considers dynamic data operations, however it was lacking of cryptographic aspect. By fulfilling cryptographic aspect of that working, it is rendered as more secure.

In [12], Provable Data Possession (PDP) considers performance and security. But it was missing dynamic operations (adding new block(s) into, deleting block(s), updating block(s)) over files which uploaded to the cloud.

This DPDP focuses fulfilling missing sides of [12] and [29]; also compounds powerful sides. Besides, by integrating integrity auditor, this model becomes more relevant to real life. Integrity auditor can sustain prejudgments about cloud storage security, which are still disputed by users and companies. As discussed in 2.1, any sanctions can be administered about cloud providers, which provisioned by integrity auditor reports.

Several experiments have been conducted by integrating IA; also Skip List (SL) and Merkle Hash Tree (MHT) has been evaluated in the proposed system.

MHTs provides better performance over SLs. However, even MHTs can prove the integrity of a whole file, it does not support CRUD operations over files that uploaded to cloud. At this point, various implementations of skip lists are stepping into the breach by supporting CRUD operations. Besides, skip lists have decent performance for CRUD operations.

Bigger blocks produce lesser number of blocks. However, general system performance is inversely proportional to number of blocks. Inherently, time needed to prepare packet and send packet over a network is directly proportional to the size of the file.

Activating IA on the system has not negative effect over system since it only changes start and finish points of the few flows. However, IA provides better control over checking integrity of the files. IA takes the burden of checking integrity of files from users' shoulder and automatizes integrity check process. Activating encryption of packet causes more time to be spent on PrepareFile and StoreFile steps. It is a tradeoff between performance and privacy; but it has slight effect on time needed for steps. Even it reduces performance a bit, if data is very sensitive, it must be activated.

Some suggestions can be offered to increase the performance of the system. These suggestions are mentioned below.

Encryption tests can be implemented using CBC mode. If it does not reduce performance much, it can be applicable for security critical systems.

Another idea to get better performance is parallelizing steps when possible. Block related operations, which is independent of other blocks' status can be parallelized. This must be achieved by refactoring code for parallel computing. While preparing a file; creating packets of blocks are block independent operation. At CSP side, after whole packet become decrypted, rest of operation can be parallelized too. Even two different resources can do those steps independent from other, nice contribution to performance can be achieved.

REFERENCES

- [1] K. Jamsa, Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Jones & Bartlett Publishers, 2011.
- [2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Recommendations of the National Institute, 2009.
- [3] "PixGood," [Online]. Available: <http://pixgood.com/cloud-computing-architecture.html>. [Accessed 01 01 2015].
- [4] R. Buyya, J. Broberg and A. M. Goscinski, Cloud Computing Principles and Paradigms, 2010.
- [5] K. Remde, "SaaS, PaaS, and IaaS.. Oh my! ("Cloudy April" – Part 3)," Microsoft TechNet Blogs, 03 04 2011. [Online]. Available: <https://blogs.technet.microsoft.com/kevinremde/2011/04/03/saas-paas-and-iaas-oh-my-cloudy-april-part-3/>. [Accessed 01 01 2015].
- [6] T. Dillon, C. Wu and E. Chang, "Cloud computing: issues and challenges," in *24th IEEE International Conference*, 2010.
- [7] "big data guide: Overview of big data technology, use cases, tutorials, markets," 01 2014. [Online]. Available: <http://bigdata-guide.blogspot.com.tr/2014/01/types-of-cloud-computing-public-private.html>. [Accessed 01 01 2015].
- [8] Y. Cherdantseva and J. Hilton, "A Reference Model of Information Assurance & Security," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference*, 2–6 September 2013.
- [9] A. A. Muthitacharoen, R. Morris., T. M. Gil and B. Chen., "Ivy: A read/write peer-to-peer file system," *5th Symposium on Operating Systems Design and Implementation*, 2002.

- [10] J. Li, M. Krohn, D. Mazières and D. Shasha, "Secure untrusted data repository (SUNDR)," *Symposium on Operating Systems Design and Implementation*, 2004.
- [11] U. Maheshwari, R. Vingralek and W. Shapiro, "How to build a trusted database system on untrusted storage," *OSDI*, 2000.
- [12] G. Ateniese, R. D. Pietro, L. V. Mancini and G. Tsudik, "Scalable and efficient provable data possession," *Proceedings of the 4th international conference on Security and privacy in communication networks*, 2008.
- [13] I. Clarke and e. al., "Freenet: A distributed anonymous information storage and retrieval system.," 2001.
- [14] "Distributed File System For Cloud," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Distributed_file_system_for_cloud. [Accessed 03 01 2016].
- [15] Q. Zhang, L. Cheng and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, pp. 7-18, 2010.
- [16] CSA, "Top Threats to Cloud Computing v1.0," CSA, 2010.
- [17] CSA, "The Notorious Nine Cloud Computing Top Threats in 2013," CSA, 2013.
- [18] G. Ateniese, R. Burns, R. Curtmola, J. Herring and L. Kissner, "Provable Data Possession at Untrusted Stores," *ACM Conference on Computer and Communications Security-CCS 2007*, 2007.
- [19] Y. Deswarte, J.-J. Quisquater and A. Saïdane, "Remote integrity checking: How to trust files stored on untrusted servers," *Integrity and Internal Control in Information Systems VI*, 2004.
- [20] P. Golle, S. Jarecki and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity.," in *Financial Cryptography*, 2002.

- [21] R. Johnson, D. Molnar, D. Song and D. Wagner, "Homomorphic signature schemes.," *Topics in Cryptology—CT-RSA*, pp. 244-262, 2002.
- [22] M. Krohn, M. Freedman and D. Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution.," *IEEE Symposium on Security and Privacy*, pp. 226-240, 2004.
- [23] G. Filho, D. L. Barreto and P. S. L. Messeder, "Demonstrating data possession and uncheatable data transfer.," *IACR Cryptology ePrint Archive*, 2006.
- [24] T. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," *26th IEEE International Conference*, 2006.
- [25] F. Sebe, A. Martinez-Balleste, Y. Deswarte, J. Domingo-Ferrer and J.-J. Quisquater, "Time-bounded remote file integrity checking," 2004.
- [26] G. Yamamoto, S. Oda and K. Aoki, "Fast integrity for large data," *SPEED*, pp. 21-32, 2007.
- [27] M. N. Krohn, M. J. Freedman and D. Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution," *IEEE Symposium on Security and Privacy*, 2004.
- [28] A. Oprea, M. Reiter and K. Yang, "Space-Efficient Block Storage Integrity," *NDSS '05*, 2005.
- [29] C. C. Erway, A. Küpçü, C. Papamanthou and R. Tamassia, "Dynamic Provable Data Possession Categories and Subject Descriptors," *ACM Conference on Computer and Communications Security*, pp. 213-222, 2009.
- [30] B. Chen and R. Curtmola, "Robust Dynamic Provable Data Possession," *In Proceedings of ICDCS Workshops*, pp. 515-525, 2012.
- [31] Y. Zhu, H. Wang, Z. Hu, G. Ahn, H. Hu and S. S. Yau, "Efficient provable data possession for hybrid clouds," *In Proceedings of ACM*

Conference on Computer and Communications Security, pp. 756-758, 2010.

- [32] Y. Zhu, H. Hu, G. Ahn and M. Yu, "Cooperative Provable Data Possession for Integrity Verification in Multicloud Storage," *In Proceedings of IEEE Trans. Parallel Distrib. Syst.*, pp. 2231-2244, 2012.
- [33] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Information Sciences*, vol. 258, pp. 371-386, 2014.
- [34] M. S. Niaz and G. Saake, "Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data," *GI-Workshop on Foundations of Databases*, 2015.
- [35] W. Pugh, "Skip lists: A probabilistic alternative to balanced trees," *Commun. ACM*, 1990.
- [36] G. P. Biswas, "Diffie-Hellman technique: extended to multiple two-party keys and one multi-party key," *Information Security*, pp. 12-18, 2008.
- [37] M. Rouse, "TechTarget," 12 October 2014. [Online]. Available: <http://searchcloudcomputing.techtarget.com/definition/public-cloud>.

APPENDIX A: COMPARISON OF PREVIOUS SYSTEMS

Scheme	Type	CSP Complexity	CU Complexity	Communication Complexity	Privacy	Multi Cloud	Block Operations				Op	
							$\rightarrow$	+	*	-	S	C
PDP	HomT	$O(t)$	$O(t)$	$O(1)$	$\checkmark$	$\diamond$	$\checkmark$				$\checkmark$	
SPDP	MHT	$O(t)$	$O(t)$	$O(t)$	$\checkmark$		$\checkmark^*$	$\checkmark^*$		$\checkmark^*$	$\checkmark$	
DPDP-I	SL	$O(t \log n)$	$O(t \log n)$	$O(t \log n)$	$\checkmark$		$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	
DPDP-II	SL	$O(t \log n)$	$O(t \log n)$	$O(t \log n)$			$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	
SecCloud	MHT	$O(t)$	$O(t)$	$O(t)$	$\checkmark$						$\checkmark$	$\checkmark$
IA	MHT	$O(t \log n)$	$O(t \log n)$	$O(t)$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	
IA	SL	$O(t \log n)$	$O(t \log n)$	$O(t)$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	

Table 2: Comparison of Previous Schemes

APPENDIX B: DETAILED FLOW OF SYSTEM

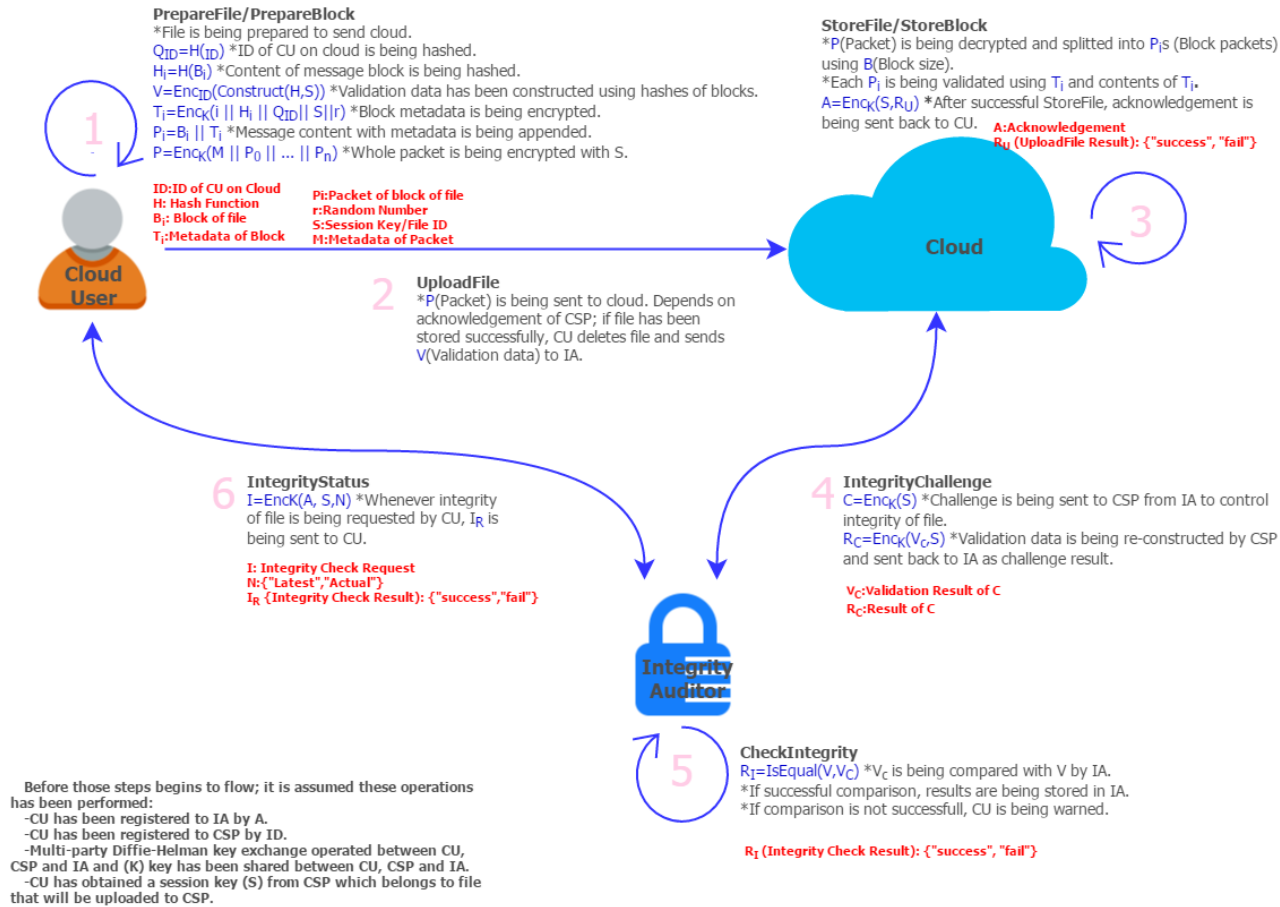


Figure 28: Detailed Flow of System

APPENDIX C: DETAILED TEST RESULTS

		with IA						without IA				
			2	4	8	16	32	2	4	8	16	32
With Encryption	SL	128	3,389	3,231	3,110	3,077	3,062	3,372	3,227	3,125	3,049	3,028
		256	5,167	4,714	4,503	4,430	4,419	5,182	4,728	4,539	4,461	4,401
		512	17,889	16,810	16,341	16,158	15,954	18,050	16,625	16,177	15,883	15,650
		1024	76,870	52,505	47,429	44,124	40,485	76,716	52,242	47,191	44,432	40,201
		2048	157,255	108,011	94,629	89,556	80,878	147,696	107,794	94,912	87,758	84,933
	MHT	128	2,931	2,788	2,680	2,673	2,636	2,906	2,801	2,691	2,631	2,616
		256	4,448	4,068	3,899	3,814	3,818	4,503	4,070	3,921	3,840	3,815
		512	15,420	14,305	14,118	13,976	13,752	15,541	14,430	13,896	13,675	13,569
		1024	66,185	45,626	40,931	37,946	35,221	65,055	45,346	40,632	38,212	34,693
		2048	145,406	90,293	83,084	76,132	73,542	127,166	93,673	82,574	75,648	73,127
No Encryption	SL	128	2,236	2,067	2,021	1,978	1,956	2,181	2,036	1,972	1,960	1,916
		256	3,513	3,163	2,904	2,883	2,841	3,410	3,063	3,032	2,979	2,918
		512	11,806	10,926	10,458	10,324	10,162	11,425	10,623	10,337	10,212	10,282
		1024	47,659	35,703	30,828	30,004	27,165	51,476	38,659	29,683	27,503	26,050
		2048	105,078	69,287	58,224	51,030	49,876	100,433	71,478	61,598	55,902	53,508
	MHT	128	1,948	1,803	1,738	1,721	1,684	1,900	1,759	1,694	1,688	1,654
		256	3,025	2,720	2,523	2,529	2,472	2,939	2,628	2,616	2,589	2,532
		512	10,319	9,582	9,098	8,910	8,760	9,928	9,231	8,890	8,732	8,781
		1024	41,463	30,704	26,512	25,923	23,389	44,836	33,363	25,795	23,873	22,664
		2048	85,647	64,233	52,775	45,102	40,687	86,473	61,400	53,652	47,461	45,588

Table 3: Test Results for PrepareFile Phase

*Orange to Dark Columns: Block Size(Mb), Light to Dark Blue Rows: File Size(Mb), Results: Intervals(ms)

		2	4	8	16	32
SL	128	5,328	2,787	1,645	1,123	941
	256	10,627	5,617	3,375	2,351	1,840
	512	21,373	11,362	6,546	4,553	3,653
	1024	42,982	22,218	13,025	9,344	7,473
	2048	86,200	44,098	26,593	18,332	14,504
MHT	128	3,267	1,921	1,288	1,026	885
	256	6,682	3,886	2,594	1,964	1,730
	512	13,130	7,814	5,257	4,077	3,404
	1024	26,496	15,459	10,447	7,975	6,698
	2048	52,756	31,595	21,301	15,832	14,061

Table 4: Test Results for Integrity Challenge Response

*Orange to Dark Columns: Block Size(Mb), Light to Dark Blue Rows: File Size(Mb), Results: Intervals(ms)

		2	4	8	16	32
SL	128	1,139	544	286	139	66
	256	2,151	1,112	528	268	117
	512	4,314	2,113	1,119	487	259
	1024	8,399	4,231	2,117	1,208	525
	2048	17,094	8,038	4,148	2,140	1,001
MHT	128	1,002	513	247	137	79
	256	1,927	918	511	238	107
	512	3,802	1,839	941	506	209
	1024	7,704	3,687	1,754	924	449
	2048	14,829	7,356	3,641	1,841	953

Table 5: Test Results for CheckIntegrity Step

*Orange to Dark Columns: Block Size(Mb), Light to Dark Blue Rows: File Size(Mb), Results: Intervals(ms)

APPENDIX D: PSEUDOCODES

```
Function CreateValidationData(String filePath, Int ValidationStructureType)
    ValidationContent valContent
    List hashFileList = filePath.findFiles("*hash")
    HashfileList.orderAlphabetical()
    Switch(ValidationStructureType)
        case "MHT":
            valContent=new MHT()
        case "SL":
            valContent=new SL()
    For(String hashFilePath:hashFileList)
        String hashValue = CreateSHA2OfFile(hashFilePath.read())
        valContent.push()
    return valContent
End Function
```

Algorithm 1: Pseudocode of Creation of V

```
Function PrepareFile(String filePath)
    List<String> blockFileList
    List<String> packetFileList
    blockFileList = SplitFile(filePath)
    For(String blockPath: blockFileList)
        packetFileList.add(CreatePacketofBlock(OP_NEW, blockPath, ID, S,
K))
    plainPackPath = AppendBlockFiles(filePath, packetFileList)
    encryptedFilePath = EncryptFileWithAES(plainPackPath, K)
    return encrpytedFilePath
End Function
```

```
Function PrepareBlock(String blockPath, int operationType)
    String packPath
    List<String> packetFileList
    Switch(operationType)
        case OP_INS:
            packPath= CreatePacketofBlock(OP_INS, blockPath, ID, S, K)
        case OP_UP:
            packPath= CreatePacketofBlock(OP_UP, blockPath, ID, S, K)
        case OP_DEL:
            packPath= CreatePacketofBlock(OP_DEL, blockPath, ID, S, K)
    packetFileList.add(packPath)
```

```

        plainPackPath = AppendBlockFiles(filePath, packetFileList)
        encryptedFilePath = EncryptFileWithAES(plainPackPath, K)
        return encrpytedFilePath
End Function

Function SplitFile(String filePath)
    File file = new File(filePath)
    List blockFileList
    Int i = 1
    While (byte[] blockContent = read(file, blockSize)
        blockFileName = filePath + "." + i
        WriteToFile(blockFileName, blockContent)
        blockFileList.add(blockFileName)
    return blockFileList
End Function

Function CreatePacketOfBlock(int opType, String blockPath, String ID,
String S, String K)
    Int index = blockPath.getExtension()
    String hashFilePath = CreateSHA2OfFile(blockPath)
    String QID = CreateSHA2OfString(ID)
    String r = ToString(GetRandomNumber(16))
    String metaData = i + hashFilePath + QID + S + R
    String encMetaData = EncryptStringWithAES(metaData, K)
    String packetPath = WritePacketOfBlock (blockPath + ".bpack",
encMetaData, blockPath)
    Return packetPath
End Function

Function WritePacketOfBlock(String blockPath, String encMetaData, String
blockPath)
    File packetBlock = new File(blockPath)
    File blockFile = new File(blockPath)
    String content = blockFile.read()
    blockFile.write(content)
    blockFile.write(encMetaData)
    return blockPath
End Function

Function AppendBlockFiles(String filePath, List fileBlockPaths)

```

```

File plainFile = new File(filePath + "fpack")
For(String blockPath: fileBlockPaths)
    String content = ReadFile(blockPath)
    plainFile.write(content)
return plainFile.getPath()
End Function

```

Algorithm 2: Pseudocode of PrepareFile and PrepareBlock

```

Function CheckIntegrity(Object vBase, object v)
    If(v.type == MHT){
        baseData = ((MHT)vBase).getRoot()
        valData = ((MHT)v).getRoot()
        return CheckIntegrity(baseData, valData)
    Else
        baseData = ((SL)v).getRoot()
        validationData = ((SL)v).getRoot()
        return CheckIntegrity(baseData, valData)
    End Function

```

```

Function CheckIntegrity(MHTPart pBase, MHTPart p)
    If(pBase == null && p ==null)
        return true
    If(pBase.value == p.value)
        Return(CheckIntegrity(pBase.left,          p.left)          &&
CheckIntegrity(pBase.right, p.right)
    Else
        Return false
    End Function

```

```

Function CheckIntegrity(SLPart pBase, SLPart p)
    If(pBase == null && p ==null)
        return true
    If(pBase.value == p.value)
        Return(CheckIntegrity(pBase.bottom,        p.bottom)        &&
CheckIntegrity(pBase.right, p.right)
    Else
        Return false
    End Function

```

Algorithm 3: Pseudocode of CheckIntegrity