Secure Data Deduplication in Cloud Environments

by

Fatema Rashid

B.S.Comp Sc., National University of Computer and Emerging Sciences, Pakistan, 2004

M.S.Comp Sc., Ryerson University, Canada, 2009

A Dissertation

presented to the Ryerson University

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy in the

Program of Computer Science

Toronto, Ontario, Canada

AUTHORS DECLARATION FOR ELECTRONIC SUBMISSION OF A DISSERTATION

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

I authorize Ryerson University to lend this dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

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

Abstract

Secure Data Deduplication in Cloud Environments

© Fatema Rashid, 2015

Doctor of Philosophy

Computer Science

Ryerson University

With the tremendous growth of available digital data, the use of Cloud Service Providers (CSPs) are gaining more popularity, since these types of services promise to provide convenient and efficient storage services to end-users by taking advantage of a new set of benefits and savings offered by cloud technologies in terms of computational, storage, bandwidth, and transmission costs. In order to achieve savings in storage, CSPs often employ *data deduplication* techniques to eliminate duplicated data. However, benefits gained through these techniques have to balanced against users' privacy concerns, as these techniques typically require full access to data. In this thesis, we propose solutions for different data types (text, image and video) for secure data deduplication in cloud environments. Our schemes allow users to upload their data in a secure and efficient manner such that neither a semi-honest CSP nor a malicious user can access or compromise the security of the data. We use different image and video processing techniques, such as data compression, in order to further improve the efficiency of our proposed schemes. The security of the deduplication schemes is provided by applying suitable encryption schemes and error correcting codes. Moreover, we propose proof of storage protocols including *Proof of Retrievability (POR)* and *Proof of Ownership (POW)* so that users of cloud storage services are able to ensure that their data has been saved in the cloud without tampering or manipulation. Experimental results are provided to validate the effectiveness of the proposed schemes.

Acknowledgments

I would like to express my inmost gratitude and appreciation to my supervisors, Professor Isaac Woungang and Professor Ali Miri for their absolute support throughout this journey. They have been my guide and mentor not only in the technical aspects but also persuasively conveyed a spirit of research and motivation to me. I could not have asked for more dedicated, understanding and knowledgeable teachers for my PhD research.

I also wish to thank the members of my thesis committee, Dr. Alireza Sadeghian, Dr. Alex Ferworn and Dr. Cungang Yang for their time and contributions during the entire process of my PhD defense.

I would like to thank my parents Kahkashan Khalid and Syed Khalid Zaki for their prayers and all emotional and moral support.

I would specially like to mention my little angels Nabiha and Reja for being my source of motivation and keeping the life joyful and happier throughout this passage.

I dedicate this thesis to my husband Rashid who was always there for me with all his love and support, without which I would not have been able to accomplish a bit of this success.

Dedication

Dedicated to my husband

Rashid

I would not be here without his support and his strong belief in me.

Contents

List of Tables

List of Figures

List of Abbreviations

CSP	Cloud Service Provider
TTTD	Two Threshold Two Divisor
AES	Advance Encryption Standard
MWSR	Many Writer and Single Reader
SLA	Security Socket Layer
PDP	Proof of Data Possession
POR	Proof of Retrievability
POW	Proof of Ownership
MHT	Merkel Hash Tree
BER	Bit Error Rate
SISO	Soft-in-Soft-out
PRF	Pseudo Random Function
SPIHT	Set Partitioning In Embedded Hierarchial Trees
EZW	Embedded Zero Tree of Wavelet Transform
RS	Reed Solomon
HD	High Definition

List of Abbreviations

CAVLC	Context Adaptive Variable Length Coding
CABAC	Context Adaptive Binary Arithmetic Coding
SEI	Supplemental Enhancement Information
MV	Motion Vector
QCIF	Quarter Common Intermediate Format
PSNR	Peek Signal to Noise Ratio
ECC	Error Correction Code
GOP	Group of Pictures

Chapter 1

Introduction

In the recent years, data deduplication has been advocated as a promising and effective technique to save the digital space by removing the duplicated data from the data centers or clouds. Data deduplication is a process of identifying the redundancy in data and then removing it. The resulting unique single copy is stored and will then serve all the authorized users.

1.1 Context and Motivation

The cloud computing paradigm is the next generation architecture for the business of information technology, which presents to its users some huge benefits in terms of computational costs, storage costs, bandwidth and transmission costs [1]. Typically, the cloud technology transfers all the data, databases and softwares over the Internet for the purpose of achieving huge cost savings for the CSP. In cloud computing [1], services such as *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)*, *Software-as-a-Service (SaaS)*, *and Database-as-a-Service (DaaS)* are currently offered. This thesis is concerned with the DaS property of cloud computing, among which cloud data storages services are of main interest. Dropbox,

Mozy, and SpiderOak are examples of popular cloud storages [1].

With the advent of cloud computing and its digital storage services, the growth of digital content has become irrepressible at both the enterprise and individual levels. According to the EMC Digital Universe Study [2], the global data supply has reached 2.8 trillion Giga Byte (GB) in 2012, but just 0.5% of it was used for various kind of analysis purposes. The same study has also revealed that volumes of data are projected to reach about 5247 GB per person by 2020. Due to this explosive growth of digital data, there is a clear demand from CSPs for more cost effective use of their storage and network bandwidth for data transfer purpose.

Also the use of Internet and other digital services have given rise to a digital data explosion, including those in cloud storages. A survey [2] revealed that only 25% of the data in data warehouses are unique in data warehouses. In the presence of this huge problem of big data, it would be beneficial in terms of storage savings if the replicated data could be removed from the data storages. According to a survey [3], only 25 GB of the total data for each individual user are unique and the remaining ones are similar shared data among various users. On the enterprise level [4], it was reported that businesses hold an average of three to five copies of files, with 15% to 25% of these organizations having more than 10 copies of the files. Keeping these facts in view, CSPs have massively adopted data deduplication , a technique that allows the CSP to save some storage space by storing only a single copy of previously duplicated data.

A major issue hindering the acceptance of cloud storage services by users is the data privacy issue associated with the cloud paradigm. Indeed, although data is outsourced in its encrypted form, there is no guarantee of data privacy when an honest but curious CSP handles the management of confidential data while these data reside in the cloud. This problem is even more challenging when data deduplication is performed by the CSP as a

way to achieve cost savings. Data deduplication is being adopted by many popular cloud storage vendors like Dropbox, Spider Oak, Waula and Mozy. Recently, Dropbox [5] was reported to have stolen millions of passwords of some users, but they shifted the blame to the users by indicating that the hackers stole those passwords from some other servers and then used them to access Dropbox. But whatever the case is, the users cannot afford to compromise the security of their own data, therefore the attack was perpetrated either by an anonymous external attacker or it was an internal fault attributed to the CSP. In the light of above discussion, it is a dire need of current big data and cloud computing paradigm to handle to issue of data growth. Data deduplication is an effective technique exercised to handled the issue. The primary condition is that the data deduplication should be designed in accordance to the security and efficiency requirements of the system in consideration.

1.2 Data Deduplication Technical Design Issues

Following is an compact explanation of some of the main technical design considerations of data deduplication.

1.2.1 Types of Data Deduplication

- Hash Based: Hash based data deduplication methods use a hashing algorithm to identify the chunks of data after the data chunking process.

- Content Aware: Content aware data deduplication methods rely on the structure or common patterns of data used by the applications. Content aware technologies are also called byte level deduplication or delta-differencing deduplication. Key element of the content-aware approach is that it uses a higher level of abstraction when analyzing

the data [6]. In this, the deduplication server sees the actual objects (files, database objects etc.) and divides data into larger segments of sizes from $8MBto100MB$. Since it has the knowledge of the content of the data, it finds segments that are similar and stores only the changed bytes between the objects. That is why it is called a byte level comparison.

- Hyper Factor: HyperFactor is a patent pending data deduplication technology that is included in the IBM System Storage ProtecTIER Enterprise Edition V2.1 software [7]. According to [8], a new data stream is sent to the ProtecTIER server, where it is first received and analyzed by HyperFactor. For each data element in the new data stream, HyperFactor searches the Memory Resident Index in ProtecTIER to locate the data in the repository that is most similar to the data element. The similar data from the repository is read. A binary differential between the new data element and the data from the repository is performed, resulting in the delta difference which is stored with corresponding pointers.

Another aspect to be considered when dealing any deduplication system is the level of deduplication. Data deduplication can take place at two levels file or block level. In block level deduplication, a file is first broken down into blocks (often called *chunks*) and then the blocks are compared with other blocks to find duplicates. In some systems, only complete files are compared, which is called *Single Instance Storage (SIS)*. This is not as efficient as block level deduplication as entire files have to be stored again as a result of any minor modification to that file.

1.2.2 Inline vs Post Processing Data Deduplication

Inline data deduplication refers to the situation when deduplication is performed as the data is written to the storage system. With inline deduplication, the entire hash catalog is usually placed into the system memory to facilitate fast object comparisons. The advantage of inline deduplication is that it does not require the duplicate data to actually be written to the disk. If the priority is high-speed data backups with optimal space conservation, remove inline deduplication is probably the best option to choose from. *Post processing deduplication* refers to the situation when deduplication is performed after the data is written to the storage system. With post processing, deduplication can be performed at a more leisurely pace, and it typically does not require heavy utilization of the system resources. The disadvantage of post processing is that all duplicate data must first be written to the storage system, requiring additional, although temporary, physical space on the system.

1.2.3 Client vs Server Side Deduplication

Client deduplication refers to the comparison of data objects at the source before they are sent to a destination (usually a data backup destination). A benefit of client deduplication is that less data is required to be transmitted and stored at the destination point. A disadvantage is that the deduplication catalog and indexing components are dispersed over the network so that deduplication potentially becomes more difficult to administer. If the main objective is to reduce the amount of network traffic when copying the files, client deduplication is the only feasible option. On the other hand, *server side deduplication* refers to the comparison of data objects after they arrive at the server/destination point. A benefit of server deduplication is that all the deduplication management components are centralized. A disadvantage is that the whole data object must be transmitted over the network before deduplication occurs. If

the goal is to simplify the management of the deduplication process server side deduplication is preferred. Among many popular vendors such as DropBox, SpiderOak, Microsoft Sky Drive, Amazon S3, Apple iCloud and Google Drive, only SpiderOak allows to perform server side deduplication [9].

1.2.4 Single User vs Cross User Deduplication

Single User: The deduplication can be done on a single user side, where the redundancy among his/her data is identified and removed, but the single user data deduplication is not very practical and does not yield maximum space saving. To maximize the benefits of data deduplication, cross user deduplication is used in practice. This technique consists in identifying the redundant data among different users and then removes the redundancy and save a single copy of the data. According to [3], 60% of the data can be deduplicated on average for individual users by using cross user deduplication techniques. In order to save bandwidth, CSP and users of their services are inclined to apply client-side deduplication where similar data is identified at the client side before being transmitted to the cloud storage. However, the potential benefit of data deduplication in terms of storage space and storage cost can also have some associated drawbacks. In Feb 2012 Drop Box disabled the client side cross user deduplication due to some security concerns [9].

1.3 Thesis Contributions

The main contributions of this thesis can be summarized as follows.

- We have designed a secure cross-user client side data deduplication framework for cloud environment, made of file chunking module, convergent encryption module, indexing module, a searchable encryption module, and a data integrity verification module [10].

The proposed framework is meant to ensure the security and integrity of the data outsourced by the users, assuming that the CSP is semi-honest, i.e. honest but curious.

- We have designed and implemented an enterprise level data deduplication scheme for cloud environment, where a cross-user data deduplication is performed by the enterprise before outsourcing its data in the cloud; and a cross-enterprise data deduplication is performed by the CSP to further remove the duplicates, resulting in cost and space savings [11]. Proof of storage protocols (POR and POW) are also designed that allow the users to probe the CSP (assumed to be semi-honest) in order to verify the integrity of their data in the cloud at any time, and the CSP to ensure that the data is released to its true owner [12].

- We have designed and implemented a secure image deduplication in cloud storage environments through image compression, that consists of a partial encryption and a unique image hashing scheme embedded into the SPIHT compression algorithm [13]. Proof of storage protocols (POR and POW) for images based on the use of the SPIHT compression algorithm are also designed that allow the users to ensure that their images are stored securely in the cloud, and the CSP (assumed to be semi-honest) to authenticate the true owner of the images.

- We have designed and implemented a secure video deduplication scheme in cloud environments using H.264 compression, which consists of embedding a partial convergent encryption along with a unique signature generation scheme into a H.264 video compression scheme [14]. Proof of storage protocols (POR and POW) for videos are also designed that allow the users to ensure that their video data are stored securely in the cloud, and the CSP (assumed to be semi-honest) to authenticate the true owner of the video data before releasing it [15].

These contributions have been published in part in the following:

- "A secure data deduplication framework for cloud environments," *In Proceedings of the Tenth Annual Conference on Privacy, Security and Trust (PST 2012)*, (Paris, France), July 16-18 2012.

- "Secure Enterprise Data Deduplication in the Cloud", *In Proceedings of the IEEE 6th International Conference on Cloud Computing (IEEE CLOUD 2013)*, (Santa Clara, US), June 27-July 2, 2013.

- "Proof of Retrieval and Ownership Protocols for Enterprise-Level Data Deduplication," *In Proceedings of the 23rd Annual International Conference on Computer Science and Software Engineering (CASCON 2013)*, (Markham, Canada), November 18-20, 2013.

- "Proof of Retrieval and Ownership Protocols for Images through SPIHT Compression," *In Proceedings of the 6th International Symposium on Cyberspace Safety and Security (CSS 2014)*, (Paris, France), August 22 – 24, 2014.

- "A Secure Video Deduplication Scheme in Cloud Storage Environments using H.264 Compression," *In Proceedings of the 1st International IEEE Conference on Big Data Computing Services and Applications (IEEE BigDataService 2015)*, (San Francisco Bay, USA), March 30 – April 4, 2015.

- "Proof of Storage for Video Deduplication in the Cloud," *In Proceedings of the 8th IEEE International Conference on Cloud Computing (IEEE Cloud 2015)*, (New York, USA), June 27 – July 2, 2015.

In addition, "Secure Image Deduplication Through Image Compression" has been submitted for a possible publication in an special issue on *Security and Privacy in Cloud Computing* of Journal of Information Security and Applications.

1.4 Thesis Organization

The reminder of the thesis is organized as follows:

In Chapter 2, a literature review of data deduplication schemes that are relevant to the scope of this thesis is presented, and various algorithms that are involved in the design of our proposed data deduplication schemes are highlighted and their choices are justified.

In Chapter 3, a secure data deduplication framework for cloud environment is presented, along with a security analysis.

In Chapter 4, the design of our secure data deduplication scheme for small and medium size enterprises is presented, with focus on textual data only. A security analysis of our scheme, along with a proof of storage and a proof of ownership, are also presented.

In Chapter 5, the design of our secure image deduplication scheme for cloud environment is presented, along with the associated proof of storage and a proof of ownership.

In Chapter 6, the design of our secure video deduplication scheme for cloud environment is presented, along with the associated proof of storage and a proof of ownership.

In Chapter 7, thesis conclusions and some possible future work are presented.

Chapter 2

Literature Review

This chapter overviews some research works in the literature on data deduplication that are relevant to the scope of this thesis. Concepts are classes of algorithms that are relevant to the secured data deduplication schemes presented in this thesis are discussed.

For the sake of clarity, the following notations are adopted in this Chapter only:

- SDD Framework: Secure Data Deduplication Framework for Cloud Environments
- SEDD: Secure Enterprise Data Deduplication in the Cloud
- SID-IC: Secure Image Deduplication through Image Compression
- SVDS-H264: A Secure Video Deduplication Scheme in Cloud Storage Environments using H.264 Compression
- POR-POW: Proof of Retrieval and Ownership Protocols for Data Deduplication.

2.1 SDD Framework

The first task of any data deduplication process is to divide the data (here a file) into smaller units. This process is referred to as chunking and the resulting units are called chunks. There are several chunking algorithms proposed in the literature [16]. In our SDD framework, we are proposing to use the Two Thresholds, Two Divisors (TTTD) algorithm [17] to perform file division since it can effectively handle small insertions and deletions in a given file. Moreover, in [17], Kave and Tang have argued that the use of this algorithm will only affect the neighboring chunks in case of a change in the file content. To the best of our knowledge, the TTTD algorithm has only been used in data backup servers , but not in cloud environments. The TTTD algorithm is actually specialization of the *Content Defined Chunking (CDC)* algorithm, which makes use of the *Basic Sliding Window (BSW)* algorithm [18]. Unlike traditional chunking algorithms based on the distance from the beginning of the file with fixed size chunks, CDC algorithms are based on the content of the given file. A sliding window CDC algorithm such as basic sliding window (BSC), scans and fingerprints the content of the file. A chunk boundary is set if the distance to a relative point of data (i.e. the previous boundary chunk or beginning of the file) is equal to a pre-defined threshold t. This will allow such algorithms to effectively identify parts of data which are not changed, and hence improve the deduplication process performance. However, such algorithms may lead to chunks which are too large of or too small. In TTTD, this problem is dealt with by ignoring those chunks that are smaller than a given lower bound threshold, $T_{\min}$, and by breaking the chunks that are larger than a given upper bound threshold $T_{\max}$ into fix size chunks. TTTD has been proven to be one of the best performing data chunking algorithms on both random and actual large data files [16].

In our SDD framework, each chunk of the file is encrypted by the user to protect the

privacy of the enclosed data. To achieve this task, we propose to use the *convergent encryption* algorithm [19]. This convergent encryption algorithm utilizes the content of the data itself rather than a random cryptographic key chosen by the user, to produce the same cipher text at each iteration. This feature makes this type of encryption technique ideal for data deduplication schemes in which privacy requirements for users' data is to be taken into account. It should be noted that the use of other encryption schemes such as symmetric key encryption or public key encryption schemes requires the use of different keys by/for different users for the same plaintext resulting in different ciphertexts, making this unsuitable for cross-user data deduplication.

The convergent encryption scheme introduced in [19] was primarily designed for use in disk-based data deduplication systems, and not in cloud storage environments , where a need for efficient search and retrieval algorithms is a mandatory requirement.

In [20], it was argued that combining the convergent encryption algorithm with a data deduplication scheme is an attractive approach that could result in an infinite storage service capability, providing that the cloud storage providers who will be utilizing this approach be trusted and be granted full access to the users' data. However, no proof of this claim was provided.

In [21], Raykova *et al.* introduced a deterministic encryption scheme derived from a probabilistic public key encryption scheme, which is then used for secure database searches. Their proposed encryption scheme might also be useful in data deduplication. However, the data search process in the cloud storage using this encryption scheme is yet to be investigated.

Once the file has been divided into chunks and these chunks have been encrypted by the user (in order to protect the privacy of the embedded data), the encrypted data chunks need to be indexed for efficient retrieval by the CSP. For this purpose, we will be using the *B+ trees*. *B+ trees* are an effective data structure tool for indexing the data for quick search and

retrieval [22]. They were first proposed by Thwel and Thein [22] for indexing the data in the standard disk-based system, but their proposed approach did not considered the security requirement of the chunked data. We believe they can be used in cloud environments as well, for the purpose of indexing the data for quick search and retrieval of files by the CSP. To the best of our knowledge, *B+trees* have not yet been applied for data deduplication purpose in the cloud environment context.

In [23], Lillibridge *et al.* proposed an improvement to the disk lookup performance of a data deduplication algorithm by considering large-size chunks and but fixed-sized batches, and by comparing the id of a targeted (new) chunk with a few carefully selected chunks in those batches. As pointed out by the authors, the success of their approach, as pointed out b the authors, depends on the *chunk locality* , i.e. the tendency for chunks in the backup data streams to reoccur together [23]. However, using this approach may result in poor performance of the data deduplication algorithm in the sense that more and more replicated chunks will end up being stored as a result of running the data deduplication algorithm, in particular when cross-user deduplication is performed. As a follow-up, Lillibridge et al. [24] tried to address this issue by proposing a distributed setting that uses file similarity rather than chunk locality to apply data deduplication. However, this proposal requires multiple sets of backup nodes and "in the worst case, reads require one disk seek per chunk, along with any additional seeks required to find the location of the chunk by accessing the appropriate index" [24], making this impractical for most cloud storage settings. It should also be noted that the problem with the storage of replicated data still exist in this approach although the required overhead is improved compared to the authors' earlier approach.

Alternative solutions for data indexing and retrieval include the use of machine and learning-based techniques. In [25], Dinerstein *et al.* proposed a Support Vector Machine (SVM)-based technique to train individual deduplication rules. However, the performance

of their proposed approach is highly dependent on the defined rules and its underlying data set, making it unsuitable for a general cloud storage setting. Similarly, in [26], Benjamin *et al.* proposed the use of a Bloom filter to decide whether a piece of data is new to the system. They also make the same locality assumption for the streamed data as suggested by Lillibridge *et al.* [23]. Furthermore, their scheme is dependent on the availability of locally preserved cache over a given backup dataset. For these reasons, their proposed schemes may not be suitable for data deduplication in large and distributed cloud environments.

One of the key features of our proposed framework is the encryption of a data index of the deduplicated data using an asymmetric searchable encryption scheme [27]. The idea of using asymmetric searchable encryption schemes (as well as symmetric-based or multi users-based schemes) to provide privacy to cloud users against untrusted cloud providers was first introduced in [27]. In cloud storage environments, often, the party searching the data is different from the one writing the data. Furthermore, there might be many readers, but only a single writer of the data. Given these constraints, our proposed framework allows the user to outsource (respectively write) the data in the cloud and the CSP to search (respectively read) the data using an asymmetric searchable encryption scheme.

Symmetric searchable encryption schemes support the single writer and single reader setup whereas data deduplication scenario does not necessary support the same. Similarly, multi-user symmetric searchable encryption schemes work in a setup where there are many readers and a single writer for the data, and this setting may not be suitable for data deduplication in the cloud environment since the only reader is the CSP. Several authors [28] [29] have investigated ways to achieve complex search queries in the asymmetric searchable encryption scheme setting, including conjunctive searches. Issues related to the application of asymmetric searchable encryption schemes in practical systems have been discussed in [30], [31], [32], along with the use of such encryption schemes to guarantee the privacy of the

queries [33].

The last module of our framework deals with the use of proof of storage in order to verify the integrity of the data in the cloud. In [34], the proof of retrievability for large files is discussed. In [35], Ateniese et al. presented a model for provable data possession by untrusted servers. Their approach reduces the network traffic and communication from the server because the user does not need to retrieve the data to verify its integrity and only samples a few sets of data rather than the entire data set. But, to the best of our knowledge, these approaches have not yet been modeled in the cloud computing environment or data deduplication context.

2.2 SEDD Scheme

In our Secure Enterprise Data Deduplication (SEDD) Scheme, the first step is realized by a file chunking module, which is responsible for breaking down a given data file into a series of chunks of variable sizes. We propose to use $B^\star$ trees for keeping an index of the chunks. $B^\star$ trees are a variant of B^+ trees [22] which are commonly used in standard disk-based systems.

Different indexing schemes are proposed in the literature to secure the indexing process. In [36], it is argued that B^+ trees are efficient data structures that can be used to generate a secure index over a database. As a proof of concept, an indexing scheme is proposed, where nodes in the buckets are grouped and an homomorphic encryption is applied to these groups in order to hide access patterns from the adversary. However, this scheme does not consider applying data deduplication within the database because homomorphic encryption does not produce the same ciphertext for the same plaintext which is a basic requirement of data deduplication in our scheme. Rather, the focus is solely on protecting the access and search

patterns from the CSP. In [37], B^+ trees are used for indexing the data in the cloud, and a data chunking technique is used to break the data into different chunks that constitute the nodes of the trees. From these B^+ trees, a local index and a global index are built based on the structured overlay, with the goal of reducing the amount of data transferred into the cloud while facilitating the deployment of database back-end applications. Again, this scheme does not deal with security issues, and does not consider applying data deduplication within the database.

In [38], Curtmola *et al.* proposed a searchable encryption in the form of a cryptographic primitive that can be used to enable keyword-based searches on the encrypted database without revealing the keywords to the CSP. In the same vain, a similar type of algorithm (referred to as a *private keyword search*) was proposed in [39]. In [40], Boneh *et al.* introduced the first public key searchable encryption scheme, in which the users (and not the owner of the data) can search the encrypted data with the help of the public and private keys. In [41], Waters *et al.* implemented an audit log for database queries that uses hash chains for integrity protection and identity-based encryption. Their scheme used extracted keywords to enable searching on the encrypted logs. It was argued that this scheme could be extended to other types of applications for searchable encryption apart from logs. The majority of searchable encryption schemes work under a single user setting where only the owner of the file can issue valid queries for his/her data. In the multi-user setting, all authorized users can search the data with the help of private keywords. In [38], Curtmola *et al.* proposed a searchable encryption scheme where the single user setting is transformed into a multi-user setting via broadcast encryption. Since broadcast encryption is subject to a high cost primitive [39], it is not suitable for use in the cloud environment or in enterprise models where huge amounts of data are expected to be uploaded. Similarly, in [42], Chai and Gong proposed a searchable encryption scheme which allowed the user to search the encrypted data. Their scheme also

uses a tree-based structure to perform data integrity verification when the CSP is assumed to be semi-honest. One of its major drawbacks is that it works only for keywords that had the same prefix, therefore it is not suitable for our proposed scheme since it could not be used in our data deduplication setup phase. In [39], Yang *et al.* introduced an enterprise model which works in a multi-user setting within a single enterprise using the private keyword search. In addition, a searchable encryption method in the cloud , which uses bilinear mappings for query generation, was also designed and validated. For these reasons and the fact that our proposed deduplication technique is expected to save huge volumes of storage space in its cross-enterprise setup phase, we have adopted and adjusted the enterprise model and searchable encryption method introduced in [39] in our proposed scheme.

2.3 SID-IC Scheme

Data compression can be defined as a way to reduce the storage cost by eliminating the redundancies that are present in most of the files. Compression can be either lossy or lossless. Lossy compression reduces the file size by eliminating some unimportant data that will not be recognized by human user after decoding. This technique is often used for video and audio compression. On the other hand, in lossless compression, each bit of data inside the file is manipulated in such a way that the data size is minimized and the exact original data can be reconstructed from the compressed data after decoding. Since our focus is on natural images that are commonly known as non-medical and non-archival images, we have considered using lossy image compression algorithms in this work; more precisely, a lossy compression scheme with low bit rate and for which the size of the compressed data to be encrypted is as small as possible compared to that of the total compressed data. This is justified by the fact that compression schemes that satisfy these requirements have been

proven to be suitable for encryption [43].

Candidate such schemes include the quadtree-based compression algorithm [44], [45], the Embedded Zero Tree of Wavelet Transforms (EZW) compression algorithm [46], [47], to name a few. In our proposed scheme, we have used the SPHIT algorithm [48] for image compression, which is a variant of the EZW algorithm.

The EZW algorithm [46] has the property that the bits in the bit stream are generated in some order of importance, yielding a fully embedded code. It creates an hierarchy of coefficient bands called pyramid decomposition, where the number in the label band is the level of the pyramid [49]. At the first step, the wavelet coefficients are quantized by the coarsest quantization step. In subsequent steps, the differences between the original and reconstructed values from previous steps are quantized with finer quantization steps, so that the bits in the bit stream represent the wavelet coefficients at levels of increasing accuracy. Therefore, after the first level of the significance map is completed and the quantized values have been received, the image can be reconstructed no matter when the transmission is stopped. The algorithm then starts sorting the significant coefficients by selecting an initial threshold, which is reduced by a predefined factor at each pass. The SPHIT algorithm [48] adopts the same principles as the EZW algorithm, with the added feature that it can help classifying the important and unimportant parts of the compressed data. The important parts are those needed for decompressing correctly the image [47] whereas the unimportant parts are those that simply provide minor details of the image. Examples of image encryption schemes involving the use of the SPIHT-based compression algorithm are provided in [50], [51], [52], [53], [54], [55], [56], [57], [58].

In [50], Martin proposed a secured SPIHT scheme (so-called SecSPIHT) which combines a wavelet-based image compression method with and efficient encryption, resulting to a lower computational overhead of the encryption operation. In this scheme, a smaller number

of bits in the code domain is selected and encryption is applied to these bits in order to achieve confidentiality of the data content without sacrificing the compression performance. The ability of the scheme to compress and encrypt arbitrarily-shaped visual objects is also demonstrated. In [51], Martin et al., proposed an image compression scheme (so-called CSPIHT), which compression and encryption. CSPIHT is shown to encrypt all the significant bits representing the LIS and LIP coefficients during the first K sorting passes with an average of up to 0.40% of the bits encrypted for images data coded at 0.8 bpp. In their scheme, only the significant bits of the individual wavelet coefficients are encrypted in order to ensure the confidentiality of the image data.

In [52], Ravishankar and Venkateshmurthy proposed an image encryption scheme in which the sensitive regions of an image are selected by the user, then segmented, encrypted, and permuted. The outputs of this operation are further utilized by the user to construct the encryption information in the header of the image before sending it for upload. This scheme is unsuitable for deduplication due to the human intervention that occurs in the selection of the regions of the image, which may lead to having the same image with different sensitive areas. Similarly, Taneja et al. [53] proposed a SPIHT-based partial encryption technique to efficiently encrypt digital images. Their scheme is shown to reduce the processing time with an encryption of less than 0.5% of the data when the image is encoded at 0.8 bpp by encrypting the bits and their locations.

In [54], Lu et al. presented their so-called Red Green Blue scheme (RGB) that decorrelates the color component correlation of the image. Basically, the RGB scheme sorts the red, green and blue pixels in three different arrays and further apply an inter pixel displacement algorithm on those arrays. The resulting output is then used as input to a compression algorithm. This scheme is suitable for deduplication purpose since there is no additional randomness or information added by the human users. However, it was shown that the use

of the compression algorithm lead to excessive computational overhead. In [55], Hua-Qian et al. proposed an image compression scheme made of image encryption and compression algorithm based on properties of the discrete wavelet transform and the SPIHT coding. In their scheme, the confusion property is meant to retain the image details whereas the diffusion property is implemented to preserve the bits that contain the important information on the image for the purpose of image reconstruction. Similarly, in [56], Rengarajaswamy and Rosaline proposed an image encryption technique in which the image is first encrypted by means of a stream cipher. Then, the resulting output is further compressed using SPIHT, leading to higher encryption and compression rates for larger images.

In [57], Sengupta and Ghosh proposed an image encryption scheme made of two steps. An encryption of the image is first performed using a combined chaotic encryption algorithm based on the Logistic Map and discrete chaotic encryption method , itself based on the Chebyshev chaotic sequences. Next, the SPIHT compression is applied on the encrypted image, yielding a scheme that can resist against brute-force attacks and statistical analysis attacks while also providing a satisfactory compression ratio when different gray scale images are applied. Similarly, in [58], Zhang and Wang presented an image encryption scheme that consists in embedding the encryption algorithm into the CSPIHT coding method [51]. The bit values of different types resulting from the CSPIHT algorithm are judiciously changed to yield the reconstructed images with varying degradation degrees, which in turn allow by analysis to retrieve the most important bits in the CSPIHT compressed bit stream. It was shown that this scheme can resist to various types of attacks.

In terms of image hashing, media hashing schemes such as fingerprinting, digital signature, passive watermarking can be utilized to trace possible duplicates of the digital media by using a unique hash sequence associated with them [59]. Typically, these types of schemes are characterized as non-invasive, meaning that no additional information is embedded into the

data content in order to secure it further. On the other hand, watermarking schemes [60] are basically meant to protect the rights of ownership of the digital data and does not necessary involve database operations. In the context of this thesis, such media hashing techniques where identical images should generate identical hashes even after encryption, are desired in order to expect some accuracy in the deduplication phase. Few candidate images hashing schemes that have been designed to satisfy these types of constraints are schemes based on compression algorithms using Fourier transforms [61], schemes based on the Daubechies discrete wavelet transform [62], schemes based on robust meshes [63], or schemes that use the information already produced during the image compression and encryption steps (rather than being independent in nature), and which are well suited for partial encryption purpose. In [64], it was proved that JPEG and MPEG compression algorithms are not suitable for partial encryption. In [65], it was also proved that the Quad tree compression algorithm (resp. the Wavelet Embedded Zero Tress algorithm) can generate 13 % to 37% (resp. less than 2%) encrypted data out of the total generated compressed data, and compression algorithms with low bit rate are better when utilized with partial encryption. For these reasons, we use the SPIHT compression algorithm. A description and performance analysis of this algorithm are given in [65].

Based on our above discussion and our earlier works on data deduplication [10], [11], [12], we are proposing to use the SPIHT compression algorithm for image compression and the partial encryption to prevent the CSP from decrypting the compressed image. Further, the deduplication is performed by using the compression information (i.e. the SPIHT coefficients) to produce the unique hash of the image. This image hash is produced in such a way that the individual users will not add any independent information to the image signature; that way, identical images will produce identical hashes for secure and accurate deduplication.

2.4 SVDS-H264 Scheme

Videos are often compressed enough before being stored in any data warehouse. The most popular methods used for video compression include the MPEG and H.264 compression algorithms. The MPEG algorithm [66] is an object-based algorithm that makes use of local processing power to recreate sounds and images. On the other hand, the H.264 algorithm [67] is a block-based motion compensation codec most widely used for high definition (HD) videos. Its working is based on frames, which are further divided into macro blocks. According to [67], the H.264 algorithm yields a better motion compensation minimum clock size support and a better bit rate, compression efficiency, compared to the MPEG algorithm. For these reasons, we have adopted the H.264 compression algorithm in our proposed SVDS-H264 scheme in order to facilitate the video deduplication step.

In order to perform the deduplication on the videos, the CSP needs to compare the received videos against the ones already in the cloud storage. Different signature generation schemes for different scenarios have been proposed in the literature. In [68], two robust hash algorithms based on discrete cosine transform (DCT) are proposed, one based on the classical basis set and the other on the randomized basis set. The DCT hashes are robust, but are less secure as videos with similar hashes can easily be found. The random hash is more secure than the DCT hash, but it requires the use of a secret key, thereby it is not suitable for used for cross-user deduplication in our SVDS-H264 scheme. Moreover, this approach does not use video compression, thus may not be efficient in terms of storage savings when large videos are considered. In [69], a robust hash based on 3D SPIHT compression algorithm is proposed. The 3D discrete wavelet transforms are used to generate the hash of the compressed video, which is unique and robust. But, it is shown that the SPIHT algorithm is not as efficient and practical compared to the H.264 compression algorithm. In [70], a video fingerprinting

method based on the centroid of gradient orientations method is presented. This method is chosen due to its pairwise independence and robustness against common video processing steps such as lossy compression, resizing, frame rate change, to name a few. But, it requires that some important input parameters be given by the users in order to calculate the hashes; which makes it unsuitable for cross-user deduplication in the cloud since in this case, it is required that hashes be calculated by using some standard methods. Similarly, in [71], the hash signatures are also calculated based on symmetric pairwise boosting and radial projections of the key frames, but again the lack of compression algorithm and vital input parameters set by the users make such scheme unsuitable for cross-user video deduplication in the cloud. In [72], a video hashing scheme based on the H.264 compression algorithm is presented. In this scheme, the video is divided into frames which are grouped together as group of pictures (GOP) on the basis of some similarity. In doing so, we believe that the probability of achieving higher deduplication ratios increases if the video is considered in smaller groups rather than as a whole. In our proposed SVDS-H264 scheme, we have used the hash signature strategy described in [72].

In order to protect the videos from the semi-honest CSP or any external malicious users, the video has to be encrypted. Approaches for encrypting the H.264 videos can be grouped into four categories, namely, heavy weight encryption [73], permutation encryption [74], partial encryption [64], and perceptual encryption [69] approaches. In the heavy weight encryption approach [73], the entire video is taken into account and the whole sequence of bytes is encrypted, therefore, this approach is not suitable for cross-user deduplication in the cloud. In the permutation encryption approach [74], the bytes of the videos are interchanged, which may result in lower quality video. Moreover, this approach requires the user's input for randomizing the bytes, therefore, it cannot support cross-user deduplication. In the partial or selective encryption approach [64], only a certain portion of the video is encrypted in such

a way that unauthorized users cannot decrypt the video meaningfully. This approach has been shown to be efficient, cost effective, and suitable for cross-user deduplication purpose in the cloud [11]. In the perceptual encryption approach, the video is encrypted in such a way that only a certain amount of perceptual information can be touched, i.e. the encrypted video data is still partially perceptible even after the encryption, making it possible for the users to listen or view low-quality versions of the video. This type of approach does not fulfill the requirement of private data storage in the cloud.

Among the above described approaches, the partial encryption approach has been adopted for use in our work. Representative partial encryption-based methods are described as follows. In [75], a video compression algorithm is proposed, for which the sign bits of the DC coefficients of the Y, Cb, Cr blocks of I-frames and the sign bits of the motion vector in B and P frames are encrypted using a key. This encryption method involves the selection of some sensitive areas by the users, and thus is not suitable for cross-user deduplication in the cloud. In [76], the video is converted into Discrete Cosine Transformation (DCT) coefficients, then these coefficients are encrypted using a secret sharing method. The use of secret sharing ensures that there is no combination of the members of a group that can discover the secret. The motion vectors are then scrambled using a pseudo-random number generator. This scheme is also not suitable for cross-user deduplication in the cloud since it is required that the videos coming from all the users (not only from members of a group) be deduplicated across the cloud storage.

In [77], a selective encryption of the H.264 compressed video is proposed, which works on the GOP and various types of videos. For complex texture videos with high, medium, and low intensity motion vectors, all the intra prediction modes and non zeros motion vectors in 100%, 70% and 30% of the GOP are encrypted respectively. For the non-complex texture with high, medium, and low motion intensity vectors, all the intra prediction modes, 3

low band DCTs, and motion vectors in 100%, 70% and 30% of the GOP are encrypted respectively. This scheme requires no specific design input parameter from the users, thus is suitable for cross user deduplication. It is also shown in [77] that this scheme utilizes smaller amounts of data for encryption and is superior to the scheme proposed in [78] in terms of computational cost. Compared to the scheme proposed in [78], this scheme is also proved to be more perceptually and cryptographically secured. Therefore, we tailor this encryption scheme in order to achieve the security target for our proposed cross-user deduplication scheme.

To the best of our knowledge, we have so far come across only one paper in the literature that deals with video deduplication [79]. In this paper, Katiyar and Weissman proposed an application framework for video deduplication, in which ordinal signatures are used for video segments comparison, but no video compression algorithm is invoked. Moreover, their scheme is not secured against the data storage manager since the signatures are not encrypted by the users and the entire process is controlled by the database provider, who is given full trustworthiness. Also, their scheme performs destination deduplication - where the objects are compared at the servers side - rather than source deduplication - where the objects are compared at the client machine side. To the best of our knowledge, there has been no prior work reported in the literature dealing with video deduplication in the context of cross-user source deduplication, with the assumption of honest but curious CSP. This thesis proposes a novel scheme (so-called SVDS-H264 scheme) that incorporates a selective encryption, a H.264 video compression algorithm, and a unique GOP-based signatures generation scheme, for the purpose of secure video deduplication in the cloud storage environments.

2.5 POR-POW Scheme

In order fulfill the requirements of data security and data privacy in the cloud storage environment, three well known types of storage protocols in the data security domain can be envisaged: (1) a proof of data possession protocol (PDP) - which is run by the data owner to verify that his/her data in being kept intact in the cloud storage. In case of mishandling detected, the owner of the data can simply remove the data, but cannot correct it; (2) a proof of data retrievability protocol (POR) - which is run by the data owner to check the integrity of its data, and in case of any tempering detected, the data owner can retrieve the correct data by identifying and correcting the errors; and (3) a proof of ownership protocol (POW) - which is run by the CSP to ensure that he/she is releasing the data to the authorized users only. Notions of *PDP*, *POR*, and *POW* play important roles in cloud storage security. POR is a compact proof by a storage provider to a client that a target file F is intact, in the sense that the client can fully recover it [80]. This concept was introduced by Juels and Kaliski in [80]. PDP schemes allow the user to verify the integrity of data stored in the cloud without guaranteeing the ability to retrieve the file. This concept was introduced by Ateniese in [81] and further explored in [82, 83]. POW as discussed in [84] ensures the CSP that the security of the data has not been compromised by releasing it to the unauthorized users.

In the context of POR and POW protocols, one of the early works on POR schemes can be found in [85]. In this work, a distributed setting was proposed in which a file is divided into blocks which are assigned to different servers. These servers in turn verify the integrity of the data by exchanging with each other MACs of their corresponding assigned blocks. This work did not present the formal definitions of the proposed architectures, and no security analysis was provided. A very concrete work in the field of PDP is resented in [81] and

presents a scheme for proof of data possession that used homomorphic verifiable tags with blocks of files is presented, along with a security proof based on the game theory concept. The issues with such schemes are twofold. First, their requirements make them unpractical for the real world applications. Second, they do not allow for data retrievability in case of data corruption. In a follow up work [82], an extension of these schemes is provided in the form of a publicly verifiable version guaranteeing an unlimited number of queries between the user and the server. In [83], further improvements on these same schemes have been proposed, resulting to minimized overhead (almost constant irrespective of the file size at the server side). However, the question of their practicality for most applications is yet to be investigated. In addition, such schemes do not allow for data retrievability when corruption occurs on the data. Another work presented by Shachams [86] proposed a scheme in which homomorphic authenticator tags are used for the blocks of files. In this scheme, the bandwidth requirement by aggregating the tag values for a larger number of blocks. Their scheme also guarantee unlimited numbers of trials. Above all, none of the aforementioned schemes deals with data deduplication. In [87], error correcting code techniques were used to design a POR protocol, in which the concept of data sentinels was introduced. Sentinels are error-correcting code checks that are generated independently of the bit strings whose retrievability [87] is being checked and are randomly dispersed among the data being stored. An enhancement of this proposal supporting a fully Byzantine adversarial model [80] uses double encoding of files and achieves lower storage overhead. However, the insertion of sentinels into the original data blocks makes these schemes unsuitable for data deduplication. The main reason is that insertion of sentinels into arbitrary positions in the original data makes it difficult for the CSP to identify and discard similar data.

In [88] proposed a proof-of-storage scheme that supports data deduplication is presented. The data is segmented into small 'chunks' and deduplication tags for each chunk of the file

are calculated. Data and these tags are then outsourced to the cloud for authentication at a later time. However, this scheme assumes that the CSP is fully trusted, thereby he/she can access the plaintext data. It is also assumed that the security of this scheme is proven in a Random Oracle Model based on Computational Diffie-Hellman assumptions.

In [89] and [90] data operations at the block level are considered while applying the POR protocol. However, the Sobol random sequencing method [89] is used, which is not suitable for data deduplication since random arrangement of blocks will also make it difficult for the CSP to identify duplicated data. In [90], a POR scheme for POR for dynamic data using homomorphic tags and Merkle Hash Trees is also proposed. However, the tags for the chunks (of the file) are calculated and uploaded, leading to less practicality as pointed out in [83].

Proof of ownership schemes have been proposed in the context of data deduplication in [84], [91]. These approaches use the Merkle Hash Tree approach to provide a proof of ownership assuming a trusted CSP. In this thesis, the POR method [87] and the POW [84] are modified to accommodate the requirements of data deduplication (textual, images, videos) assuming that the CSP is semi-honest. To the best of our knowledge, our POR and POW schemes are the only ones combining data compression, deduplication and proof of storage protocols in a set up where CSP is assumed to be semi-honest.

Chapter 3

Secure Data Deduplication Framework for Cloud Environments

In this chapter, a new privacy preserving cross user client side deduplication framework is proposed. Our framework uses an efficient deduplication algorithm to divide a given file into smaller units. These units are then encrypted by the user using a combination of a secure hash function and block encryption algorithm. An index tree of hash values of these units is also generated by the user and encrypted using an asymmetric search encryption scheme. This index tree will enable the CSP to search through the index and return the requested units. Our proposed data deduplication framework for cloud environment is meant to allow the CSP to perform data deduplication techniques without the need to access either the user's plain texts or decryption key.

3.1 Background

Data deduplication identifies redundant data among different users and then removes the redundancy and saves a single copy of the unique data. Therefore, data deduplication offers

a unique opportunity to CSP to provide their users with more space at a low cost. However, the potential benefit of data deduplication in terms of storage space and storage cost also comes with some associated drawbacks. One of the major concerns of users and organizations outsourcing their data to cloud storages is the issue of data integrity and privacy. For example, cross-user deduplication can lead to information leakage to dishonest users. More importantly, cloud storage providers need to access the users' data to employ deduplication algorithms, which can result in serious privacy breaches for these users. Although there has been a lot of recent interest in providing adequate user privacy in the cloud [27] and improving existing deduplication techniques [17], there is currently no work that addresses both user privacy need and efficient data deduplication in an integrated fashion". In this Chapter, a novel secure data deduplication framework framework for cloud environment is proposed, which can ensure the privacy of the data in the cloud storage and can allow the CSP to perform data deduplication without compromising the data privacy and integrity.

The design of the aforementioned framework rely on several algorithms as follows. First, the two Threshold Two Divisor (TTTD) chunking algorithm [17] is used to perform the file (data) division into smaller units (chunks). Second, *convergent encryption* algorithm [19] is used to allow the CSP to perform the data deduplication. It is also meant to provide the data privacy against the CSP or any malicious user. Third, an indexing scheme is designed to generate an index of the encrypted file chunks for fast search purpose by the CSP. To this end, *B+ trees* [22] are used due to their efficiency as data structure tool for indexing the data for quick search and retrieval. Fourth, an asymmetric searchable encryption scheme [27] is used for encrypting the generated index of the reduplicated data. Sixth, inspired by the work presented in [34], proofs of storage protocols are designed to verify the data integrity in the cloud storage.

The rest of the Chapter is organized as follows. In Section 3.2, the above-mentioned components of our proposed framework are described in-depth. In Section 3.3, some discussions on the working of the proposed framework from both the users and CSP perspectives are provided. In Section 3.4, a summary of the Chapter is presented.

To realize our framework, we are combining the stand alone algorithms for purpose of building a complete structure for our deduplication framework. We are using the TTTD algorithm [17] to perform the file division into smaller units in the first part of our scheme. Secondly we are using the *convergent encryption* algorithm [19] to maximize the security of our scheme. Convergent encryption is playing a vital part in allowing the CSP to implement deduplication and at the same time provides data privacy against the CSP. In the third module of our scheme, we are generating an index of the data for fast searches. To achieve this, we need to index the encrypted file chunks for efficient retrieval by the cloud storage provider. For this purpose, we will be using the *B+ trees* since they are an effective data structure tool for indexing the data for quick search and retrieval of files [22]. The fourth module uses an asymmetric searchable encryption scheme [27] for the encryption of the generated data index of the deduplicated data. The last part of our scheme proposes to use proof of storage protocols in order to verify the integrity of the data in the cloud discussed in [34]. Further details of the background work can be found in Section 2.2.

The rest of the chapter is organized as follow: In Section 3.2, the proposed framework modules are presented in-depth. In Section 3.3, some discussions on the working of the proposed framework from both the users and the CSP perspectives are given. In Section 3.4, we conclude the chapter.

3.2 Proposed Framework Architecture

Our proposed framework is composed of five modules. These modules are the *File Chunking Module*, the *Chunk Encryption Module*, the *Index Generation Module*, the *Index Encryption Module* and the *Data and Index Encoding Module*. As stated earlier, we consider the problem of preserving users' data privacy in the case where cloud storage providers are considered semi-honest, i.e. honest-but-curious. In this setting, we cannot allow complete data access to cloud providers which is needed for standard deduplication techniques. To balance the need for storage efficiency for providers through the deduplication process and the need for users' privacy, our framework uses file chunk and chunk encryption modules to break files into smaller chunks and encrypt these chunks at the users' end. This ensures that the encrypted data is no more readable by the cloud storage provider. However, the encryption process is done in such a way that it allows the provider to perform deduplication on the encrypted chunks rather than the corresponding plain texts. The encrypted chunks are then arranged in an index generated through B+ trees in the next module. The B+ tree is further encrypted using an asymmetric encryption scheme. The reason for generating the B+ tree is to minimize the data search time taken by the cloud storage provider to search for and retrieve a particular piece of data. The B+ tree is also encrypted to keep the data private from the the cloud storage provider. There are a few basic assumptions regarding the functional requirements of the framework. Firstly, the user will have to download a client application on his machine. Secondly, for the purpose of file chunking, the user will submit the file to the application. The values for the parameters used in the TTTD algorithm will be assigned by the client application according to the guidelines set by the cloud storage provider and will depend upon the characteristics of the file such as size and format. Thirdly, the client application will use the same secure hash algorithm and block encryption algorithm

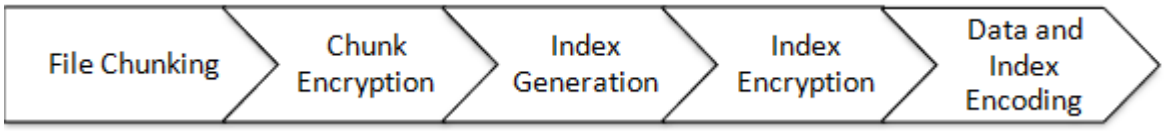

Figure 3.1: Flow of Data between Modules

across all the users. These measures will maintain uniformity in the processes of chunking and encryption and thus will help to achieve efficient data deduplication by detecting more similarities between the different users' data.

Through this setup we aim to achieve more storage space savings than in the set up where no data deduplication is applied and at the same time to keep the deduplication overhead as low as possible. In the following subsections, the details of each of these modules are described. The flow of information between the different parties is graphically described in Figure: 3.1

3.2.1 File Chunking Module

The file chunking module is the first module of the framework. When the user decides which file to upload to the cloud, he makes this file the input to this module. The output of this module is a series of chunks of the given data file. Here the file is break down into chunks of variable sizes. We propose to use the TTTD algorithm [17] to divide the file into appropriately sized chunks. TTTD is a two-threshold, two-divisor algorithm. The four parameters that are used in the TTTD algorithm are respectively: the main divisor D - which is used to find fingerprint matches; a secondary smaller divisor D' - which is utilized when the search for D-matches is not successful, a minimum chunk size threshold $T_{\min}$, and a a maximum chunk size threshold $T_{\max}$. Initially, the search window is moved until it reaches $T_{\min}$, at which point the fingerprint of the current window data is calculated. A comparison

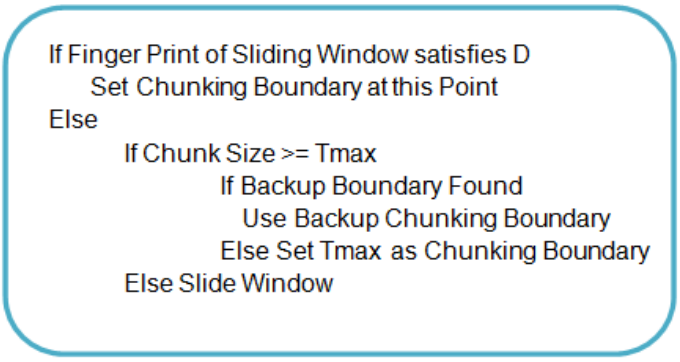

Figure 3.2: Pseudo-code of the TTTD Algorithm

is done at this step to see if the calculated fingerprint has D-matches, and the comparison is continued until the window size reaches $T_{\max}$. A D-match occurs when the algorithm compares a given number of bits in each fingerprint with the main divisor D and they are equal. Then, the point where the D match was found is marked as the partition point or the boundary. If the search for the chunk break point fails, within a $T_{\max}$ window, the same procedure is repeated using the backup D' parameter.

Note that D' has a smaller value than D. The chunks that are produced by this module are then sent to the Chunk Encryption module for further processing. The pseudo-code for the above mentioned the TTTD algorithm is depicted in Figure: 3.2.

In the TTTD algorithm [17], the values of $T_{\min}$ and $T_{\max}$ are setup in accordance with the expected average size of the chunk and the type of data used. Commonly used values for $T_{\min}$ and $T_{\max}$ range from 2KB to 16KB. In fact, when employing the TTTD algorithm to divide the file into appropriate size chunks, the above-mentioned four parameters are used for handling the following tradeoff: achieving improved deduplication ratios if working with too small size chunks - which in turn result in increasing associated overhead due to handling and maintenance (in terms of storage) of large amount of meta data - versus achieving improved overhead due to a decrease in the required meta data if working with large size chunks -

which in turn will result in inferior deduplication ratios.

3.2.2 Chunk Encryption Module

The chunk encryption module is responsible for calculating the cryptographic hash of the chunks based on the convergent encryption scheme [19]. The input to this module is the series of chunks from a specific file. The outputs of this module are the encrypted chunks and their hash values (so-called chunk ids), along with the meta data for each chunk. This meta data includes the file id to which the chunk belongs, the sequence number of the chunk in the file, and the chunk encrypted key.

The purpose of the chunk encryption module is to provide the cloud storage provider with the encrypted data only. The convergent encryption scheme [19] is a determinate mapping between the plain text and its corresponding cipher text - i.e. every time the convergent encryption is applied to a given plain text, the same cipher text will be produced. The basic idea is to use the chunks own content to calculate a cryptographic key, by applying a secure hash function to its content. This hash value will be referred to as the *Chunk Id*. Here, we will adopt the *SHA-1* hashing algorithm although any other secure hash function could be utilized as well, including the new NIST hash standard *SHA-3*. Using the *SHA-1* algorithm [92], the chance of hash collision for a 16 petabyte system has been shown to be less than 1 out 10^{24}.

After a similar block has been detected, an additional full block validation might be required [92] before the data get deleted. The above-mentioned Chunk Id (truncated if necessary) will then be used as the encryption key for the *AES* algorithm (or any other secure block ciphers) in order to encrypt the plain text chunk. In doing so, the user will store the encryption key, but the key will not be shared with the CSP.

3.2.3 Chunk Indexing Module

The chunk indexing module is responsible for generating the B+ trees, in which the nodes consist of the hash values of the chunks computed by the Chunk Encryption module. Each node will then have pointers to its chunk meta data as well. The meta data will contain the File ID to which it belongs to and the sequence number of the chunk in the file. The input to this module is the Chunk Ids for a specific chunk series belonging to a single file. The output of this module is a B+ trees consisting of cryptographic hash values in the nodes for the chunks, for a single file.

B+ trees are considered to be very fast in terms of searching due to their multilevel index structure, where there are maximum and minimum bounds on the number of keys in each index node. In a B+ trees, all records are stored at the leaf level of the tree; only the keys are stored in interior nodes. B+ trees are the most commonly used data structure for storing sorted data blocks in the context of file systems. Since we are also dealing with chunks of the files in our framework, and the chunk ids are the cryptographic hash values which can be sorted as simple numeric values, the B+ trees would be very efficient in terms of searching time for the chunk ids. The reason for proposing B+ tress for indexing is that they can reduce the search time from $O(n)$ to $O(\log n)$, by minimizing the full chunk indexing for identifying the new data in the data stream [22].

The ultimate target is to reduce the number of chunk pair evaluations, which can be achieved by using B+ trees since they have the same length from root to the leaf for all paths, and thus, can dramatically reduce the search time. Indeed, the number of intermediate levels within the tree is relatively small compared to that of other tree data structures. The most important feature of B+ trees is that the inter node connections are done by pointers, which in turn are very fast to access. Also, by using pointers, we are ensuring that each node can

access its neighbor nodes through the next pointers no matter how far way that neighbor node is. The root can have a minimum of two children only. Since the numeric hash values will be stored in the ascending order, the search process is expected to be very efficient due to the specific feature of the B+ trees algorithm.

3.2.4 Index Encryption Module

The index encryption module is responsible for encrypting the B+ tress (generated from the Chunk Indexing Module) using an asymmetric searchable encryption scheme [27]. This is done by the user for the CSP to search over the index and return the requested chunks. The input to this module is the B+ trees for the chunks of a single file. The output of this module is the encrypted B+ trees (our so-called the encrypted index).

The asymmetric searchable encryption scheme offers a suitable choice for the setup where the searcher is different from the owner of the data. One such scenario is known as MWSR [27]. The encryption under the MWSR setup is also a good match to the cloud storage problem in which many users write their data and encrypt their index, but the only reader is the cloud service provider.

The CSP will first read the encrypted index to find the matches so that deduplication can be performed and the data can be retrieved from the index whenever the request comes from the user. Although the asymmetric searchable encryption with the MWSR setup can provide efficient functionality in a number of applications, a number of issues with regard to security and performance will need to be taken into consideration before its use in the proposed framework.

- First, the asymmetric searchable encryption scheme proposed in [27] uses elliptic curves arithmetic for the evaluation of pairings rather than the evaluation of cryptographic

hash values. This operation turns out to be slow and requires comparatively more time and resources [27]. It would be desirable to find out computationally more efficient asymmetric searchable encryption schemes that do not use pairing, or to improve the performance of existing asymmetric searchable encryption schemes that use pairing. In [93], Bellare *et al* propose database encryption method that allows for logarithmic time search. But, the proposed protocols only works for a single-user setting, and its extension to a multi user setting is left as future work.

- Second, in the proposed asymmetric searchable encryption algorithm [27], the server has no knowledge about the content of the data as long as it does not have the token. However, once the server gets the token for keyword x, the server knows which encrypted documents contain x. In this case, the server can mount a dictionary attack against the token to find out which keyword the client is searching for, and thus, the server can consequently find which encrypted documents contain that keyword.

To address these shortcomings in our use of the aforementioned asymmetric searchable encryption scheme, the keywords are considered as the Chunk Ids, which in our case, are generated as cryptographic hash values. Given the onewayness properties of secure hash functions, it will be computationally infeasible for the server to find out the actual encrypted documents containing this encrypted chunk whose hash values is contained in the requested token. Furthermore, since the hash values are already encrypted, the data security in the cloud will be greatly strengthen.

3.2.5 Data and Index Encoding Module

This last module is responsible for organizing the meta data, the encrypted data chunks, and the encrypted index, in such a way that the user can verify the integrity of the data stored

on the cloud at any point of time using a proof of storage protocol. The proof of storage protocol is executed between a client and a server, where a client can ensure that the data is not tampered while it is stored on the server.

The proof of storage protocol operates as follows: the user first encodes the data and its index before uploading it to the cloud. Then, the proof of storage is run any time as desired by the user in order to verify that the server has not tamper the integrity of the data. In doing so, the number of information exchange between the client and the server should be kept minimal.

In our framework, the proof of storage protocol is meant to be privately verifiable. This will ensure that only the user himself can utilize his/her public key to verify the integrity of the data. Of course, this will be done in addition to the already publicly verifiable proof of storage where any party can take the client's public key and run the proof of storage over the data. Steps that can be taken to design/implement a proof of storage protocol in cloud environments (thus that might be suitable for our proposed framework) are available in [94], [95].

The dynamic proof of storage protocol [96] is another interesting alternative proof of storage scheme since it can enable the user to update the data while running it; however, if employed in our framework, the fact that the user is restricted to only run the protocol to verify the integrity of the data in the cloud rather than updating the data, should be taken into account.

3.2.6 Framework Summary

To summarize, our proposed framework is meant for addressing the privacy challenge raised by data deduplication in cloud infrastructures. Our framework is composed of five modules.

The TTTD algorithm is used in the first module to divide the file into chunks. In the second module, cryptographic hash functions are used to encrypt these chunks. In the third module, B+ trees are incorporated in order to generate the index which consists of the chunk ids in the nodes. In the fourth module, the resulting B+ trees are encrypted through the ASE scheme to ensure some efficiency in the data search for the Cloud Storage Provider in the cloud. Finally, the fifth module is used to verify the data integrity using proofs of storage.

3.3 General Discussions

We will now discuss the working of our framework from the user and the CSP point of view respectively.

3.3.1 User's Perspective

Let us consider the case where a user wants to upload some files to the cloud.

The user will start by downloading the client application to his machine. This application will then generate a cryptographic master key which will be kept secret from the CSP since it will be stored locally on the user's machine. This application will have three components namely the Data Processor, the Data Verifier and the Data Finder, which will interact as follows:

- Whenever the user wants to upload data to the cloud, the Data Processor is invoked. The Data Processor has the five modules described earlier in Section Proposed Framework Architecture.

- When the user wants to verify the integrity of data, he will trigger the Data Verifier. The Data Verifier will then use the user's master key to interact with the CSP to ensure

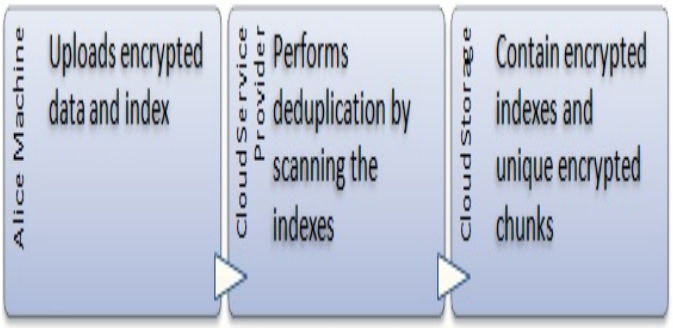

Figure 3.3: Data Proccessor

the integrity of data which reside in the cloud.

- When the user wants to retrieve files from the cloud, he will trigger the Data Finder. The Data Finder will then generate the tokens which will contain the Chunk Ids (chunk cryptographic hash values) for the files the user wants to retrieve.

- The CSP will receive the tokens and will search the encrypted index (i.e. B+ trees) for the requested Chunk Ids. It will retrieve the Chunk Ids and its meta data from the index, fetch the corresponding encrypted chunks, and return the requested encrypted chunks to the user.

- The user will utilize his private key to decrypt each chunk and reconstruct the file again.

The benefit of this scheme is that the CSP has no access to the user's plain text or the user's data index, but it is still able to improve the storage efficiency through the use of a deduplication process. The working of the Data Processor and Data Finder are illustrated in Figure: 3.3 and Figure: 3.4 respectively.

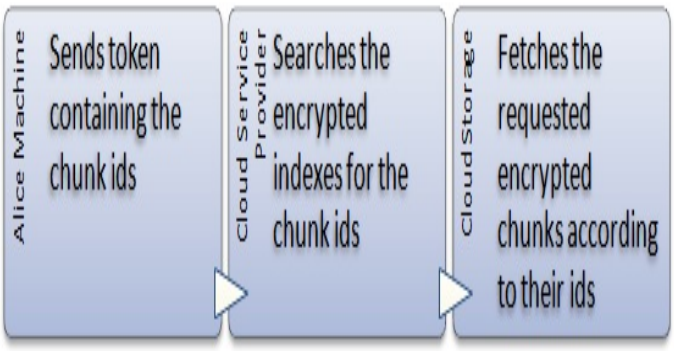

Figure 3.4: Data Finder

3.3.2 Cloud Service Provider's Perspective

Next, let us consider the CSP's part of the scheme.

- When the encrypted chunks and encrypted chunk index reaches the cloud, the CSP will search its own storage of encrypted indexes for any identical chunks.
- If there is any identical chunk detected, it will not store these chunks, and the duplicated data will be discarded with the pointers set to the original single copy of the chunk.
- The CSP will handle the meta data which will show that a new user is now authorized to access this chunk.

3.4 Chapter Summary

Data deduplication is an effective technology for removing data redundancy and saving storage space, but there are privacy and security issues associated with its use. We have designed a framework made of a combination of existing technologies, to address and handle the problem of securing the data deduplication procedure of cloud data storage. In our framework,

the deduplication process is broken into five modules: The first module is responsible for dividing the files into smaller units. The second module is responsible for encrypting these units using a secure hash algorithm and a block encryption algorithm. The third module generates an index of the encrypted units of the data. The fourth module encrypts the index using an asymmetric searchable encryption scheme. This encrypted index will then be used by the cloud storage provider to search the data. The fifth module encodes and arranges the meta data of the data and the index.

Our framework does not require cloud storage providers to be trusted when handling the users' data. It has also been designed to ensure the privacy of the data as it only provides access to an encrypted copy of the data to the CSP while enabling the deduplication of the encrypted data.

Chapter 4

Secure Enterprise Data Deduplication in the Cloud

This chapter proposes a novel two-level cross user, client side data deduplication framework that can be used in cloud storage by enterprises. At the enterprise level, the enterprise performs cross-user data deduplication and outsources its data to the Cloud. At the CSP level, cross enterprise data deduplication is performed by the CSP to further remove the duplicates, resulting in cost and space savings. Our framework is meant to allow the enterprise to facilitate operations such as searching over encrypted data, sharing data within the enterprise, and downloading data from the Cloud directly, in a secure and efficient manner without the need to trust the CSP. This Chapter also proposes a proof of retrieval protocol that allows the CSP to check the authenticity of the true owner of the data and a proof of ownership protocol that allows a user to check the integrity of its data against the semi-honest CSP or any malicious miscreant.

4.1 Background

The secure enterprise data deduplication presented in this Chapter is an instance of the scheme proposed in [97] in the sense that it focuses on the design of an enterprise model where it is assumed that different enterprises running the same type of business can use a single cloud, and each of these enterprises has its own internal users. The users belonging to a given enterprise store their data in the cloud using the enterprise server. A two-levels data deduplication scheme is introduced: one at the enterprise level, and the other at the CSP level. At the enterprise level, each individual enterprise performs its own data deduplication among its users (cross-user deduplication). At the CSP level, a second data deduplication is performed on the data submitted by the different enterprises to the cloud (cross-enterprise data deduplication). To our knowledge, no prior work has been done that deals with ensuring the the security of data deduplication in the cloud using an enterprise model where cross user and cross-enterprise data deduplication are both integrated. The merit of the proposed model is that it can be useful for small or medium size enterprises that do not have a huge number of resources and that intend to use these resources for other internal computations or operations rather than for storage purpose. The main design features of our proposed enterprise model can be summarized as follows:

- Enterprise Level Data Deduplication: We propose a solution for the enterprises that can help maximizing their storage savings in the cloud. This solution applies the above-mentioned two-level deduplication techniques, namely single and cross-user data deduplication at the enterprise level, followed by cross-enterprise data deduplication at the cloud storage provider level.

- Secure Indexing Scheme: $B^\star$ Tree-Based Secure Indexing Scheme: Certain file characteristics are used for indexing, with the goal to maximize the deduplication ratios.

The formats of the tree index and the data index are designed in such a way that data deduplication is supported at both the enterprise and CSP levels. Moreover, a convergent encryption technique is also used in the process of indexing, which is a critical part of the data deduplication process.

- Multi-user Searchable Encryption for Data Deduplication: The current state of deduplication caters to trusted servers who have access to the plaintext data. The data might be encrypted for secure transmission but the CSP can decrypt it to find the duplicates. In our proposed data deduplication model, a multi-user asymmetric searchable encryption is applied on the encrypted data in the cloud using a private keyword search that supports the data deduplication method. Using this encryption technique, not only the data owner has the capability of issuing search queries for particular files containing a particular keyword, but also any other user belonging to the same enterprise.

- Secure and Efficient File Sharing: We propose a method that allows the users to securely share their files with each other at the enterprise level without having to trust the CSP and without using additional resources other than the ones used in the searchable encryption for data sharing purpose. Our proposed File Sharing and Searching scheme is meant to ensure that the enterprise only uses its resources for storage saving and security purposes while the remaining functionalities associated with the file sharing and searching scheme are carried out by the CSP.

- Proof of Data Retrieval *POR*: Typically, a POR protocol is a compact proof by a CSP to a client that a target file F is intact in the sense that the client can fully recover it. POR schemes focus on allowing the CSP to verify that the user is indeed the owner of the requested file. Assuming that the semi-honest CSP can only have access

to perform cross-user client-side data deduplication in order to maximize the storage and bandwidth savings, our proposed POR scheme relies on the use of appropriate encoding and encryption strategies based on which a MAC scheme is derived, which ensures the security requirements for both the CSP and the users.

- Proof of Ownership (POW): Our proposed POW scheme design relies on the use of: (1) the Merkle Hash Tree (MHT) as underlying structure, (2) the $B^\star$ [22]- to keep track of the index of the data chunks, (3) the searchable encryption technique proposed in [39] and the data structures inherited from [84]- to satisfy the data deduplication requirements, and (4) the concept of data sentinels [87]- for the purpose of data integrity check. The proposed POW scheme enables the CSP to authenticate the users of the data with minimal computational and storage overheads.

The rest of the Chapter is organized as follows. In Section 4.2, our proposed enterprise deduplication framework is described, along with a security analysis. In the Section 4.3, the POR and POW protocols are presented and their security requirements are discussed. In Section 4, a summary of the Chapter is presented.

4.2 Proposed Enterprise Data Deduplication Scheme

Our proposed enterprise data deduplication framework is composed of three steps: In the first step, the data and its meta data are indexed in such a way as to ensure a complete data privacy against a semi-honest CSP. The second step consists in performing the multi-user private keyword searchable encryption on the encrypted data within a particular enterprise, keeping the searches and the resulting files secret from the CSP. Step 3 makes use of a strategy to support data sharing between users, by utilizing the existing meta data, the

indexing structures, and the searchable encryption scheme. Basically, there are three parties involved in our scheme. 1) The CSP who is hosting the storage services and who is considered as semi-honest in nature. The CSP is expected to act according to a predefined and agreed upon protocol, but he/she can be curious to learn about the data, thereby may accidentally delete or modify the stored data. 2) The enterprise is a small/ medium company outsourcing its data to the cloud for secure storage. 3) Users: The enterprise has employees who are authorized to store the enterprise data in the cloud in accordance to the rules governing the enterprise. Details are provided in the following.

4.2.1 Threat Model

There are many enterprises that share a single, common cloud storage provider. In these cases, each enterprise in the list uploads its data to the cloud, with the goals of minimizing data volume and maximize savings in terms of storage space. The enterprise will have many users under its banner and these users will upload their data via the enterprise server in the cloud.

The CSP is considered to be semi-honest, and is supposed to act according to a prescribed protocol. For this reason, the CSP cannot be completely trusted, therefore the data and its index must be encrypted. The CSP is trusted by the data owner to only return the encrypted files to the users who requested the files using the encrypted queries for a particular keyword. The keywords and the data are both encrypted and kept secret from the CSP. Another assumption is that the CSP and the users with whom the data is shared will not collude with each other. If this situation happens, the CSP will have access to all of the decrypted text of the data. In our analysis section, we have considered some of the potential attacks that can be carried out by malicious users. Finally, it is also assumed that the users from a

given enterprise in the list cannot share or search the data belonging to the users of another enterprise.

4.2.2 Secure Indexing Scheme

We focus on the design of a secure indexing structure for the encrypted data, with the property that even if the encrypted index and the data both reside in the cloud, it will not be possible for the CSP to access the underlying plain text. To achieve this goal, we propose to use the convergent encryption for chunk encryption. The reason is that this encryption scheme generates the same cipher text for the same given plain text.

The user will employ the SHA-3 hash to find an encryption key for a given chunk. This key will then be used to encrypt the chunk according to the AES scheme which is symmetric by nature. The key for *Chunk1* can be calculated as $K_{ck1} = Hash(CK_1)$ and the chunk can be encrypted as $CK_1 = AES(K_{ck1}, CK_1)$ where K_{ck1} is the encryption key derived from the chunk content itself, i.e. the encryption key is the hash of the chunk content. Therefore, for the same chunks, the encryption keys will be the same, and thus, the encrypted chunks will also be the same. Since the CSP is not allowed to access the pplain text he/she will only be capable of identifying the similarity from the *Chunk ID* and the content.

The *Chunk ID* (viewed as the global identifier of the chunk in the system) is the hash of the encrypted chunk i.e. $CKID = Hash(CK_1)$, where Hash denotes a hash function. Similarly, a global file identifier is needed as well. The *File ID* is the hash of the entire file content. This can be represented by $FileID = SHA-3(FileContent)$, where $FileID$ could be the hash of the file name. Since the file name is a shorter attribute which can easily be compromised, we use the file content as the file identifier (in our model) rather than the file name.

Next, the user will build a $B^\star$ tree (derived from a B^+ tree) which is composed of the chunks. The key attribute on which the tree is built is the *Chunk ID*. The $B^\star$ trees are used for indexing because they require only one disk I/O per level. As an example, for a B^+ tree of height 4, there are 300 million records for a fan out factor of 133, which is an average configuration [98] for the indices. Thus, the B^+ tree is converted to a $B^\star$ tree by ensuring that each disk page is 66% full.

$B^\star$ trees have the property that all the data reside in the leaf nodes only and intermediate nodes do not have any data. The leaf nodes will have all the information necessary about the chunk and will be used to reconstruct the file from the chunks. This information includes: *File ID* - which indicates to which file the chunk belongs, file type - which is the name of the type of file, chunk offset - which is the sequential number of the chunk in the file, and chunk length - which is the actual length of the chunk. File type is included in the data structure because we will be taking advantage of the fact that only the file or chunks of the same type can be duplicated [99]. This will save huge amount of scanning for the CSP.

Next, all the nodes in the $B^\star$ tree are encrypted using a single encryption key using the *AES* scheme, where Ind_{FKi} indicates the file index encryption key. It is generated by the enterprise for every file and for its owner. In addition, there is only one Ind_{FK} for a given file and this key is supposed to be possessed by the file owner at all times, and cannot be shared with the cloud storage provider. The encrypted index and data is now transferred from the user to the enterprise server.

On the cloud storage provider side, the meta data is arranged in such a way as to speed up the deduplication. This is performed both within the Meta data Storage and the Chunk Storage. The Meta data Storage has all the indexing information, is in the primary storage for frequent access whereas the Chunk Storage which has all the actual encrypted chunks and resides in the secondary storage. The Meta data Storage is responsible for handling

all of the meta data for the users. The cloud storage provider will arrange the files first as tagged under the enterprise, then according to the user id and under each user, the files will be arranged in a type-wise manner. Keeping the data with respect to file type, the cloud storage provider will then have to perform a limited number of computations in order to identify similar chunks. In [99], it is proved statistically that file type is a crucial factor in determining file similarities.

The cloud storage provider will first scan through the data within the enterprise that is requesting a new upload, then the data of other enterprises doing the same kind of business will be scanned, and finally, the data originating from the remaining enterprises will be scanned, which may create some important amount of overhead in processing. This overhead is expected because each user is allowed to encrypt chunks using its own encryption key, and no trust is put in the cloud storage provider to decrypt the meta data for identifying similar chunks. In the following, the operations involved in different modules of our proposed framework are described in some detail.

First Upload of the File by the Owner

The following patterns are passed on to the enterprise server by the user: the encrypted data along with the encrypted index, the root of the $B^\star$ tree (denoted by R), the File Id, and type and the un-encrypted keyword w for this particular file. The user will keep the File Id, the root of the $B^\star$ (R) and Ind_{FK} for future reference. The enterprise performs the deduplication by scanning the File Ids, and subsequently the chunk ids of the files submitted by their own users and the owner of the file. This is the first level of deduplication which is performed on the enterprise server to save storage cost in the cloud. This deduplication is basically the single user and the cross-user deduplication within the enterprise. There is a high probability that many users will be uploading the same files within the same enterprise.

Thus, the enterprise can maximize deduplication ratios. The scanning of the owner's data will save some computation time since deduplication within the single user is likely to give duplicates in the first place [99]. Once the unique chunks are identified by the enterprise, it sends the meta data of the unique chunks to the cloud. When the CSP receives the chunks from the enterprise, all targeted enterprises of the same kind of business will be scanned for the purpose of cross enterprise deduplication since it is possible to find the same file in that collection of enterprises. Many vendors such as Drop box [100]and SpiderOak [101] practice only single user deduplication in order to save some storage space (as opposed to using cross-enterprise deduplication). In order to determine the file level similarity, the CSP will first narrow down the search based on the file type. Next, the File Id will be matched against that of any file within the users accounts of the same type of enterprises. If the same File Id is found, then the $B^\star$ tree for that file will be scanned to find the chunk level similarity. If no file is matched, the next scan will be performed for all the chunks within the users' accounts of another collection type of similar type of enterprises in order to find chunk level similarities. The final step in the cloud will be to scan chunks belonging to the users in the remaining enterprises that have not been processed yet. After applying deduplication at the cloud storage provider side, the cloud server will ask the enterprise to only send the unique chunks with respect to its storage. The enterprise will then send the encrypted chunks, the encrypted $B^\star$ tree, the un-encrypted keyword w for the targeted file, and the file meta data i.e. the File Id and the file types of the unique chunks to the cloud.

Subsequent Downloads by the Owner of the File

In order to download the file later on, the owner will directly interact with the cloud storage provider rather than going through the enterprise server. The main reason for doing so is to relax the enterprise operations, allowing it to focus on utilizing its resources for data

encryption and index, as well as key generation for encryption, and searchable encryption purposes. In fact, all the meta data and data are generated and encrypted in such a way that the enterprise does not need to use its resources for frequent operations such as downloads and sharing of files among its users. First, the owner of the file will be authenticated by his/her identity (Id) in the cloud. This Id and password must be generated by the enterprise initially. The user requesting the file as an owner must be authorized before given access to the file since he/she has all rights to modify or delete the data. After the access control phase is resumed, the data owner will use the root of the $B^\star$ tree as the key and the File Id (also equal to SHA-3 (File X)) as the message for calculating the MAC. He/she will then send this MAC and the File Id to the cloud storage provider to request a download of that file. Since the cloud storage provider knows the root of the $B^\star$ tree and the File Id in the meta data, he/she will be able to calculate the MAC using this information. If the MAC from the user and that calculated by the cloud storage provider match, then it will be concluded that the user requesting the file is authenticated as the original owner of the file and the integrity of the $B^\star$ tree in the meta data storage in the cloud has not been compromised. The cloud storage provider will then release the file to the user. The advantage of using the $B^\star$ tree and File Id for authentication purposes is that neither the cloud storage provider nor the data owner requires any additional information. Moreover, the network traffic will not be burdened since only the MAC values will be sent over the network from both sides.

4.2.3 Searchable Encryption for Data Deduplication

The next fundamental component of our framework is to devise a searchable encryption scheme that can be used for the purpose of scanning the encrypted data in the cloud. Our analysis of the current-state-of-the art searchable encryption schemes revealed that the

scheme proposed in [39] can be adjusted and integrated in our framework in order to satisfy our needs. This scheme (referred to as a multi-user searchable encryption scheme) can allow a user other than the owner of the file, to search and decrypt a file within the same enterprise. A single encrypted keyword is associated to a file so that the scheme can be qualified as a private keyword search. Asymmetric encryption is then used by combining a public key and private key for the users. This scheme [39] satisfies the important properties of searchable encryption schemes established and proven in [39], namely, distinct query generation, efficient and complete user revocation, and query unforgeability. The distinct query generation property ensures that each authorized user has a distinct query key for generating valid keys; the efficient user revocation property is to ensure that the revocation of a particular user does not affect other users; and the query unforgeability property is meant to ensure that no one, including the semi-honest Cloud Storage Provider, can generate valid queries on behalf of another genuine user.

The searchable encryption framework proposed in [39] is made of five components, namely, the SetUp by the Enterprise, the Add User by the Enterprise, the Remove User by the Enterprise, the Write Record by the User, the Gen Query by the User, and the Search by the cloud algorithms. The workings of these algorithms are described in [39]. To use this framework for our purposes, some of these algorithms have been modified as follows. In our model, we use deduplication in a multi-user setting, thus, in addition to revoking a user from the set of authorized users pool, it is required that the data associated with this user be deleted. This involves resetting pointers if the revoked user is sharing data with other users. This change is to be implemented in the Remove User algorithm. In our scheme, in order for the Write Record algorithm to work, the user has to send the un-encrypted keyword w along with the file and associated meta data to the enterprise server to which he/she is associated. The enterprise will use this information and its master key to generate the index

for the file. In [39], the enterprise uses this index to perform the remaining encryption of the file whereas in our model, the user chunks the file and uses the convergent encryption to encrypt the file. For the index, the user utilizes Ind_{FK} i.e. the Index encryption key for a file) to encrypt the $B^{\star}$ tree index. This key is issued to the user by the enterprise for every file to be uploaded (the key is referred as *ek* in [39]). The user then passes the encrypted file and the plain keyword to the enterprise. $Enc(d_i)$ is the encrypted file and the index given by the user to the enterprise. The index $Indx(d_i.w)$ is calculated by the enterprise for the keyword. Therefore, both $Indx(d_i.w)$ and $Enc(d_i)$ are stored in the encrypted database residing in the cloud. The cloud will then apply cross-enterprise deduplication and request only unique data from the enterprise in order to save storage space.

Sharing of Files between Users within an Enterprise

The last component of our proposed deduplication framework is to allow the owners of files to share a particular file with any of the authorized users within the enterprise. In order to maintain the integrity of an enterprise data, it is assumed that the owners of the data can only share the data within enterprise to which they belong and not with external users. To implement file sharing between users, we will inherit the data and indexing structures proposed in [39]. The file sharing module will be integrated into the searchable encryption in such a way as to minimize the computational load on the user and the cloud storage provider. In doing so, the user and the owner of the file will not be required to deal with the enterprise server, but rather, will directly interact with the Cloud Storage Provider, allowing the enterprise to focus its resources on securing the encryption of the data and index, as well as handling user management and data deduplication. This can be achieve as follows.

Suppose that Alice is the owner of a file X, and she wants to share that file with Bob. Alice and Bob work in the same enterprise and are authorized by the enterprise to do

such transactions. First, Alice will prepare a message M, for the Cloud Storage Provider as follows: $M = (FileID, BobsID, DateofSharing)$, $M = Enc(AlicePrivateKey)$ and $M = Enc(CloudPublicKey)$. Second, the keyword w associated with the file X will be concatenated with M and sent to Bob. Third, Bob will calculate the query $q = (u, q_u(w))$ where u represents Bob, w is the un-encrypted keyword sent by Alice along with M, and $q_u()$ is the query generation key used by the query generation algorithm [39]. Fourth, q and M are sent to the Cloud Storage Provider, who will authenticate Bob and then Bob will decrypt M using his private key and then Alice's public key. This decryption is also meant to ensure that the message was originated by Alice. The message M will acts as an indication to the Cloud Storage Provider that Alice had the desire to share file X with Bob on a specified date. Fifth, the Cloud Storage Provider will then directly fetch the file X through its ID (without running any database scanning processes) and send back the encrypted file and its associated decryption keys which are in its possession. Bob will then calculate the decryption key using his own keys issued by the enterprise in order to read the file. This scheme will save some computation time for the Cloud Storage Provider since no additional data structures for the sharing operation are involved. Also, the Cloud Storage Provider does not need to scan through the entire database to find a single file X. Also, for the sharing of files to occur, the enterprise does not need to generate any new parameters.

Figure: 4.1 illustrates the operations between the Cloud Storage Provider (cloud) side, the user side, and enterprise side. In Figure: 4.1, the user who is an employee of the enterprise sends the encrypted chunks along with the associated meta data to the enterprise server. The enterprise in turn uploads the data to the cloud after applying enterprise level deduplication. Later on, the user can directly interact with the Cloud Storage Provider to download the file, search the encrypted file, or share the file within the enterprise.

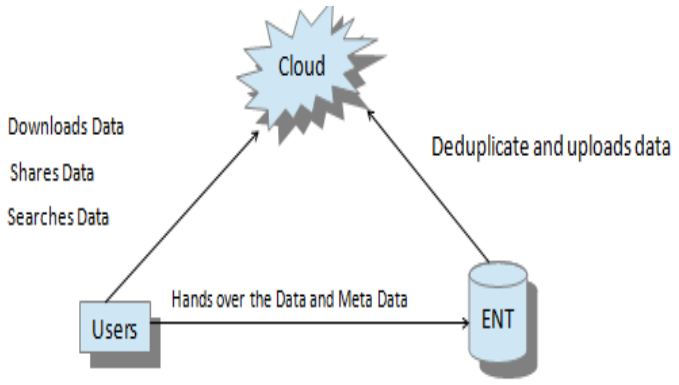

Figure 4.1: Operations between the User, Enterprise, and Cloud

4.2.4 Security Analysis

We consider attacks from the CSP who is considered to be semi-honest in our setting. There are two types of adversariess, namely internal and external. An internal adversary is an employee within the same enterprise and an external adversary is anyone else. Users working at different enterprises under the same cloud are considered to be external adversaries as well. If the CSP acts as an active adversary who technically cannot only intercepts the messages, but may also try to modify or change them, then the following threats can be considered.

1. If the cloud storage provider intends to tamper with the encrypted B* tree and the encrypted chunks, it will be hard to do any meaningful modification to the data without the corresponding keys. The data owner will be aware of any illegitimate access and modifications to the data as soon as he/she is no longer able to decrypt the data to obtain the plain text using the keys generated by the enterprise. Data integrity will also be protected by the additional security provided by the enterprise through incorporating the message authentication mechanism of a lower layer security protocol such as SLA. Moreover, the MAC algorithm will indicate any tempering and will thus ensure data integrity every time the data is requested by a user.

2. If the cloud storage provider can somehow access the encrypted B* tree and the encrypted chunks due to user id and password leakage, it will not be possible for the cloud storage provider to decrypt the B* tree because to do so, Ind_{FK} would be required, and this parameter is kept secret and is ignored by the cloud storage provider during the data uploading process, as well as during subsequent requests and the file sharing process. This parameter is generated by the enterprise and is thus considered to be safely generated and securely sent to the employees by the enterprise.

3. If the cloud storage provider somehow accesses the encrypted chunks due to id and password leakage, it will not be possible for the cloud storage provider to decrypt them since knowledge of the corresponding keys for all the chunks is required and these keys are already encrypted in the B* tree, as well as other information required to build up the file. Thus, the cloud storage provider will not be able to obtain any meaningful information from the encrypted chunks. The searchable encryption scheme introduced in [39] will be used to ensure query privacy. Indeed, query privacy indicates the volume of information that the cloud storage provider can extract from encrypted queries. Using the query privacy scheme [39] will ensure that, apart from the information that can be acquired by the cloud storage provider via observation, user queries will not reveal any other information to the cloud storage provider, i.e. the cloud storage provider cannot learn access patterns and search patterns from observations only.

The security of our scheme is based on the principle of Kerckhoff [102] which states that the security of the system depends on strength of its secret private keys, and not the secrecy of its design. If these keys are leaked, the privacy and integrity of the data will be compromised; otherwise it is very difficult to obtain any meaningful information from the encrypted data and metadata. Since we are proposing an enterprise model, it can be

assumed user id, password, initial security parameters, the keys generated by the enterprise for authorized users only, as well as Ind_{FKi} for a specific file are transmitted to the employees via a secured medium or mechanism. The leakage of any of these parameters will be the only way to compromise data integrity when using our proposed scheme.

Internal users who are employees within the same enterprise can access files in two cases. (1) First, when the true owner of file X performs subsequent downloads of that file: In this case, after the user has successfully passed the access control check and is recognized as the owner of the file, he/she will still require knowledge of the root of the B* tree of the file index, the file Id, and the Ind_{FK} of the original owner of the file in order to be able to decrypt it.These security parameters will prevail for all subsequent downloads/uploads of the file (as long as the user is still recognized as the owner of the file). If a malicious inside user has these parameters for any other employee, he/she will be able to access the files for that employee. (2) Second, when sharing of the file happens between the owner of the file and any other user within the same enterprise: If a malicious user wants to originate a message from Alice for sharing one of her files, he will need to know the file Id to prepare the message M. He will then need Alice's private key to encrypt M on behalf of Alice. In this case, a malicious internal user will be required to know the right keyword associated with the file in order to generate a valid query. (3) Third, when an internal user wants to attack a file during the searchable encryption scheme [39], the embedded query unforgeability method can be used to ensure that for any given user, no adversary is able to compute a valid query without knowing the query keys. Finally, whenever the data is transmitted from the enterprise to the cloud storage provider or from the user to the cloud (as shown in Figure: 4.1), it is sent in the form of random blocks with no meaningful order in their sequence.

We have analyzed three possible threats to our proposed model from the cloud storage provider perspective. Now, we discuss some of the attacks and their solutions. In [103],

and [104], a few attacks that a malicious user can perform have been discussed. In this proposed scheme, these attacks are also dealt with as follows.

In the so-called *Identifying Files* attack, Alice uploads a file to the cloud and Bob and Alice use the same cloud storage provider service. Bob knows the file and wants to know if Alice has already uploaded the file to the cloud. All that Bob has to do is to upload and backup a copy of the specific file since Bob knows that the file is unlikely to be in the possession of any other user. If the cloud storage provider generates a signal (in the form of an alert message) that the file is already in the cloud, then Bob is assured that Alice has the file in her possession. To address this problem, our enterprise model will attempt to identify a similar upload as done by Bob. To do so, a minimum number of chunks from the user will be required so that the user will not be able to identify that the file is already present in the cloud. This number could be set according to the state of the network bandwidth and time (high time or down time) of the upload.

In the so-called *Learning the Contents of Files* attack, which involves the use of a brute force attack on all possible contents of a file, Bob can start uploading possible versions of the file which he suspects Alice already has. Upon getting a signal (in the form of an alert message) from the cloud storage provider for duplication of a specific version of the file, Bob would be able to figure out the contents of the file that Alice has. This is true in the case of forms where only a few fields are changed and the entire document is almost the same. To address this problem, our enterprise model will again ask for a minimum number of chunks irrespective whether the file has been previously uploaded or not. Requesting the chunks would never let Bob realize that a specific file or specific contents of a particular file has already been uploaded by someone within the enterprise. Doing so will requires additional bandwidth between the enterprise server and the user.

In the *Poison Attack* attack [104] a malicious user, Bob somehow obtains the hash value of the file F. Using this hash as a file name, he uploads the wrong file. Later on, Alice wants to upload the file F using the hash value of the file F. The cloud storage provider tells her that the file has already been uploaded. Alice deletes her file and the cloud storage provider does the deduplication. Later on, Alice downloads the file and finds that the content of the file was poisoned and she has the wrong file under the right name. To address this problem, our enterprise model will use the chunks and the B* tree as a source of verification that a user really possesses the file at the time of uploading. Our scheme requires any user to present the B* tree index and unique encrypted chunks in order to ensure that he/she is really the true owner of the right file. Our scheme is dependent on the true contents of the file rather than a file name and its hash value. Using the B* tree as an index at the time of uploading further authenticates that this particular user indeed has the entire file.

4.2.5 Asymptotic Complexity Analysis

The performance of the system is linked to the computational load at the enterprise side and at the cloud storage provider side. Since the enterprise also has at least a fair number of resources, there should not be any problem for the enterprise as well. Moreover, our framework will help the enterprise to use its computational resources on the indexing, encryption, and key generation for the search only. Later on, the users can directly interact with the cloud storage provider to access the data without involving the enterprise. The user will chunk the file and generate the B* tree index for the file chunks. These chunks are encrypted using convergent encryption at the user's side. For a B * tree of order b and height h, inserting a record requires $O\ (\log_b n)$ operations, where n is the number of chunks of the file, which in turn is dependent on the size of the file. This index generation will further be eased by

applying a bulk loading to the B* tree where all the chunks will be sorted according to their Ids and the generation of the tree will occur. This will help reduce the I/O of the enterprise server. For the purpose of query generation on the user's side, the computational power used is *O(1)* [39]. To complete the search for the queries, the cloud storage provider does a paring operation and m hash operations, where n is the number of files. Thus, the computational load on the cloud storage provider is *O(m)*. As far as sharing of files among users is concerned, the cloud storage provider does not scan all the entries for the enterprise and only has to do calculations for a requested file. For downloading a file, a user has to calculate the MAC and the cloud storage provider also has to calculate the MAC using the root of the B* tree. These operations lead to negligible computation cycles when performed on huge cloud servers and enterprise servers. In other words, our framework offers a balance distribution of computational load between the cloud server and the enterprise server while maintaining a tradeoff between data privacy and reducing costs of storage through data deduplication.

4.3 Proposed POR and POW Schemes

We are proposing a scheme for private data deduplication protocol in the cloud storage setting. We would like to highlight the significant technical difference between public and private data deduplication protocols [84, 91]. In private data deduplication protocols, we consider the data to be encrypted by the user first before the upload, with keys not being shared with the cloud provider, whereas in public data deduplication protocols the data is either uploaded either in plain text form or encrypted with shared keys between users and the cloud storage provider. In the former, i.e. private data deduplication protocols, the impact of encryption on users' data in different can make cross-enterprise deduplication attempts challenging.

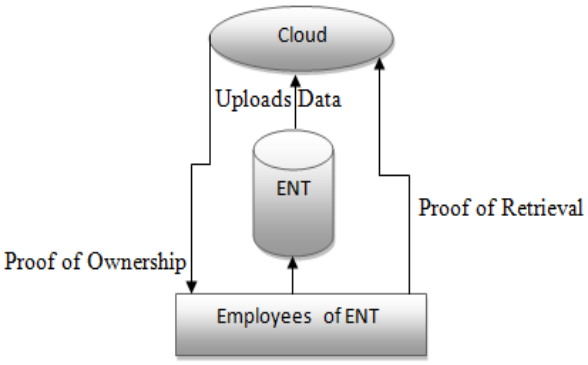

Figure 4.2: Flow of POR and POW Protocols among the Cloud, ENT and Users

The employees are the users of the cloud storage services and are authorized to run the POR protocol with the cloud for assuring the integrity of the data in the cloud. These users will also be scrutinized by the cloud for the ownership of their data via the POW protocol. Figure: 4.2 depicts the operational flow for POR and POW protocols involving all the three players. There are several basic approaches to verify the integrity of stored data in the cloud. A naive approach is for the user to download the file and verify its integrity. In most application, this represents very expensive approach in terms of bandwidth use and hence it does not represent a viable option. Another simple approach requires a user to compute a keyed hash $h_k(F)$ for a given key k and a file F. The user would then upload the source file F to the cloud, while retaining the hash value $h_k(F)$. For the POR, the user would send the key used to the cloud storage provider and ask it to compute and resend the hash of the file. By keeping many hash values and their corresponding keys, the user can run this POR protocol multiple number of times. There are several drawbacks to this approach including the need for the cloud storage provider to compute the hash value of the entire file and the linear correlation between the number of the keys and the hash values kept to the number of POR inquiries which can be make. In the next section, we will present an economical and secure scheme for a POR protocol in which a user can verify the integrity of his data against

the cloud storage provider who is practicing data deduplication at the enterprise level.

We first list a set of performance requirements for a POR protocol over a private data deduplication scenarios:

1) the protocol should use the common public, secure functions to ensure client-side, cross-user data deduplication in order to identify cipher texts corresponding to the same underlying plain text. 2) the protocol bandwidth use should be significantly lower than what is needed to download typical file requested. 3) The protocol should incur minimum computational overhead on the cloud and the cloud. Ideally, the protocol remove or limit the need of operating on the entire file, as necessary. 4) The protocol should also either not require extra storage or limit extra storage needed to perform POR for users and the enterprise. In the next section, we will examine the performance of our proposed *Private Data Deduplication protocol for Proof of Retrieval* - hence forth referred to as *PDD-POR* with respect to the four criteria listed.

Our protocol needs to address a number of issues: First, the CSP is not trusted, all the data and its corresponding indices need to be encrypted. Given that the protocol has to also enable cross-enterprise level data deduplication in the cloud, we need to introduce a key management scheme which is independent of the cloud storage provider. We propose to use convergent encryption [19] for the purpose of independent encryption keys across the enterprises. The user will only have to have the index encryption key irrespective of the file size. Moreover, at cost of some metadata per file, the user can verify the integrity of the file with the help of an aggregated MAC. Such an approach will remove the need for the cloud storage provider of traversing the entire file for each iteration of the POR. In short, PDD-POR protocol divides the files to smaller blocks of fixed sizes, i.e. 'chunks'. A convergent encryption is then applied to each chunk and the result in indexed using a scheme such as the B* tree index [105]. Then, each chunk is encoded with erasure codes such as Turbo

encoding scheme to protect against corruption of small portions of the file. In the next phase, the chunks are classified into segments and then an aggregated MAC is calculated for each segment. The encrypted blocks and the encrypted index is then put in the cloud and the aggregated MACs are kept with enterprise. In the following, we will explain in detail each step, and discuss the logic for their selection as well as their performance.

4.3.1 Private Data Deduplication - Proof of Retrievability

File Encryption

To preprocess a given file F for deduplication, the protocol breaks down the file into chunks C_i, $F = C_1, C_2, ...C_n$ for some n using the TTTD algorithm [16]. Each chunk is then encrypted using convergent encryption [19]. Convergent encryption is an encryption scheme which uses the content of the data as the encryption key, which make it suitable for data deduplication, as the encryption of similar plain text chunks across enterprises will result in the same cipher text. This would enable the CSP to apply cross-enterprise deduplication on duplicated (cipher text) data.

Let $K_i = Hash(C_i)$ be any secure hash, such as SHA3 of the chunk C_i. Then K_i will be the encryption key used for the *Chunk* i, and can be used (subject to some post processing) with any symmetric key encryption algorithm such as AES. The resultant encrypted chunk is now $C_{K_i} = SymmetricEnc(C_i, K_i)$. The hash of C_{K_i} with be stored with and used by the cloud storage provider to identify identical chunks. The global identifier of the file is $File_{ID} = Hash(File_p)$.The encrypted chunks are then indexed using a B* tree and the entire index tree is encrypted with an encryption master key for that file. The details of the index formation can be found in our previous work [105].

Error Correction

PDD-POR protocol applies an error-correcting encoding to the encrypted file chunks. This is to ensure that if a size able fraction of the F is uncorrupted as guaranteed by the encoding, then the file F can be recovered in whole with high probability by users. We are suggesting that a turbo Code encoding scheme is used for the purpose of erasure coding. This would result in achieving a random like code design with just enough structure to allow for an efficient iterative decoding method [106]. These codes have exceptionally good performance as compared to the classical codes particularly at the moderate BER and for large block length [106]. The block length of 4 Kb is considered to be a reasonable choice for good results. Another attractive feature of turbo codes is that they are actually composed of two or more simple constituent codes arranged in the variation of the concatenation scheme [106]. For this reason, it is possible to employ simple SISO decoders for each constituent code in an iterative fashion in which the soft output values of one of the decoders are passed to the other and vice versa until the final decoding is achieved [106]. This would allow for almost maximum likelihood decoding performance. The basic structure of the turbo encoding consists of an information sequence $X = (X_1, X_2.....X_n)$, X_i's are bits, two systematic feedback recursive convolutional encoders and an interleaver that 'randomizes' the order of inputs to the first encoder, before encoding them to get the second output stream. We will append at the end of the information sequence X, m termination bits to return the first encoder to all zero state so that the recursive effect will not produce different outputs for same input streams. Now the information sequence becomes X of length $n+m$ bits and is represented by the vector $(X_1, X_2.....X_{n+m})$. Since we are using the systematic encoding where the input stream also appears as a part of the output stream, the first input sequence is transmitted, then the second sequence is produced by the first encoder. The

input sequence goes through the first encoder which generated its parity sequence denoted by $(Y_1, Y_2.....Y_{n+m})$. In many turbo code encoding settings, the use of a random interleaver is expected to produce better results. However, using the randomness of output of such interleaver will hinder the deduplication process. Therefore, the interleaver used in our protocol uses an odd-even strategy to permute the input bit stream. The mechanism is that first the input bits are left unchanged before encoding, but only the resulting odd-positioned bits are stored. Then, the input bits are interleaved using fixed bit permeation and encoded again, but this time only the even positioned bits are stored. This would result in the third encoded stream is represented by $(Z_1, Z_2.....Z_{n+m})$. The final transmitted stream sequence is represented by the vector $(X_1Y_1Z_1, X_2Y_2Z_2....X_{n+m}Y_{n+m}Z_{n+m})$. The generator matrix for both the encoders of the turbo encoding scheme will be specified by the CSP for the users to maintain uniformity at the enterprise level. The matrix can be made different for different types of the files since the deduplication is expected to yield maximum ratios on the basis of file types [99]. Narrowing the search by file type maximizes the deduplication efficiency since the files of the same type can only be identical in content. We apply systematic error correcting version of turbo coding due to the fact that systematic encoding of the chunks has many benefits for our framework. The systematic encoding is the one where the original input stream remains unchanged and appears in the output as well.

The choice of the encoders and the generation matrix are specifically made to transform this into systematic encoding. The code rate, which is the portion of the useful and non-redundant data stream, of the turbo codes is 1/3 since we are using three sequences in the output stream. If the POR protocol indicates that some parts of the data file are not intact, the error decoding (turbo decoding) scheme can help retrieving the original file.

Message Authenticated Code (MAC) Generation

The user will now prepare the metadata used as part of the POR protocol. The user will virtually divide the file into q segments. There will be a specific number of chunks, say v in each segment $S = s_1, s_2.....s_q$ of a specific file. The user will select a PRF to generate random indices of the position of the chunks in the file denoted by $P = p_1, p_2.....p_n$. For example, suppose that for file F, the user set the value of v to 4. The randomly generated positions P are broken into the groups of 4. A set of MAC values is calculated denoted by $M = m_1, m_2.....m_q$ where m_1 is the aggregated MAC of the chunks in the segment s_1. The key k for the MAC is kept secret from the cloud storage provider and is only known to the user. The encrypted and encoded chunks are uploaded to the cloud along with the set M through the enterprise server. The user will keep the seed for generating the random index positions used by the PRF .

The next three phases includes a *challenge phase*, a *respond phase* and a *verify phase*. The user initiates a challenge to the cloud storage provider by specifying the randomly selected values of the encrypted and encoded chunks. The user calculates these indices of the chunks in a specific segment based on the value of v and the PRF seed. For instance, the user generates the indices with the seed of PRF and knowing v, divide the indices into the group of v segments. He sends these indices, the index of the m_i in M to the cloud storage provider in the form of $(p_1, p_2.....m_n, m_i)$. The cloud storage provider gives a response in the next response phase by returning the data chunks at the positions $P = p_1, p_2.....p_n$ and the MAC at the position m_i from the set M in the form of $((m_i), (C_{K_1}, C_{K_2}.....C_{K_n}))$. The user can verify the storage of the data by calculating the aggregated MAC of the chunks in the response. If both are equal to each other then the POR is successful. In case of any discrepancies, the user is also able to retrieve the file through error encoding used.

4.3.2 Private Data Deduplication -Proof of Ownership Scheme

We now present a Private Data Deduplication- Proof of Ownership (PDD-POW) which is based on our PDD-POR scheme set up, and hence does not require any additional overheads. It would enable the cloud storage provider to ensure that the user requesting a file is actually the owner of the file. We use a MHT to provide a POW of the data in question. The MHT approach was also used in [84], but the assumption made in that scheme was that the cloud provider is trusted and has access to the plain text data. Our PDD-POR protocol required that the user calculate a set M consisting of aggregated MACs of the randomly selected chunks in the form of segments. This set M is stored by the user. Since the set M is supposed to be used by the owner of the file to verify the POR, it is reasonable to expect that it should also be in the possession of the true owner of the file and that this information can be used in a POW scheme. Our protocol will be based on binary MHT. Suppose MT_{bh} is a binary MHT over the M using b leaves and the hash function h. The leaves of the MT_{bh} are read from left to right and are the hash values of the elements in the set $M = m_1, m_2.....m_v$. Due to its binary nature, each node has two children. The leaf nodes L of the MT_{bh} are $H(m_1), H(m_2).....H(m_v)$. Any non-leaf node NL of the MT_{bh} is calculated as $NL_i = h(L_{ir}, L_{il})$ where L_{ir} and L_{il} are the right and left child nodes of NL_i. Going upwards in this fashion, one can reach the root of the MT_{bh} by hashing up the values of the child nodes for each internal node. The user calculates the MT_{bh} over the set M and sign the root $Rt(MT_{bh})$ of the tree. The user then sends the root of the MT_{bh} and the set M to the cloud via the enterprise server in the first upload of the file. The cloud calculates the MT_{bh} over M and signs the root as this root is the same as sent by the user. Now to run the POW, the cloud storage provider chooses a random j belongs to M and sends the leaf index j to the user. The user provides a sibling path from the selected leaf index j to the root.

This path of internal nodes should be sufficient enough for the cloud storage provider to reconstruct the tree and verify the already signed root. If the root signed by the cloud and the user previously is the same as the new one calculated by the path provided by the user, the cloud storage provider verifies that the user is the true owner of the file. An advantage of our approach, as compared to other proposed solutions in the literature, is that it does not require large amount of storage to provide power of ownership.

4.3.3 Security Analysis of PDD-POR and PDD-POW

One of our key design requirements of our protocols was the ability to resists attacks from semi-trusted cloud storage providers and malicious inside users. We are assuming that the employees from different enterprises are not allowed to access each other data. A key component of the security of our proposed protocols relies on the fact that only encrypted data are outsourced to the CSP, and the responses to the queries made are based on pseudo random selection of this encrypted data.

We first focus on the security of the proposed POR protocol. An important consideration is the number of times the user can successfully run the POR protocol for a particular file against attacks from semi-honest cloud providers. This number is closely related to the parameter v, the number of chunks in one segment.

Once the user has used all the challenges in the set M, then he will not longer be able to run the protocol without recalculating the set M and updating it in the cloud. Recalculating the set M is not an expensive task at the enterprise level. For the files which are considered to be more sensitive in terms of security, the value of v can be set to be smaller like 3 chunks per segment. For less sensitive files, the segment size can be as big as 6 chunks. The randomization of the indices of the chunks depends upon the PRF generating the indices.

We are assuming that we are using a secure PRF and the seed used is kept secret from the cloud. For example, for files of size 4MB - a typical size for relatively common and smaller textual files at the enterprise level - the number of chunks per segment can be chosen to be 3. After encoding the file using turbo encoding the size expands 3 times due to turbo codes 2 encoders and systematic nature. The resultant size is now 4 times 3 = 12 MB. Using a chunk size of 4 KB - a very common choice in data deduplication and other applications - the number of chunks becomes *(Size of the File in KB/ Size of Chunk in KB) = 12 * 1024 (KB) / 4 (KB) =3072 chunks.* Dividing them into the segments of 3 yields 1024 challenges. This implies that for the files of size 4 MB, the user can have up to 3072 chunks and 1024 segments. The 1024 segments equals 1024 challenges over the lifetime of the file. The underlying structure of the challenges further makes it secure, since the chunks in the segments are randomly chosen and the MAC calculations done by the user in the set M are shared with the cloud in an encrypted. As for malicious internal users, our protocol requires authentication by the enterprise for the employee account to interact with the cloud under an enterprise banner. Furthermore, the malicious user needs to have the decryption key for the set M and the seed for the PRF to generate the indices for the required file.

Next, we consider the security of the proposed POW protocol. We are assuming that the user has the MT_{bh} and that the root of the tree has been signed by both the parties at the time of first upload. Now, the cloud storage provider can run the POW as many number of times as there are leaves in the MT_{bh}. The tree is built on the same encrypted set M, and in the previous example, we calculated that for the file of size 4 MB we have 1024 segments. In POW protocol, we used the aggregated MAC values for these 1024 segments as the leaf node values of the MT_{bh}. The cloud storage provider can run the protocol 1024 times since it is assured that there are 1024 unique paths for 1024 leaf nodes of the MHT. The upper bound of both the POR and POW are good enough for approximately a single challenge

per day for 3 years. To increase security of some sensitive files, the user can customize the set M for each file by altering the value of v. Making v smaller yields a larger number of challenges at the cost of more computational and storage expense. It is noteworthy that our protocol is based on using chunks of files and not the entire files as single piece of data. This property not only makes our system more secure than other work in the literature for which corrupted parts of file may not impact computation of POW. Moreover, the turbo coding use can allow for ability to recover some chunks in in case of minor corruptions.

4.3.4 Efficiency Analysis of PDD-POR and PDD-POW

The proposed PDD-POR protocol is efficient in terms of storage since it is only requires the user to store a set M and the seed of the PRF to generate the indices used as an additional information needed by the protocol. This storage requirement can be considered minimal at the enterprise level, while providing some concrete assurance for the retrieval of the files as well as storage integrity. The protocol also requires transmission of the user specified chunks by the cloud storage provider. For a single iteration of the POR protocol, it would result in v segments of specified size. For the POW protocol, the user is already has the set M of the MACs, he can encrypt it and construct the MT_{bh} over the same data structure to provide the path from index j to the root on demand. This computation can be done in advance for all possible paths to improve time efficiency. However, this would come with some storage overheads. Such trade off between the computation on demand or the storage can be decided by enterprises according to their requirements.

4.4 Chapter Summary

In this chapter, we have proposed a two-level data deduplication framework that can be used in the cloud storage by enterprises who share a single common CSP for their services. By employing the cross-user data deduplication performed at the enterprise level and the cross-enterprise data deduplication performed at the CSP level, it is expected that enterprises can outsource their data to the cloud while the CSP can achieve cost and space savings. The framework is designed based on the constraint that the CSP is semi-honest, thereby cannot be trusted when handling users' data. Our scheme is designed to ensure privacy of the data under such constraint. We then propose POR and POW protocols for private data deduplication in which the CSP is considered to be semi-honest. These schemes are secure and lightweight in terms of the computational and storage overheads imposed on the enterprise since they are built on top of our deduplication framework and both complement each other to shape up a complete a model.

Chapter 5

Secure Image Deduplication through Image Compression

In this chapter, a novel compression scheme that achieves a secure deduplication of images in the cloud storages is proposed. Its design consists of embedding a partial encryption along with a unique image hashing into the SPIHT compression algorithm. The partial encryption scheme is meant to ensure the security of the proposed scheme against a semi honest CSP whereas the unique image hashing scheme is meant to enable a classification of the identical compressed and encrypted images so that deduplication can be performed on them, resulting to a secure deduplication strategy with no extra computational overhead incurred for image encryption, hashing and deduplication. Experimental results and security analysis are provided to validate the stated goals. In order to enhance the security of our scheme, we also propose POR and POW protocols for images based on the use of the SPIHT compression algorithm. The POR protocol can be run by the users to ensure that their images are stored securely in the cloud, and the POW protocol can be run by the CSP to authenticate the true owner of the images. The efficiency of our scheme relies on the fact

that only a fraction of the compressed data is used for encoding the images. Experimental results and security analysis are provided to validate the stated goals.

5.1 Background

This chapter focuses on image deduplication since images constitute a huge portion of the digital data of the users, which are expected to be replicated in a huge number [107]. On the other hand, from a storage saving perspective, image deduplication is a desired requirement for both the end users and the CSPs. A novel SPIHT based secure image deduplication scheme in the cloud is proposed, assuming that the CSP is semi-honest. The proposed scheme consists of a partial encryption scheme involving the SPIHT algorithm - which ensures the security against the semi honest CSP in the sense that the compressed image resulting from the SPIHT algorithm is no more available in plaintext to the CSP; and an image hashing scheme involving the SPIHT algorithm - which allows a classification of the identical compressed and encrypted images in such a way that deduplication can further be performed on these data. In order to augment the security of our proposed scheme, we also propose a POR scheme so that users can apply it to ensure the security and privacy of their images residing in the cloud storage. A POW scheme is also proposed so that the CSP can use it to verify that the images are released to their true owners. Since our target is the image data, the justification of the use of the SPIHT image compression algorithm [46] in our scheme relies on the fact that images are always compressed before being uploaded to any storage. To the best of our knowledge, the proposed scheme is the first attempt to explore image compression for the purpose of secured image data deduplication in the area of cloud storage services. Also this is the first time image compression for secured image storage and proof of storage protocols are explored in the area of cloud storage services. It is also

worth mentioning that in the proposed scheme, image compression is applied first, and then the resulting output is used for encryption and hashing purposes. This approach differs substantially from the previous works where image encryption or hashing is often applied first, and then the image is compressed for transmission purpose, which may lead to excessive computational overhead and more metadata to be stored.

The scheme proposed in this chapter is made of three components, namely, image compression, partial image encryption, and image hashing. For our scheme, we have used the SPHIT algorithm [48] for image compression, which is a variant of the EZW algorithm. The EZW algorithm [65] has the property that the bits in the bit stream are generated in some order of importance, yielding a fully embedded code. For partial encryption of the image, we need to utilize the data already produced during the compression process. [64], indicates some of the specific data as significant information which is very vital for decompressing the image. Hence by encrypting only this partial information, security of the compressed image can be made tighter. We therefore, use some of te basic concepts presented in [64]. In [60], Yang and Chen introduced an image hash scheme which is based on the knowledge of the locations of the significant wavelet coefficients of the SPIHT algorithm. In our work, we have adopted this scheme, by utilizing the wavelet coefficients of the SPIHT compression algorithm to generate the image hash. Further details of the background work can be found in Section 2.4.1. In order to address the issue of POR and POW for images, we use the SPIHT compression algorithm, a refined version of the EZW compression algorithm. In [87], error correcting code techniques were used to design a POR protocol. We have used a tailored version of this technique to implement POR and POW for images. A POW that uses the MHT approach was proposed in [84] and we apply the basic concepts of this scheme for our POR and POW schemes for images. Further details of the background work can be found in Section 2.4.2.

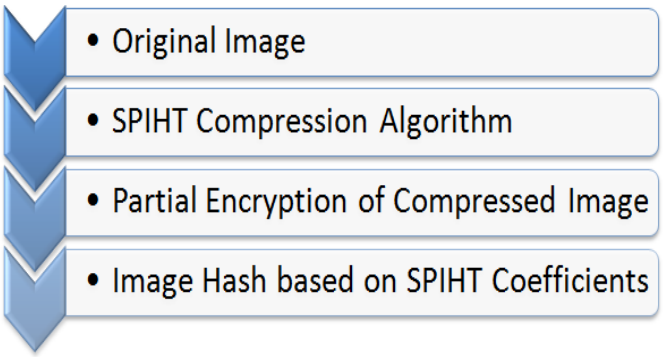

Figure 5.1: Image Deduplication Process

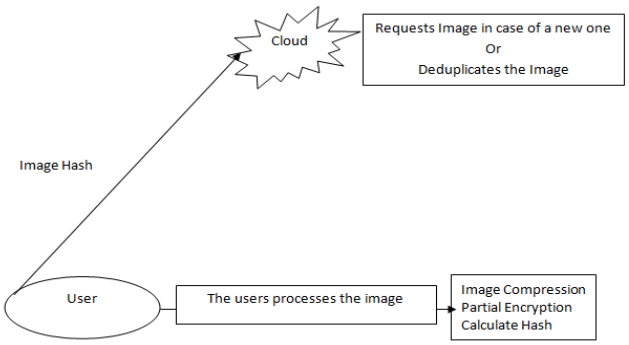

Figure 5.2: Flow of Information between the Users

The rest of the chapter is organized as follows: In Section 5.2, the proposed image deduplication scheme is described in detail along with some experimental results and security analysis. In Section 5.3, the details of the proposed POR and POW protocols are described and some experimental results are given to validate our POR and POW schemes. In Section 5.4, we conclude our work.

5.2 Proposed Image Deduplication Scheme

The design of the proposed approach for cross user client side secure deduplication of images in the cloud involves three components, namely, image compression scheme, partial encryption scheme, and hashing scheme as shown in Figure: 5.1. Figure: 5.2 illustrates the flow of

the data between the users in the proposed deduplication model. Typically, on the user's side, the user will process the image by applying image compression, partial encryption, and will calculate the hash signature of the image, in order. Next, he/she will only send the hash signature to the CSP. On the CSP side, the CSP will compare the received image hash against all the signatures already present in the cloud storage. If a match is not found, the CSP will instruct the user to upload the image. Otherwise, the CSP will update the image metadata and then will deduplicate the image by saving only a single unique copy.

5.2.1 Image Compression

We propose to utilize image compression in order to achieve image deduplication. The reason for doing so is that the images are compressed anyways for efficient storage and transmission purposes. In fact, applying the compression first and encrypting the compressed information next can help saving huge computational time and resources. As discussed above, the SPIHT algorithm [48] is used for image compression since it can produce an embedded bit stream from which the best images can be reconstructed [47]. In addition, it uses an embedded coding method which makes it a suitable scheme for progressive optimal transmission that produces fine compression ratios. It also uses the significant information sets to determine the tree structures, and its execution relies upon the structure of the zero trees of the EZW algorithm. Due to the fact that the SPIHT compression algorithm is independent of any keys or other form of user input, it is also suitable for deduplication since it meets the requirement that identical images compressed by different users should appear identical in the compressed or encrypted form to the CSP so that the similarity can be identified; and also, it does involve the use of chaos theory or random permutation from the users side.

The SPIHT algorithm is based on the fact that there is a correlation between the co-

efficients that are in different pyramids (bands) levels of the hierarchy of the underlying structure. It maintains this information in the zero trees by grouping insignificant coefficients together. Typically, each $2x2$ block of coefficients in the root level of this tree structure corresponds to further trees of coefficients. Basically, the SPIHT algorithm can be break into three phases, namely, initialization, sorting phase, and refinement phases.

- Initialization phase: Let *Coef(i,j)* denotes the coefficient at node *(i,j)*. For *Coef(i,j)*, three coefficient sets are further defined: *O(i,j)* the set of the coordinates of the children of node *(i,j)* in the case that this node has children; *D(i,j)* the set of coordinates of the descendants of the node *(i,j)*; and the set of coordinates of all coefficients in the root level. Then *L(i,j)* = *D(i,j)*− *O(i,j)* is the set of the descendants of the tree node, except its four direct offspring. Starting at the threshold T, any set of coefficients S is said to be significant (with respect to that threshold) if there is a coefficient in S that has a magnitude at least equal to T. Three lists are maintained by the algorithm, namely (i) *LIS* the list of insignificant sets - which contains the coordinates of the roots of insignificant sub trees - this list is further divided into two types of coefficients: *D(i,j)* and *L(i,j)*; (ii) *LIP* the list of insignificant pixels - which contains the coordinates of those coefficients which are insignificant with respect to the current threshold T - these are insignificant coefficients that are not part of any of the sets of coefficients in emphLIS; and (iii) *LSP* the list of significant pixels - which contains the coordinates of those coefficients which are significant with respect to the current threshold T.

  At the beginning of the algorithm, the threshold T is selected such that $T \leq \max_{(} i,j)|Coef(i,j)| < 2T$. The initial states of the lists are set in a specific manner, i.e. *LIP* maintains H the set of coordinates of the tree roots, *LIS* maintains the set *D(i,j)*, where *(i,j)* are coordinates with descendants in H, and *LSP)* is empty.

- Sorting phase: The algorithm identifies the significant coefficients by dividing the sets *D(i,j)* into two subsets, namely the set *L(i,j)* and the set *O(i,j)* of individual coefficients, and by putting each significant coefficient into the *LSP* list with respect to the current threshold level *T*.

- Refinement phase: This step consists of the refinement of all the significant coefficients of the list *LSP* from the previous pass in a binary search fashion. After each pass, the threshold is decreased by a factor of 2 and the whole process starts again. The algorithm ends when the desired bit rate is achieved.

A description of the SPIHT algorithm is available in [108].

5.2.2 Partial Encryption of the Compressed Image

We propose to partially encrypt the compressed image (obtained from the image compression step) before uploading it in the cloud storage. The reason for doing so is to ensure the security of the data from the CSP or any malicious user. Encrypting only a part of the compressed image will reduce the amount of data to be encrypted, thereby may considerably reduce the computational time and resources [64]. In order to satisfy the basic requirement of cross user deduplication, i.e. identical images compressed by different users should appear identical in the compressed/encrypted form, we propose to use convergent encryption [22] to encrypt the coefficients generated by the SPIHT algorithm since such scheme will allow the CSP to classify the identical compressed and encrypted images.

Typically, the output of the SPIHT compression algorithm is a stream of encoded wavelet coefficients along with the zero trees for the structure of the coefficients. It contains the sign bits, the refinement bits, the significance of the pixels, and the significance of the sets. Thus, to correctly decompress the image, the decompression algorithm must infer the significant

bits accurately.

In this regard, it was suggested in [64] that only the significant information be encrypted. This information is determined by the significant information of the pixels in the highest two levels of the pyramid as well as in the initial threshold for the SPIHT algorithm. As an example, if the root is of dimension of $8x8$, then the *(i,j)* coordinates will be encrypted if only and only if $0 \leq i, j < 16$, and there will be no need to encrypt the significant information belonging to subsequent levels, i.e. the pixels in the third, fourth, or sixth level of the pyramid. This is due to the fact that the coefficients in those levels will be reconstructed with the help of the information obtained from the coefficients belonging to the highest two levels of the pyramid.

Using the information obtained from the coefficients at the highest two levels of the pyramid, the states of the above-mentioned lists (i.e. *LIS*, *LIP*, *LSP*) will also be initialized. From these levels, the states of these lists will constantly be changed, depending upon the subsequent levels. But if the initial information is not correctly derived, these lists will be pruned to errors and the image will not be decompressed accurately. Hence, by having the user encrypt only the significant information of the highest two levels of the pyramid of the compressed image generated by the SPIHT algorithm before uploading it to the cloud storage, the image will be prevented from being decompressed by the CSP. The reason is that without the knowledge of the significant information from the highest two levels, the CSP will not be able to know the initial states of the above-mentioned lists, hence will not be able to decompress the image.

The convergent encryption algorithm [22] uses the content of the image to generate the key to be utilized by the users for encryption purpose, thus it is expected that two identical images will generate identical keys, thus identical cipher textt/cipher images. More precisely, the user constructs a SHA-3 hash of the significant information $Sig_$Info of the coefficients

belonging to the highest two levels of the pyramid and use it as the key for encryption, i.e.

$$Image\_Key(IK) = SHA - 3_{H}ash(Sig\_Info) \tag{5.1}$$

Using Equation: (5.1), identical encrypted significant information of the first two levels of the pyramid (hence identical cipher text) will be produced from two identical compressed images, irrespective of the fact that these images have been encrypted by different users. Using this key, the users will perform symmetric encryption on the significant information as shown in Equation: (5.2)

$$Sig\_Info' = (IK, Sig\_Info) \tag{5.2}$$

This way, the CSP will be able to determine the match between identical images form different users without having to process the images in plain text form.

5.2.3 Image Hashing from the Compressed Image

In previous steps, the image has been compressed (using the SPIHT algorithm) and only the significant information of the highest two levels of the pyramid has been encrypted. In order for the CSP to perform client side deduplication on the image, there has to be a unique identity (referred to as image hash) for each image. This step consists in generating such image hash. The image hash is generated by the user and sent to the CSP for performing the client side deduplication. In case of a redundant image, the CSP will only set the pointers of the image ownership to the new user and will not request the image to be sent again. Using this hash, the CSP will be able to identify and classify the identical images without the necessity to possess the original image in plain text format. Indeed, the CSP will need to scan through the already stored hashes in the cloud storage to find the matches rather than scanning the entire images. It should be noted that by doing so, the CSP will be able

to remove the redundant images and store only a single unique copy of the image. The CSP only has the knowledge of the compressed image and its partially encrypted version, and the original image cannot be reconstructed from these elements in any case. It is worth mentioning that the image hash generation will not involve any additional computational or storage overhead since the coefficients generated by the SPIHT algorithm are known. In addition, the image deduplication performed in this way will be secured against the CSP since all the significant information are encrypted.

Typically, the sorting pass of the SPIHT algorithm identifies the significant map, i.e. the locations of the significant coefficients with respect to a threshold. The binary hash sequence (where 1 denotes a significant coefficient and 0 denotes a non significant coefficient) is then designed based on this significance map and on the fact that the first four highest pyramid levels under the first three thresholds are more than enough to be used as the signature of the image [60]. In addition, the use of the convergent encryption is meant to ensure that the same cipher text is obtained for the same coefficients in the plain text format.

It is worth mentioning that our proposed scheme is expected to save a huge amount of bandwidth in practice since the user will not be sending the entire image to the CSP in case of image redundancy. This saving will also be cascaded in the cloud storage where only the unique copies of the images will be stored. In addition, the amount of the metadata to be maintained for deduplication per user is nominal compared to the amount of metadata that would have been generated and maintained if the entire images were sent to the CSP for reduplication.

5.2.4 Experimental Results

Experiments were conducted by keeping in mind the following three requirements: (1) using the SPIHT compression algorithm, the deduplication of the image should be performed accurately, i.e. should not be faulty and the proposed scheme should not identify two different images as identical, and even minor alterations should be trapped; a failure to this requirement will enable the CSP to eliminate one of the images and keep only one single copy of the two images which appear to be identical but are actually not; (2) the proposed scheme should be secure enough against the semi-honest CSP since the CSP is not given access to the compressed image and thus cannot decompress it; the CSP can identify the identical images from different users only through the significant maps of the image generated by the SPIHT algorithm;and (3) the proposed scheme should be efficient in terms of computation, complexity and storage overheads.

Experimental Settings

First, some altered images from one single image are generated. For this purpose, 6 classic images are considered: Lena, bridge, mandrill, goldhill, pepper, and clock, all represented in gray scale format. 10 different kinds of altered images are introduced for each single classic image and 10 versions of that particular classic image are generated, yielding a total of 126 images (i.e. 120 altered images added to the 6 original images). The first, second, and third altered images are obtained by changing the brightness, contrast, and gamma levels of the original image respectively. The fourth altered image is obtained by compressing the original image first (using a JPEG compression algorithm), then decompressing it. The fifth altered image is obtained by applying the rotations of the original image on the standard angular positions of 90, 180 and 270 degrees respectively. The sixth altered image is obtained by

applying a 2D median filtration on the original image. The seventh altered image is obtained by introducing a Poisson noise in the original image. The Poisson noise is a type of noise which is generated from the content of the data itself rather than by adding some artificial noise. The eighth altered image is constructed by rescaling the original image, i.e. by converting the unit 8 pixel values into double data type. Rescaling can also be done to convert the image pixels into an integer of 16 bits, single type or double type. The ninth altered image is obtained by applying a rectangular shear on both sides of the original image; and the tenth altered image is obtained by removing the noise from the original image using the Weiner filter.

Next, the Computer Vision Toolbox and Image Processing Toolbox of MATLAB are used for applying the above-mentioned alterations to the original images. For our experiments, we have also developed a MATLAB implementation of the SPIHT compression algorithm described in [48].

Deduplication Analysis

As stated earlier, one of the main requirements of the proposed scheme is that it should be robust enough to identify the minor changes between two images even if they are in the compressed form, i.e. it is desirable that the signature of the image generated from the SPIHT compression algorithm be different even when the original image has been slightly altered (hence, producing a different image). Thus, we have studied the influence on the image signature of the different alterations performed on the same image. Some graphical examples of the alterations on the pepper image can be noticed in Figure: 5.3. The results showing the percentage of dissimilarity between the signature of the original image and that of the altered image (obtained as described above) are captured in Table: 5.1 and Table: 5.2, for images data coded at a rate of 0.50 bpp and 0.80 bpp respectively, on two dimensions,

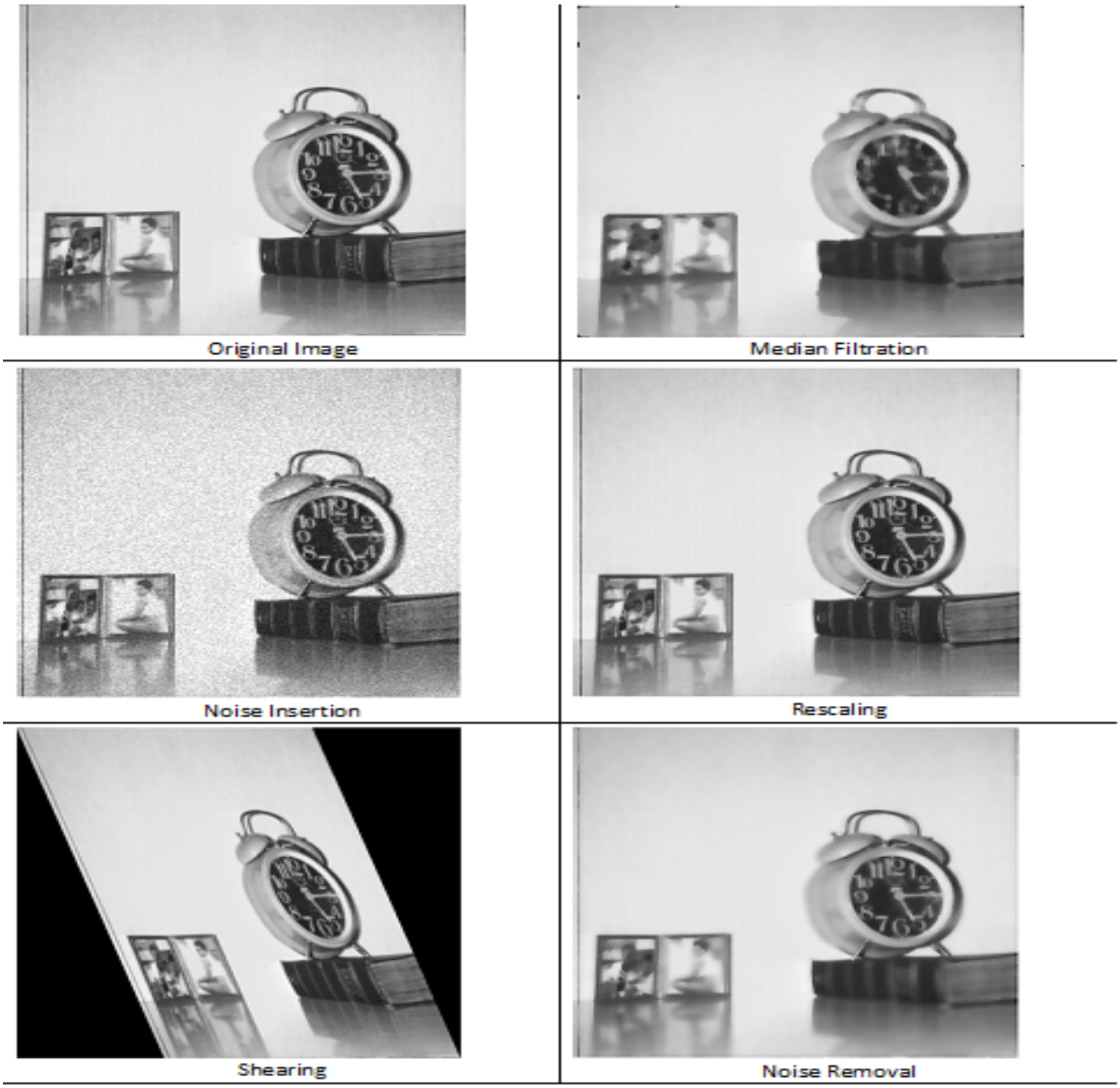

Figure 5.3: Alterations Performed on Pepper Image

Alteration	Clock (256x 256)	Clock (512x 512)	Pepper (256x 256)	Pepper (512x 512)	Lena (256x 256)	Lena (512x 512)
Brightness	6.8	5.7	6.4	4.8	5.3	4.2
Gamma	5.5	3.9	6.3	4.5	5.7	4.8
Contrast	6.3	6.2	6.0	4.9	6.7	4.6
Rotation	6.4	4.4	5.1	4.3	7.1	5.7
JPEG	0	0	0	0	0	0
Median Filtration	13.3	9.6	29.6	19.8	26.2	30.8
Noise Removal	11.4	8.2	30.5	24.66	26.3	30.1
Noise Insertion	29.1	24.2	28.8	24.18	27.1	30.9
Shearing	29.8	15.1	46.8	33.0	48.1	35.2
Rescaling	29.2	19.1	28.8	18.75	26.5	36.7

Table 5.1: % Difference between Significant Maps of the Original and Altered Images at 0.05bpp

namely 256×256 and 512×512. In Table: 5.1, it is observed that the difference between the signatures is lesser for the commonly used alterations (namely brightness, gamma correction, contrast and rotation changes) compared to the uncommon and severe alterations. These alterations are often performed by the users on the images but they do result in an entirely different appearance of the image. Although the images do not appear to change to a great extend for a human eye, the uncommon alterations (i.e. median filtration, noise insertion and removal, shearing and rescaling) are causing significant changes in the signature of the image. As shown in Table: 5.1, the percentage difference between the original and altered image

Alteration	Bridge (256x 256)	Bridge (512x 512)	Goldhill (256x 256)	Goldhill (512x 512)	Mandrill (256x 256)	Mandrill (512x 512)
Brightness	7.5	7.1	7.4	6.0	9.5	6.2
Gamma	8.2	7.8	9.15	6.7	10.4	6.3
Contrast	7.5	7.7	7.7	5.3	10.6	5.1
Rotation	7.12	6.42	6.9	6.25	8.0	6.2
JPEG	0	0	0	0	0	0
Median Filtration	19.6	7.5	28.8	19.0	20.2	11.5
Noise Removal	23.0	7.6	28.7	15.4	23.4	7.2
Noise Insertion	14.4	7.8	28.4	19.9	15.6	6.8
Shearing	29.8	17.3	30.8	21.3	30.1	18.5
Rescaling	14.4	7.8	27.5	19.7	15.5	7.1

Table 5.2: % Difference between Significant Maps of the Original and Altered Images at 0.08bpp

increases significantly for the severe alterations. As far as the dimensions are concerned, the percentage change between the signatures tends to decrease slightly as the image dimension increases. The same trend is maintained in the case the images data are coded at a rate of 0.80 bpp as shown in Table: 5.2. This shows that the signatures generated fulfill the deduplication requirements of our scheme. The altered images belonging to different users might appear to be the same but the alterations can be identified by the signatures and thus the images will not be deduplicated. The rest of the 63 images have been tested on 0.80 bpp to ensure that the images exhibit the same patterns when generating the signatures at different rates as well. As shown in Table: 5.2, the percentage of change for the commonly used alterations are less than that observed for the severe alterations, including at higher compression rates.

The alterations that have been introduced into the original images are also analyzed to determine how much these alterations affect the images' signatures. The results are captured in Figure: 5.4. In Figure: 5.4, it is observed that the shearing process has drastically changed the mathematical content of the image, producing a compressed image data significantly different from the original one. The same trend prevails in terms of difference between the compressed image data and the original data, when the other alteration techniques are utilized, namely noise insertion, rescaling, noise removal, and median filtration, in this order. It is observed that when alterations such as rotation, gamma, and contrast are used, their impact on the images' signatures are not that significant. In the case of JPEG compression, it is observed that the significant maps of the original images and that of the compressed images are similar. This is attributed to the fact that the significant maps of the original image are produced by a compression algorithm as well. Thus, in all the trials with different dimensions and compression rates, the percentage of change in the case of the JPEG compression is 0.

Angular rotations are among the most commonly performed alterations performed by

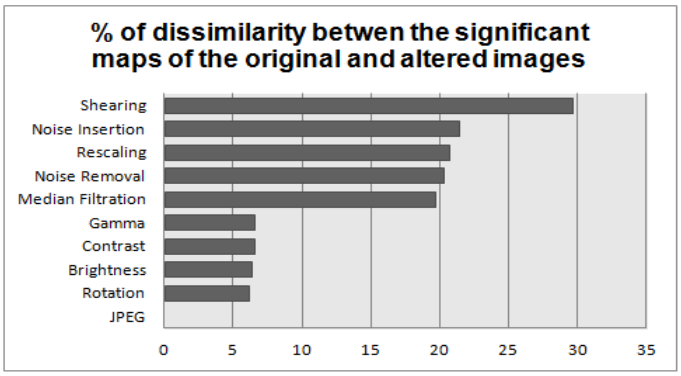

Figure 5.4: Percentage of Dissimilarity Caused by the Different Alterations.

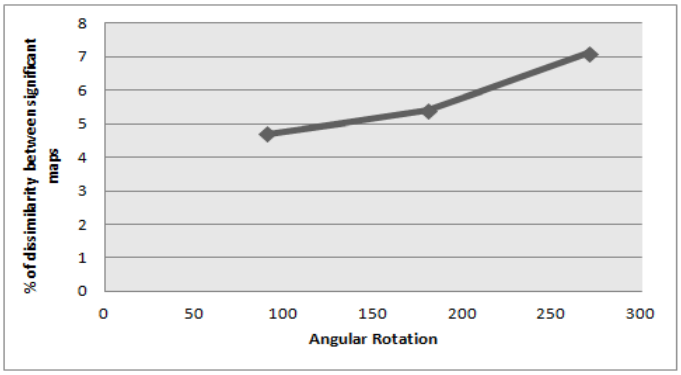

Figure 5.5: Percentage of Dissimilarity Caused by the Different Rotations.

the users on the images before uploading them in the cloud storage [60]. In this work, angular rotations are performed and their impact on the images' signatures are captured in Figure: 5.5. In Figure: 5.5, it can be observed that as the image moved away from its initial reference point, the images' signatures start to change progressively, then increases drastically, resulting to a percentage of dissimilarity between the signature of the original image and that of the altered image of about 4.6% (resp. 5.4% and 7.1%) when the angular position of 90 degrees (resp. 180 degrees and 270 degrees) are applied. This confirms a normal behavior of the signatures since the image itself changes in appearance as it is rotated.

Performance Analysis

The deduplication process of our proposed scheme also deserves some analysis to determine its performance in terms of computational overheads generated from the users' part. Due to the fact that the compressed images' data have been used to produce the unique hashes of the images for deduplication purpose, there is no other algorithm than the SPIHT compression algorithm that has been involved in the proposed scheme for generating the images' signatures. The SPIHT compression algorithm is invoked by the users to generate their compressed images before uploading them to the cloud storage. For doing so, the users needed to calculate the images' signatures from the lists *LIS*, *LIP*, and *LSP*) that have been generated during the compression process under the first three threshold levels. The performance of the deduplication process can be judged by quantifying the time taken for the signatures of the images generation versus the time taken for the generation of the above-mentioned lists.

Let T_1 be the time (measured in seconds) taken by the user to calculate the images' signatures from the lists; let T_2 (also measured in seconds) be the time taken by the SPIHT algorithm to generate the three lists under all the threshold levels, not including the time taken to perform the binary encoding and decoding steps.

In Table: 5.4 and Table: 5.3, T1 is the time consumed in calculating the signatures, T2 is the time taken by the compression algorithm to calculate the three lists under all the threshold levels and The third column shows how much percent of T2 is T1. T2 is not the total time taken by the compression algorithm since that includes some binary encoding and decoding as well. The Table: 5.4 and Table: 5.3 shows that the time taken to calculate the signatures is significantly low compared to that taken by the SPIHT algorithm to generate the data for the signatures. This was expected since the work carried out by the user on his/her own machine, i.e. performing the compression and extracting the signatures already

Alterations	Goldhill (256 x256)			Goldhill (512 x512)		
	T1 (sec)	T2 (sec)	T1% of T2	T1 (sec)	T2 (sec)	T1% of T2
Original	0.031	0.953	3.25	0.031	3.198	0.96
Brightness	0.031	0.952	3.25	0.093	3.494	2.66
Gamma	0.031	0.998	3.10	0.031	3.635	0.85
Contrast	0.031	0.936	3.31	0.031	3.588	0.86
JPEG	0.046	0.936	4.91	0.031	3.167	0.97
Rotation	0.015	0.952	1.57	0.031	3.182	0.97
Median Filtration	0.062	0.593	10.45	0.015	2.184	0.68
Noise Insertion	0.015	0.593	2.52	0.046	2.168	2.12
Noise Removal	0.015	0.53	2.83	0.015	2.163	0.69
Shearing	0.031	0.577	5.37	0.062	2.434	2.54
Rescaling	0.109	0.562	19.39	0.046	2.168	2.12

Table 5.3: Performance in Terms of Time for 0.50 bpp

Alterations	Pepper (256 x256)			Pepper (512 x512)		
	T1 (sec)	T2 (sec)	T1% of T2	T1 (sec)	T2 (sec)	T1% of T2
Original	0.094	0.58	16.21	0.035	2.137	1.64
Brightness	0.093	0.593	15.68	0.062	2.418	2.56
Gamma	0.046	0.608	7.57	0.031	2.496	1.24
Contrast	0.078	0.562	13.88	0.046	2.465	1.87
JPEG	0.062	0.546	11.36	0.031	2.122	1.46
Rotation	0.078	0.546	14.29	0.035	2.106	1.66
Median Filtration	0.015	0.39	3.85	0.015	1.544	0.97
Noise Insertion	0.031	0.421	7.36	0.078	1.763	4.42
Noise Removal	0.031	0.406	7.64	0.062	1.607	3.86
Shearing	0.031	0.53	5.85	0.036	1.934	1.86
Rescaling	0.031	0.421	7.36	0.031	1.591	1.95

Table 5.4: Performance in Terms of Time for 0.80 bpp

generated by the SPIHT algorithm, is expected to last for a relative small amount of time (or CPU cycles). Thereby, some storage space is saved by performing the image deduplication. At a compression rate of 0.50 bpp, it is also be noticed that for all the images of 256×256 dimensions (resp. 512×512 dimensions), T_2 is more than 10 times (resp. 90 times) the value of $T1$ in average. Therefore, as the dimension of the image increases, the time T_1 tends to decrease to further refine the the performance of the reduplication process. The same observations prevail when the images data are compressed at a rate of 0.80 bpp. In this case, T_2 is more than 10 times (resp. 56 times) the value of $T1$ in average for the images of 256×256 dimensions (resp. 512×512 dimensions).

The same patterns have been observed for all the 126 images, but reported the results for only 40 images in Table: 5.4 and Table: 5.3 (due to page limit restriction). As far as the compression efficiency is concerned, the SPIHT compression algorithm is not affected by the encryption and image hashing steps since these steps are performed only after the image compression has been completed. This way, the image deduplication is achieved with the help of the image compression.

5.2.5 Security Analysis

The setup of the proposed scheme allows the user to only upload the images data generated during the image compression step (it should be noted that a part of it is encrypted by the user) and the unique hash of the image calculated by the user. The CSP should not be able to decompress the image from the received compressed image data.

The security of the proposed scheme depends upon the partial encryption of the wavelet coefficients. The significant information in encrypted form is required in order to correctly decompress the image. The decoder of the SPIHT algorithm keeps track of the order of

execution of the encoder; the lists and image mask are both required for the most recent iteration of the encoder. From there on, the SPIHT algorithm starts to decode/decompress the image in the exactly reverse order of what the encoder has done until this process terminates.

In [64], it has been proved that the size of the important part is at least 160 bits if at least two sorting passes are performed by the EZT compression algorithm. Therefore, for an exhaustive search, an attacker would need at least 2^{160} tries before heading to a conclusion. For example, considering the Lena image with $512x512$ dimension, only the significant coefficients in the first two largest sub bands are required to be encrypted. These are the coefficients in the $16x16$ sub band with $8x8$ be the root level according [64]. We have calculated the size of the indicated data to be encrypted. The size of the two lists to be encrypted comes to 768 bytes in total. In addition, the mask of the image indicating if the pixels in the $16x16$ sub bands are significant or not also needs to be encrypted. The size of the mask is 575 bytes compared to that of the complete mask for $512x512$ pixels, which is 26000 bytes. Therefore, the partial encryption is indeed very efficient since the encrypted part is significantly less compared to the entire image data.

Moreover, as reported in [64], the order in which the coefficients are scanned in the refinement phase of the SPIHT compression algorithm depends upon the initial state of the three lists *LIS*, *LIP*, and *LSP* mentioned earlier. Since the initial states of these lists are determined from the coefficients in the highest two levels of the pyramid, which are encrypted, it is not possible to obtain the correct image without the knowledge of the significant information from these highest two levels. A many-to-many relationship exists between the important parts and the unimportant parts of the compressed data.Moreover, the important parts of two similar images will be drastically different, which makes difficult a possible attack from the CSP or a malicious user [64]. As far as convergent encryption is concerned, each user

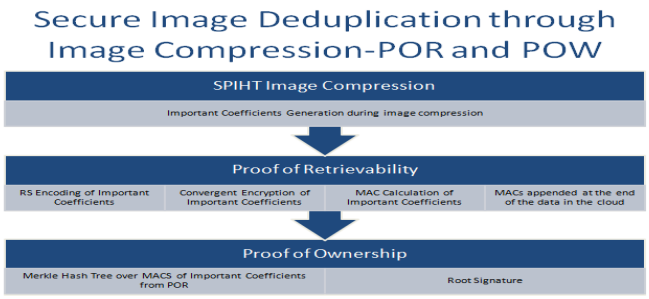

Figure 5.6: Proposed Image POR and POW Schemes

has his own key for the decryption of the image, which is kept secret from the CSP. Due to the deduplication requirements, the users cannot be allowed to choose different keys (using a public or private key model) for encryption purpose. The keys generated for two different users for the same image through convergent encryption are exactly the same. Hence, we assume that the users of the same cloud storage will not collude with the CSP, otherwise the security of the proposed scheme may be compromised.

5.3 Proposed POR and POW Schemes

Our scheme is made of three main components as depicted in Figure: 5.6, namely image compression, POR probed by the user to the cloud, and POW probed by the CSP to the user. It takes advantage of the use of error detection and correction potentials offered by the POR scheme to ensure the integrity of the data against the CSP and malicious users. The encoding/encryption parts of the POR protocol are meant to ensure that the CSP will not have access to the data in plain. The confidentiality of the data against unauthorized users is ensured through the use of the POW protocol.

5.3.1 Proof of Retrievability

Assuming that the file F to be processed contains the only important data generated from the SPIHT compression algorithm, the POR involves three operations to be performed on the data, namely, the use of an ECC - to protect the file from any modification, manipulation, and corruption; encryption, and sentinel insertion.

Encoding with an Error Correction Code

The file F is first divided into chunks of b blocks obtained by using the standard technique of striping. An ECC $C(n, k, d)$over $GF(21)$ is applied to each chunk, where n is the codeword length, k is the length of the message, and d is an even value parameter than can correct up to $d/2$ errors. It is assumed that b is a multiple of k and the file F can always be padded out for saving some storage space. Since the error correcting code increases the size of the encoded data, the new size of the chunk is n blocks so that the file is now represented as $\acute{F} = \acute{F}[1], \acute{F}[2], ...\acute{F}[\acute{b}]$ with $\acute{b} = bn/k$ blocks. The parameters for the ECC will be set by the CSP in order to maintain the similarity of identical images after encoding.

Encryption

The file $\acute{F}$ is now encrypted in such a way that it is protected from any attack from the semi-honest CSP or malicious users. Since the file constitutes the important information generated from the compression algorithm, the CSP should not have access to it in plain. This information is vital for a precise decompression of the image, therefore, it should be uploaded only in encrypted form. The Convergent encryption with 128 bit key length and 128 block length is used to encrypt all the blocks of the file $\acute{F}$ since it is assumed that the size of each block is 128 bit long,thus, the file $\acute{F}$ becomes $\hat{F} = \hat{F[1]}, \hat{F[2]}, ...\hat{F}[\acute{b}]$.

Message Authenticated Code (MAC) Generation

The MAC values of the encoded chunks are calculated as follows. The new file F (which has b blocks) is partitioned into segments of v blocks for some appropriate value of v. The positions of the blocks are being randomly generated by the PRF. These segments are now referred to as chunks. The owner of the file calculates the MAC value for each of these chunks denoted by $M = m_1, m_2.....m_q$ where m_1 is the aggregated MAC of the blocks for each chunk. The number of MAC values calculated is equal to the number of chunks that are generated. These MACs are appended at the end of the data to be uploaded in the cloud storage. The key k for the MAC is kept secret from the cloud storage provider and is only known to the user. The encrypted and encoded chunks are uploaded to the cloud along with the set M through the enterprise server. The user will keep the seed for generating the random index positions used by the PRF.

In order to verify the integrity of the data, the owner of the file will calculate the challenges as follows. A series of random data blocks positions in each chunk is calculated through PRF as follows: For instance, the user generates the indices with the seed of PRF and knowing v, divide the indices into the group of v chunks. The challenge is now of the form of a set of positions W_i and the position of MAC_i. The verifier probes the query by indicating the positions of the blocks to be checked and the position of the corresponding MAC. The prover returns the corresponding blocks and the corresponding MAC. The verifier calculates the MAC of the returned blocks and compare it against the returned MAC. If there is a matching, the blocks in question are kept intact; if not, the verifier can use the error correction code to recover the $d/2$ errors in the blocks. The basic requirements of the query and MAC calculation is that the user should be able to probe enough queries in a stretch, before downloading and recalculating the queries again. The number of queries with different

variables will be illustrated in the Experimental Results section.

5.3.2 Proof of Ownership

We now present our POW protocol, which is based on our POR scheme, and hence does not require any additional overheads. This POW will allow the CSP to ensure that the data is only released to the authorized user of the file who is indeed the owner of the file. In the previous steps of our scheme, we have already encoded, encrypted, and permuted the file F, which is composed of v chunks, each of which has b blocks. The owner of the file will then calculate a binary MHT as follows. Let MT_{bh} be a binary MHT over the M using b leaves and the hash function h. The leaves of the MT_{bh} are read from left to right and are the hash values of the elements in the set $M = m_1, m_2.....m_v$. Due to its binary nature, each node has two children. The leaf nodes L of MT_{bh} are $H(m_1), H(m_2).....H(m_v)$ and any non-leaf node NL of MT_{bh} is calculated as $NL_i = h(L_{ir}, L_{il})$ where L_{ir} and L_{il} are respectively the right and left child nodes of NL_i. Being a bottom up tree in structure, the root of MT_{bh} is determined by hashing up the values of the child nodes for each internal node. The user calculates MT_{bh} over the set M and signs the root $Rt(MT_{bh})$ of the encrypted tree. The user then sends the tree MT_{bh} and the set M to the CSP during the first upload of the file. The CSP will then calculate MT_{bh} over M and signs the root since this root is the same as the one that was sent by the user of the file. Now, to verify that the user is the true owner of the file, the CSP will run the POW protocol by randomly choosing j that belongs to M and sending the leaf index j to the user. The user will then be required to provide a sibling path from the selected leaf index j to the root. Using this path of internal nodes, the CSP is able to reconstruct the tree and verify its root against the one signed during the initial phase. In case the verification is positive, the user is accepted as the authorized owner of

the file, otherwise the CSP will not release the data to the user. Since the set M strictly requires that the user possess the entire file, as well as the encoding and compression data, it is reasonable to assume that only the owner of the file will have access to them. In this protocol, the CSP needs to store the set M and the root, and the owner needs to keep the set M as additional data at his/her side.

5.3.3 Experimental Results and Analysis

We have performed some simulations keeping in mind the requirements for efficient and secure POR and POW schemes. The security of our POR scheme is ensured firstly by encrypting the file F and number of queries that can be asked in one life session of the file. The permutation of the blocks in the subsequent step also ensures that their positions are not known to the adversary. Use of randomly positioned blocks within a chunk also forbids an adversary from eliminating the blocks without modifying any chunks. The fact that we are only encrypting and encoding the important parts of the video file and not the entire file also declines the volume of the data overall. The percentage of increase is almost the same in case of all the resolutions of the image because we are using the same standard RS encoding on all of the images, with the same parameters. The same error correction code C(n, k, d) over $GF(2^m)$ is applied to each chunk, where $m = 8$, $n = 256$, $k = 223$, and $d = 32$ (i.e. up to 2 errors can be corrected). It is also observed that the percentage of increase after the MACs are appended to the file is nominal in case of all the files and does not incur much overhead. It should be noted that the values for increase in size have been calculated by using 15 as the value of the blocks in each chunk.

Moreover, the ECC can tolerate a corruption of up to z-fraction of data blocks in F (z is dependent on which ECC is used) [87]. This feature implies that if the CSP deletes or

Lena Image Dimension	Original File Size (KB)	% Increase after Encoding	% Increase after MAC Insertion	No of Queries v=15	No of Queries v=10	No of Queries v=5
2048x2048	499776	12.5	0.829	1748	2622	5242
1024x1024	126976	12.7	0.828	437	656	1310
720x720	89653	12.5	0.826	437	656	1310
512x512	54496	12.5	0.824	218	328	656
256x256	7760	12.4	0.847	28	41	82
96x96	1547	12.4	0.870	7	11	21

Figure 5.7: Statistics for POR Scheme

modifies the z-fraction of F, the z-fraction of blocks will also changed with a high probability. Therefore, if the verifier probes enough blocks, he/she can detect the modifications performed on a z-fraction of F. Furthermore, the security of the POR scheme depends on the number of blocks per chunk. The chunks in challenges will tend to decrease over time since they are consumed by the runs of the POR protocol. Eventually, the verifier will have to recalculate the challenges for a particular file. We have simulated the number of challenges for different dimensions of the images using the Communication Systems Toolbox (for RS Codes). Using a *MATLAB* implementation of the SPIHT compression algorithm [48] and the *MATLAB* Image Processing Toolbox for processing the images, we have extracted the important part generated by the SPIHT compression algorithm for different images, and noticed that its size changes as the dimension of the image changes.

The statistics that were obtained for the classical Lena image with different dimensions are captured in Figure: 5.7. It can be observed that as the size of the file increases, the number of blocks per challenge does not affect the size of the file. When the number of blocks per challenge is increased, the security of the scheme is low in terms of queries. But, one has to take into consideration the fact that inserting too many blocks will also increase the overall size of the file due to encoding. It is also observed that when the number of blocks per chunk increases, the number of queries decreases. The tradeoff between security

and efficiency in terms of storage has to be taken into account. It can be observed that for smaller files, for example, that of size *512 x 512*, the percentage of increase in size started to raise unless we decrease the number of blocks inserted as in the case of the image of size *96 x 96*. Doing so will reduce the number of challenges that the verifier can ask for the file in one cycle.

When all the queries are exhausted, the verifier has to recalculate the values and start the cycle again. In the case where $q = 1748$, a verifier can input a single challenge to the prover every day for up four years. On the other hand, for smaller files, the number of challenges is *82* and only enough for short period of time . In case of sensitive files with smaller size, the user can use more blocks at the cost of computational overhead. As pointed out in [87], the probability of detecting a file being corrupted by a semi-honest CSP for z is $1 - (1 - z/4)^q$. According to this, the probability for larger files used in our simulation is almost 71.4% which is acceptable since the detection of the file corruption through error correcting codes is a cumulative process [87]. For smaller files with reduced number of blocks, the percentage goes down to 11.9% which can be rectified by increasing the number of blocks. Our POW protocol is efficient in terms of computations in the sense that it does not require and calculation of the data separately, but uses the data already generated for the POR scheme. For the data files like 256 x 256 with number of chunks 32 in the Lena image, the height of the tree is $O(\log n)$ where $n = 32$. The time complexity to built the MHT is also proportional to $O(\log n)$, and this reflects the fact the operation of insertion, space, and time required for carrying out the POW scheme using the MHT approach does not incur any computational overhead due to the small size of the important part of the compression data. The security of our POW scheme is therefore strong enough due to the use of hashes in its underlying data structure of the MHT.

5.4 Chapter Summary

We have presented a novel secure image deduplication scheme through image compression for cloud storage services purpose. The proposed scheme is composed of three parts, namely: SPIHT compression algorithm, partial encryption, and hashing. The compressed image obtained form the SPIHT algorithm is partially encrypted in such a way that it is not available in plaintext form to the semi-honest CSP or any malicious user, hence ensuring the security of the data from the CSP. The image hashing technique allows a classification of the identical compressed and encrypted images based on their short signatures, in such a way that the image deduplication step is carried out efficiently. We have also proposed a scheme made of a POR protocol and a POW protocol for secure image storage in the cloud based on the SPIHT image compression. Since the data from the deduplication and compression process is used to run the security protocols, our scheme can save huge computational overheads. Experimental results have shown that (1) the proposed scheme is robust enough to identify minor changes between two images even if they are in the compressed form; (2) the proposed scheme is secured against the semi-honest CSP since the CSP does not have access to the compressed images, but can identify the identical images from different users only through the significant maps of these compressed images; (3) the proposed POR and POW schemes are efficient.

Chapter 6

Secure Video Deduplication Scheme in Cloud Storage Environment using H.264 Compression

In this chapter, a secure scheme is proposed that achieves cross user client side video deduplication in cloud storage environments. Its design consists of embedding a partial convergent encryption along with a unique signature generation scheme into a H.264 video compression scheme. The partial convergent encryption scheme is meant to ensure that the proposed scheme is secured against a semi-honest CSP; the unique signature generation scheme is meant to enable a classification of the encrypted compressed video data in such a way that the deduplication can be efficiently performed on them. We also introduce POR and POW protocols for video deduplication in cloud storage environments. The POW protocol is meant to be used by the CSP to authenticate the true owner of the data video before releasing it whereas the POR protocol is meant to allow the user to check that his/her data video stored in the cloud is secured against any malicious user or the semi-honest CSP. Experimental

results are provided, showing the effectiveness and security of our proposed schemes.

6.1 Background

A secure video deduplication scheme in cloud storage environments using the H.264 compression algorithm is proposed assuming that the CSP is semi-honest. The design of the proposed scheme consists of three modules. First, a H.264 compression algorithm is applied on the given video by the user. Second, a unique signature of the compressed video is generated by the user in such a way that the CSP can use it to compare the user's video against other videos without compromising the security of the user's video; i.e. the CSP will not be able to have access to the video content itself, but will be able to deduplicate the video using the signatures generated by the users. Third, a partial encryption is applied by the user on the compressed video so that the CSP or any other malicious user will not be able to access the uploaded video in the cloud storage. Typically, the partial convergent encryption scheme is meant to ensure that the proposed scheme is secured against the semi-honest CSP; the unique signature generation scheme is meant to allow for a classification of the encrypted compressed video data in such a way that the deduplication process is further efficiently performed on them. The proposed POR scheme is meant to ensure that the data is not only safe from the malicious users, but also from the semi-honest CSP. It should be stressed that providing security from the CSP is a thought challenge since he/she is an insider and should be given a fair access to the video data in order to store it. The proposed POW scheme is meant to be used by the CSP to authenticate the true owner of the data video before releasing it. It should also be noted that the information used in the POW scheme is the one already generated for the POR scheme.

H.264 is an object-based algorithm that makes use of local processing power to recreate sounds and images [66]. The H.264 algorithm [67] is a block-based motion compensation codec most widely used for HD videos. Its working is based on frames, which can be further grouped into GOP and thus can yield high deduplication ratios for your scheme. A selective encryption of the H.264 compressed video is proposed in [77] and it works on the GOP level and on various types of videos. These characteristics makes this selective encryption scheme very suitable for our video deduplication scheme. The signature generation is inspired by the method proposed in [109], but this method is modified to fit our requirements. In [109], the scheme proposed is used for watermarking which is not useful for deduplication, therefore, it is used only for signature extraction purposes in our scheme. Our scheme for secure POR and POW protocols for videos is basically built upon the Secure Video Deduplication Scheme in Cloud Storage [14]. For the POR protocol, we use the underlying concepts of ECC described in [87] and modify it further to fit into the requirements of deduplication. In order to run the POW protocol, we use the basic technique described in [84] and tailor it to fit into our framework for deduplication. Further details of the background work can be found in Section 2.7.

The rest of the chapter is organized as follows: In Section 6.2 the proposed scheme is described in detail along with it's experimental results and security analysis. In Section 6.3, the proposed POR and POW protocols are described, and some experimental results are provided to demonstrate their effectiveness. Section 6.4 concludes the chapter.

6.2 Proposed Video Deduplication Scheme

The design of the proposed approach for secure deduplication of videos in the cloud storage involves three components: H.264 video compression scheme, signature generation from the

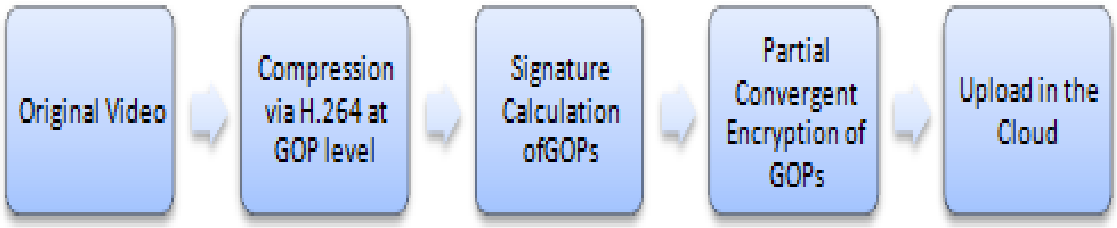

Figure 6.1: Proposed Secure Video Deduplication Scheme

compressed videos, and selective encryption of the compressed videos as shown in Figure: 6.1.

First, the user compresses the original video using the H.264 compression algorithm. Second, he/she calculates the signatures based on the GOP from the output bit stream. Third, he/she encrypt the important parts of the DCT coefficients and motion vectors according to the type of the videos. After these processing steps on the original video, it will be uploaded in the cloud storage. The CSP will then check for the identical GOPS with the help of the signatures and encrypted data. If identical GOPs are detected, the CSP will delete the new data and update the metadata for this particular data already in the cloud storage. In this way, the CSP will save huge space by performing cross-user video deduplication in the cloud storage. The detailed description of the proposed scheme follows.

6.2.1 H.264 Video Compression

The first part of our video deduplication scheme is to compress the original video sequence. The compression algorithm should be strong and efficient enough to produce highly compressed, retrievable and smaller in size version of the video. We use the H.264 compression algorithm [66]. This algorithm includes a prediction step, a transform step, and an entropy coding step.

- The prediction step: The coded video sequence generated by the H.264 algorithm is made of a sequence of coded pictures. Each picture is divided into fixed size macro blocks. It should be noted that macro blocks are the basic building blocks in the H.264 algorithm. Each macro block is a rectangular picture area made of 16×16 samples for the luma component and the associated 8×8 sample regions for the two chroma components. The luma and chroma samples of a macro block are used for prediction purpose. They are either spatially predicted or temporally predicted by the algorithm and the resulting residual is divided into blocks. These blocks are then transformed, quantized and encoded in the final stages.

  The macro blocks of the picture are further grouped into slices, which themselves divide the above regions of the picture that are eligible to be decoded independently. Each slice is a sequence of macro blocks. It is processed following a raster scan, i.e. from top-left to bottom-right. The slices can be used for error resilience since the partitioning of the picture allows a spatial concealment within the picture and the start of each slice provides a resynchronization point at which the decoding process can be reinitialized [66]. The slices can also be used for parallel processing since each frame can be encoded and decoded independently of the other frames of the picture. The robustness of the algorithm can be further strengthen by separating the more important data (such as macro block types and motion vector values) from the less important ones (such as inter residual transform coefficient values).

  There are five fundamental frames types, namely (1) the I slice - i.e. a frame in which all macro blocks are coded using the intra prediction. It is the most important type of frame used by all versions of the algorithm; (2) the P frame - i.e. a frame in which the macro blocks can also be coded using the inter prediction with at most one MCP

signal per block; (3) the B frame - i.e. a frame in which the macro blocks can also be coded using the inter prediction, where the signals per prediction block are combined using a weighted average; (4) the SP frame; and (5) the SI frame types.

It should be noted that the intra-picture prediction can be performed in two types of intra coding modes: the intra 4×4 and the intra 16×16 modes. In the intra 4×4 mode, each 4×4 luma block is predicted from spatially neighboring samples. In the intra 16×16 mode, the whole 16×16 luma component of the macro block is predicted with four prediction modes, namely, vertical, horizontal, DC, and plane.

The inter-picture prediction in P frames is another prediction method that can be utilized by the H.264 algorithm. It involves the use of P macro blocks with luma block sizes of 16×16, 16×8, 8×16, and 8×8 samples. For each 8×8 partition, this method can be used to decide on whether or not that partition should be further partitioned into smaller sizes of 8×4, 4×8, or 4×4 luma samples and corresponding chroma samples.

- The transform step: This includes the transform, scaling, and quantization sub-steps. In the H.264 algorithm, an integer transform such as the 4×4 DCT is applied to 4×4 blocks instead of larger 8×8 blocks as used in previous standard methods. The inverse transform of H.264 uses simple exact integer operations so that mismatches are avoided and the decoding complexity is minimized. For the luma component in the intra 16×16 mode and the chroma components in all intra macro blocks, the DC coefficients of the 4×4 transform blocks undergo a second transform, so that the lowest-frequency transform basis functions cover the entire macro block [66]. A quantization parameter (QP) is then used for determining the quantization of the transform coefficients. The quantization step size is controlled logarithmically by this

QP in such a way that the decoding complexity is reduced and the bit rate control capability is enhanced [110].

- The entropy coding: In the H.264 algorithm, two entropy coding are supported: the CAVLC and the CABAC. The CABAC has better coding efficiency compared to the CAVLC, but yields a higher complexity. In both of these modes, the syntax elements are coded using a single infinite-extent codeword set (referred to as Exp-Golomb code).

In our proposed video deduplication scheme, the videos are divided into GOP based on similarity; where each GOP is made of I frames and P frames. The H.264 algorithm is used with the following specifications: the GOP size is set to 15 frames, where the first frame is always the I frame used as reference for the subsequent 14 frames, which are all P frames.

6.2.2 Signature Generation from the Compressed Videos

The generated signature captures the content dependent robust bits from the macro blocks generated by the H.264 Compression algorithm. These bits are then used to authenticate or identify one GOP from another, hence, they are treated as the signature of the GOP. The signature calculation is done in parallel with the compression algorithm as the GOPs are being generated by the H.264 algorithm.

There are two broad strategies of watermarking in case of digital videos [109]: digital watermarking and digital signature. The former refers to the process of embedding some information into the video. This information could be generated by the user, which implies that some important variables could be provided by the user in the process, resulting to the side effect that the same video originated from different users may look different at the end of the signature generation, which is not a requirement for cross-user deduplication. For this reason, the technique of digital signature is utilized in this work for the authentication

of the video. In our case, it consists of embedding a SEI in the H.264 bit stream (as suggested in [109]), but an issue with using this technique is that it would require some extra bandwidth for the video to be transmitted since it increases the bits transmitted by the encoder. However, as we are dealing with cloud computing environments, a slight increase in bandwidth can be neglected since the videos uploaded by the users in the cloud are typically of large size.

The signature generation is carried out in the compressed domain, and the signatures are generated from the information produced in the transform domain of the H.264 compression algorithm. The content dependent robust bits are extracted from the macro blocks and are further used as the signature for authenticating the compressed video. It should be noted that in our use of the H.264 algorithm, the I and P frames are the minimum types of the frames that should be present for the algorithm to work.

Indeed, the video is first broken down into GOPs, which are authenticated individually by hashing the features extracted from their I frames and P frames. The hash is then considered as the digest digital signature for all the frames in the GOP. The digital signature is composed of the features extracted from the intra 16×16, intra 4×4 and inter 4×4 MBs. The I slices are composed of intra coded MBs in which each intra 16×16 and intra 4×4 luma regions are predicted from the previous 16×16 and 4×4 coded MBs of the same I frame.

For the intra 4×4 and inter 4×4, the quantized DC coefficients and the first two quantized AC coefficients in the zig zag scan order (surrounding the only DC coefficient value) for every 4×4 MB are pull out as the signature data. Then, for every intra 16×16, all the nonzero quantized *Hadamard* transform coefficients and the first two quantized AC coefficients in the zig zag scan order (also surrounding the only DC coefficient value) for every 16×16 MB are pull out as the signature data. The signature is calculated for each MB in a frame until the end of the GOP is reached. Meanwhile, these signatures are saved in a buffer and when

the algorithm signals IDR, the signatures for all the MBs for all the frames in the GOP are hashed by using *SHA2-256*, producing a 256 bit long signature for each GOP. Finally, the signatures for all the GOPs are calculated and transmitted along with the video. The CSP will compare these signatures against the ones already stored in the cloud storage in order to identify any possible duplicated parts of the videos. The deduplication at the GOP level will further increase the deduplication ratios since it is less likely that identical entire videos by different users will be uploaded in the cloud as opposed to parts of the videos.

6.2.3 Selective Encryption of the Compressed Videos

Assuming that the compressed videos have been produced by the H.264 algorithm, and the signatures have been generated by the users, the next step consists of encrypting the compressed videos so that it will not be possible for the CSP to access the plain videos. The partial encryption is known to be weaker than the full encryption [64], but it offers a fair tradeoff between security and efficiency, which are vital for cloud storage environments with large number of users.

We have used the partial encryption scheme proposed in [77], with some modifications applied on it in order to fulfill the requirements of cross-user deduplication. The partial encryption is carried out in the compressed domain and therefore is well in line with the signature generation process since that is also performed in the compressed domain. Typically, since the video is split into GOPs, the encryption is performed at the GOP level. This encryption scheme is content-based since user dependent variables are not involved in its process. Content-based encryption ensures that the same encrypted videos will be obtained for the same plain videos by different users, which is the basic requirement for cross-user deduplication.

The encryption process starts by first generating the compressed bit stream for the video. First, the user classifies the video into six different categories, namely high, medium and low intensity motion (for complex texture) and high, medium and low intensity motion (for non-complex texture) by utilizing the information generated in the intra prediction mode, DCT coefficients, and motion vectors (as described earlier). It is assumed that if the I frame of a GOP is complex in texture, the corresponding P frames will also be complex in texture. For the motion intensity, the motion vectors of the P frames are taken into account. Second, the user organizes the GOPs of the above mentioned six classes. Basically, the user calculates the average of nonzero coefficients and the average number of intra 4×4 prediction for the I frames. For the P frames, the average of the suffix length of the MV keywords and the corresponding variance are calculated. These values are then compared against the threshold values given by the CSP to all the users of that cloud. The users will use these threshold values to classify their videos and start the encryption process. The details of the classification of the videos are beyond the scope of our current research, but can be found in [77]. The partial encryption then follows after the video classification step.

The compressed video has already been broken into GOPs for the purpose of signature generation. This same set of GOPs is used for partial encryption as follows. The DCT coefficients and intra prediction modes in every GOP are encrypted according to the texture of the video. In doing so, the I frames are considered as crucial for the decoding of the P frames in the sense if errors occurred in the I frames or if these frames are tampered (even slightly), the decoding of the P frame will be affected for any GOP. Therefore, all the intra prediction modes are encrypted, no matter what the texture or motion intensity of the video is. For complex texture videos, all the nonzero DCT coefficients, except for the trailing ones are encrypted because of the fact that the videos will have a large number of high band DCT coefficients. For non-complex texture videos, only the first three low band DCT

coefficients are encrypted. For motion intensity purpose, the user will encrypt the motion vectors of the P frames. In case of the complex texture video, the user will encrypt the motion vector difference in all the P frames, i.e. the first 70% and then the first 30% of the P frames respectively for high, medium and low intensity videos for each GOP. In case of the non-complex texture video, the user will encrypt the motion vector difference in all the P frames, i.e. the first 70% and then the first 30% of the P frames respectively for high, medium and low intensity videos for each GOP.

In our scheme, the convergent encryption [19] is employed to derive the key for partial encryption from the content of the compressed video rather than getting the key chosen by the users individually. Therefore, for the same content, the same key will be generated without the users knowing each other. Thus, different users will have the same key as well as the same encrypted videos, irrespective of the knowledge of each other keys. This will make it easier for the CSP to compare the encrypted parts of the videos and perform deduplication in case of duplicated videos without actually decrypting or decompressing the video data.

The convergent encryption steps are described as follows. Let us assume that user partially encrypts the video data as described earlier and the video is complex textured, with medium motion intensity. Then three data sets are to be encrypted, namely, $INTRA-PRED = (\textit{All the intra prediction modes in a GOP})$,
$NZ-DCT = (\textit{All non zero DCT coefficients in a GOP})$ and
$PER-MVD = (\textit{\% of the MVDs in a GOP})$. The user will calculate the keys $K1$, $K2$, and $K3$ as follows: $K1 = \textit{SHA2 (INTRA-PRED)}$, $K2 = \textit{SHA2 (NZ-DCT)}$ and $K3 = \textit{SHA2 (PER-MVD)}$; then will perform the following encryption: $Enc(INTRA-PRED) = AES(K1, INTRA{-}PRED)$, $Enc(NZ_DCT) = AES(K2, NZ_DCT)$ and $Enc(PER{-}MVD) = AES(K3, PER-MVD)$. In this way, for every vector and for every MB of any two identical GOPs, the encrypted data will appear to be similar. The CSP will then be able to

compare the GOPs without breaching the privacy and security of the uploaded data while the storage space will be maximized through cross-user deduplication. Once the video and the signatures are uploaded by the user in the cloud, the CSP will compare the signatures of the compressed video for deduplication purpose. If the signature matches, the new copy is discarded and the metadata is updated. Depending upon the resources and security requirements, the CSP can compare the encrypted part of the videos after the signature matches, in order to ensure that the two GOPs are identical before deleting the data. Since convergent encryption is applied, the same plain videos would result is the same cipher videos, thus enabling cross-user deduplication in the presence of a semi-honest CSP.

6.2.4 Experiments Results

The experiments have been conducted by keeping in mind the following requirements: (i) some digital space must be saved by applying the H.264 compression algorithm and deduplication; (ii) the compression algorithm to be efficient in terms of computation, complexity and storage overheads; (iii) the signature generation step must be robust enough to identify the GOPs for deduplication; and (iv) the attacks from the external users are much more straightforward than the ones from the internal users. Therefore, our goal is to show that our proposed scheme is secure against the semi-honest CSP. The use of partial encryption and signature generation done at the user end is therefore analyzed thoroughly for security and efficiency purposes. In order to analyze security and efficiency of our proposed scheme, we tested it on 6 different video sequences, namely Akiyo, Foreman, Claire, Grandma, and Highway. The specifications of all these video sequences are are well known and are presented in Table: 6.1. We have chosen these videos for our testing because they belong to different classes of videos [111]. Foreman and highway are classified as non-complex textured and

Video Sequence	Size of Video (MB)	Total GOPs	Avg Size of GOP (KB)
Akiyo	1.17	20	53.33
Grandma	3.07	58	54.9
Mobile	2.80	20	154.66
Foreman	0.944	14	60.74
Claire	1.33	33	42.10
Highway	6.14	134	49.45

Table 6.1: Video Sequences Information

medium intensity videos. Grandma, Akiyo and Claire are classified as non-complex textured and low intensity motion videos. Each video sequence is first encoded into the QCIF, and then the frames are extracted. The videos are then input into the H.264 compression algorithm with the GOP size of 15. The first frame is the I frame and the rest of the frames are P frames, which implies a IPPP format. The QP value is set to 28. The algorithm works on 4×4, 16×16 MB modes for I and P frames. The rate of compression for H.264 is set to 30 Hz. The details of the algorithm can be found at [67]. The system configuration for the experiment is an Intel Core i3 processor with the 2.40 GHz frequency and 4 GB RAM. The underlying operating system is operating on 64 bits. We implemented our scheme in MATLAB R2014a version. Finally, different methods, all implemented in MATLAB, are used for the extraction and processing of QCIF format videos, as well as the extraction of the frames.

The video deduplication is the most important requirement which we need to fulfill. The deduplication aspect of our scheme is captured in Table: 6.2. We have calculated the amount of space saved in the cloud storage in the case that the CSP practices cross-user

Video Sequence	% of Space Saved for 25% of GOPs	% of Space Saved for 50% of GOPs	% of Space Saved for 75% of GOPs	% of Space Saved for 100% of GOPs
Akiyo	26.65	53.33	79.99	100
Grandma	25.92	51.85	77.78	100
Mobile	27.61	55.23	82.85	100
Foreman	22.5	45.02	67.53	100
Claire	27.06	51.12	75.17	100
Highway	26.98	53.96	80.94	100

Table 6.2: Deduplication Performance of the Video Sequences

deduplication at the GOPs level for 25%, 50% , 75% and 100% of GOPs in our experiments. From the results shown in Table: 6.2, It can be observed that for complex textured videos such as mobile, the percentage of digital space saved is higher than that of the non-complex textured video. This is attributed to the fact that these videos are larger in size (i.e. there are more information for complex textured videos) on the disk as compared to the others. As more and more GOPs are replicated, the space savings increase at the cloud storage end. The CSP will simply need to update some metadata for the GOP, indicating that this particular GOP had also belong to this user in addition to some other users as well. The size of the metadata is very nominal as compared to the space saved by simply storing a single copy of the GOP rather than a copy for each user. Moreover, for CSPs with large storages, the metadata space is negligible because of its textual nature. For videos that are large in size and high motion intensity, it can be noticed that there are substantial space savings as in case of the Highway video. The percentage of space saved increases as more and more GOPs are identical. In case the entire video is the same, 100% of the space is

Video Sequence	Avg time to encode(sec)	Avg PNSR	Avg time to calculate hash of sig (sec)	Avg length of sig (Bytes)
Akiyo	146.685	39.9	0.0524	1536480
Grandma	173.846	39.3	0.0480	1568745
Mobile	220.54	35.21	0.0632	1743569
Foreman	160.66	37.7	0.0832	1812960
Claire	166.38	41.5	0.068	1463256
Highway	165.65	39.4	0.0603	1722373

Table 6.3: Compression Performance of the Video Sequences

saved. It can also be noticed that the amount of space saved also increases as the size of the video increases, which is very favorable for a cloud computing setup since the users try to upload large videos at times. The videos uploaded by different users in the cloud are often not exactly identical, but at times cropped from the beginning or in the end. Unlike images, videos are also not often enough modified in terms of brightness and color by the users, but usually can be changed in length by simply cropping some parts of them. This explains why our scheme is based on the GOPs so that the deduplication ratios can be improved as much as possible.

The second requirement of our scheme is that the inclusion of the deduplication process should not incur any extra overhead on the process of uploading the videos in the cloud storage from the user's end, i.e. the proposed scheme should be computationally cost-effective in terms of resources at the user's end. We are proposing to use the H.264 compression algorithm as the basis of all our further computations. Since videos are compressed anyways before being uploaded in the cloud, the computational overhead in reduced to a great extend.

The user is calculating the signatures from the information produced from the compression algorithm. The encryption is also carried out on the information generated by the compression algorithm. Therefore, the performance efficiency should be determined in terms of how much time the user spent on these two actions. From Table: 6.3, it can be observed that the average time to encode the videos is much higher than that to calculate the signature. In case of the Highway video sequence, the average time to compress the video is 165.65 seconds and the average time to calculate the signature at the GOP level is 0.0603 seconds, which is nominal compared to the time to encode. It can also be noticed that as the size of the video increases (from foreman to highway), the number of GOPs naturally increases, but the time taken to encode and the PNSR also depends on the nature of the video. The mobile video sequence is complex in texture and smaller in size, but took the highest time to encode. The time taken (and consequently the number of CPU cycles used) to compare the signatures is also insignificant in the case of large cloud storages since the size of the signature is merely 256 bits. For instance, as can be seen from Table: 6.3, the size of the actual signature for the grandma video is 1568745 bytes, which has been reduced to 256 bits by the SHA hash. These signatures will be transmitted along with the compressed videos, but because of their size, they do not incur any overhead in terms of bandwidth consumption.

For the grandma video, a total of 58 signatures need to be transmitted with an approximate size of 14848 bits, which again is negligible for the user. The signatures are also calculated at the GOPs level, i.e. for each GOP, a 256 bits signature is generated, which is very small compared to the original size of the GOP. Depending upon the computational capacity of the cloud and security and efficiency tradeoff, the signatures can be calculated for each frame rather than at the GOP level, and each frame signature can be checked for deduplication. The time taken to calculate those signatures is also negligible, even at the user's end. The highest amount of time taken is consumed in the compression step; and the

other times are negligible in that context.

For the CSP, the benefit comes when he/she has to compare the signatures (256 bits) rather than the GOPs themselves (e.g. 53.33 KB). The size of the video to be stored also gets reduced after compression, which is also a benefit for the CSP in terms of storage savings. The proposed scheme is designed in such a way that the performance of the H.264 compression algorithm is not affected by the signature calculation and the partial encryption because these methods are applied on top of the information generated by the H.264 compression algorithm. The performance of the H.264 compression algorithm is shown in Table: 6.3 in terms of PSNR. The performance of our proposed scheme can be judged from the results depicted in Table: 6.2 and Table: 6.3.

We have also quantified the time consumed in the partial encryption of each GOP. In our proposed video deduplication scheme, the partial encryption presented in [77] is adopted and adjusted to meet the requirements of cross-user deduplication. Therefore, we have considered the same experimental settings used in [77], in terms of number of frames in GOPs, compression rate, and QP factor. We conducted the experiment on a few samples and obtained almost similar results as the ones shown in [77] in terms of time consumed for partial encryption. Indeed, the Foreman video took 0.44019 seconds, the Highway video took 2.26 seconds, the Akiyo video took 0.354 seconds and the mobile video took 1.308 seconds. From these results, it is clear that the time taken to perform partial encryption increases as the size of the video increases (the Highway video being the largest of all) and this time tends to increase for videos with complex texture since they have more information to encrypt. Overall, the time taken to encrypt the partial compressed information is negligible compared to the time taken to encode the video. Moreover, considering the tradeoff between security and efficiency, this amount of time can be overlooked when it comes to cloud computing.

6.2.5 Security Analysis

The setup of the proposed scheme allows the user to only upload the video data generated during the H.264 compression step, and at the same time, to partially encrypt the video and generate the unique hash of each GOP. The CSP should not be able to decompress the video from the received compressed data, which is partially encrypted. The security of the proposed scheme depends upon the partial encryption of the DCT coefficients and motion vectors. We have used the convergent encryption, which utilizes the 256 bits hash of the data as the key for the AES encryption. The possibility of AES encryption being compromised is very seldom since it requires a computational complexity of $2^{126.1}$ to recover the key. Therefore, for the proposed scheme, an attacker would need $2000 \times 2^{126.2}$ computational complexity to recover the entire Highway video, which is not considered to be easily achievable by any attacker. The security of the proposed scheme is further tightened by not involving any third party for the key generation or the key management, and the users are encouraged to use SHA-256 bits to generate the key themselves. The security of the SHA-256 bits is very strong since it requires a computational complexity of minimum 2^{178} to be attacked in case of differential attacks. Other types of attacks would require much higher computational complexity to be compromised. The collision probability is also very high ($2^{28.52}$) compared to that of the old SHA1 and MD5 hashes. The perceptual quality of the videos is too bad if decrypted by the CSP since he does not have access to the required video compression information in order to accurately recover the video. The videos appear to be too chaotic to deduce any concrete results from them. As far as convergent encryption is concerned, each user has his/her own key for the decryption of the GOP, which is kept secret from the CSP. Due to the deduplication requirement, the users cannot be allowed to choose different keys or a public/ private key model for the encryption step. The keys generated for two different

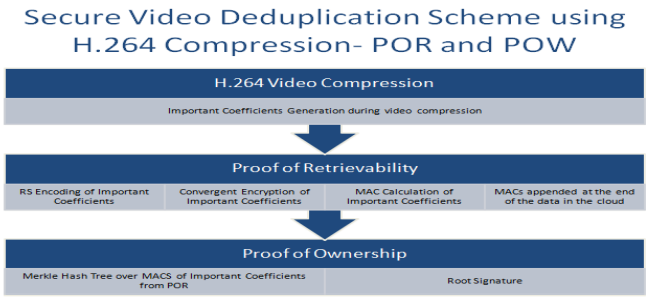

Figure 6.2: Proposed Video POR and POW Schemes

users for the same GOP through convergent encryption are exactly the same. Hence, it is assumed that the users of the same cloud storage will not collude with the CSP, otherwise the security of the proposed scheme could be compromised.

6.3 Proposed Proof of Storage Protocols

The basic functionality of any POR or POW scheme is to guarantee the verifier that the integrity of his/her data will be maintained. The verifier probes the queries to the prover in order to investigate the prover's claim of storing the data truthfully. Any POR or POW scheme should also allow enough queries to be probed before all the queries are exhausted. The number of queries generated for one session should be as large as possible. One naive approach is to download the whole video data and check for its integrity, but in the current era of Big Data (especially videos) occupying a large volume of digital space in the cloud storages, this approach is not applicable. Therefore, the users have to generate the queries before uploading the data to the cloud storage, to be probed later on for POR or POW protocols.

6.3.1 Proof of Retrievability

Provided that the video sequence has been compressed by the H.264 compression algorithm, the signatures have been calculated at the GOP level, and the important parts of the compressed video data data have been partially encrypted, the POR protocol can be applied, typically by encoding those important parts using a forward error correction code. Precisely, let us consider those important data as a file F. The POR protocol works as follows. First, the file F is broken into chunks of blocks using the standard technique of stripping; and an error correction code *c(n, k, d)* over $GF(2^8)$ is applied to each chunk, where n is the length of the codeword, k is the length of the input message, and d is an even value parameter that has the ability to correct up to *d/2* errors. The chunks are then encrypted by the convergent encryption in scheme in order to secure the data from the CSP. The fact that the parameters for the encoding are set by the CSP and the working of convergent encryption ensures that the cross user deduplication can be performed. The output of this operation is a new file *(F = F[1], . . . ,F[b'])*, where $b' = bn/k$, assuming that b is a multiple of k. Second, the MAC values of the encoded chunks are calculated as follows. The new file F (which has b blocks) is partitioned into chunks of v blocks (randomly generated positions by PRF) for some appropriate value of v. A set of MAC values is calculated denoted by $M = m_1, m_2.....m_q$ where m_1 is the aggregated MAC of the blocks for each chunk. The key k for the MAC is kept secret from the cloud storage provider and is only known to the user. The encrypted and encoded chunks are uploaded to the cloud along with the set M through the enterprise server. The user will keep the seed for generating the random index positions used by the PRF . The number of MAC values calculated is equal to the number of chunks that are generated. These MACs are appended at the end of the data to be uploaded in the cloud storage.

In order to verify the integrity of the data, the owner of the file will calculate the challenges as follows. A series of random data blocks positions in each chunk is calculated through PRF as follows: For instance, the user generates the indices with the seed of PRF and knowing v, divide the indices into the group of v chunks. He sends the selected indices for specific chunk, the corresponding index of the m_i in M to the cloud storage provider in the form of $(p_1, p_2.....m_n, m_i)$. The cloud storage provider gives a response in the next response phase by returning the data blocks at the positions $P = p_1, p_2.....p_n$ and the MAC at the position m_i from the set M in the form of $((m_i), (C_{K_1}, C_{K_2}.....C_{K_n}))$. The user can verify the storage of the data by calculating the aggregated MAC of the blocks in each chunk in the response. If both are equal to each other then the POR is successful. In case of any discrepancies, the user is also able to retrieve the file through error encoding used

6.3.2 Proof of Ownership Protocol

The proposed POW protocol is dependent on the POR protocol and does not require any additional calculations. From the above description of the POR protocol, considering the important parts of the compressed video data that have been encrypted and encoded, and considering the file F that have been broken into chunks of (randomly positioned) blocks, the MAC value of each chunk is calculated and stored in a set M. The owner of the file then calculates a binary (MHT) over the set M. The leaves of the tree are denoted by b and the hash function is denoted by h. Let MT_bh be a binary MHT over m using b leaves and the hash function h. These leaves are read from left to right and they are the hash values of the elements in the set $M = m_1, m_2, \ldots, m_v$. Each node has two children and it is a bottom up tree. The leaf nodes L of the tree MT_bh are $H(m_1), H(m_2), \ldots, H(m_v)g$ and any non-leaf node NL of the tree MT_bh is calculated as $NL_i = h(L_ir, L_il)$, where L_ir

and $L_i l$ are respectively the right and left child nodes of NL_i. The root of the tree $MT_b h$ is calculated by hashing up the values of the child nodes for each internal node. The user then calculates the tree MT_b over the set M, signs the root $Rt_(MTbh)$ of the tree, and sends the tree $MT_b h$ and the set M to the CSP during the first upload of the file. The CSP calculates the tree MT_b over M and signs the root if it is the same as that sent by the owner of the file. For the CSP to verify that the user requesting the data is the true owner of it, the CSP will run the POW protocol by sending a randomly selected j i.e. a randomly selected leaf index in M. The user requesting the data will now be required to provide a sibling path from the selected leaf index j to the root. After acquiring this path from the user, the CSP will reconstruct the tree from this provided path and verify the root against the one signed during the initial phase. In case of a matching, the user requesting the data will be accepted as an authorized user. In case of failure to verify the user, the CSP will not release the data. Only the genuine user is expected to own the entire file, along with the encoding and compressed video data. It is safe to assume that this user is the real owner of the file. To run the POW protocol, the only additional data that the CSP stores is the set M and the root. On the other hand, the owner only needs to store the set M.

6.3.3 Experimental Results

Our experiments is performed by keeping in mind the requirements of an efficient and secure POR and POW protocols. First, the protocols should not be computationally heavy since the majority of the processing is done at the users end. Second, the storage requirements should not exceed some reasonable limits as the data deduplication process is meant to save some digital space. Third, the protocols should generate enough challenges in one stretch such that the user can keep the proposed POR and POW schemes on 6 different video sequences,

Video sequence	Data size (KB)	% Increase after encoding	% Increase after MACs
Akiyo	4054	12.54	0.579
Grandma	11756	12.54	0.200
Mobile	4054	12.54	0.579
Foreman	2838	12.54	0.826
Claire	6689	12.54	0.352
Highway	19419	12.54	0.086

Table 6.4: Change in the Size of the Important Data of the Video Sequences

namely Akiyo, Foreman, Claire, Grandma, Highway, and Mobile. These videos are selected for testing because they belong to different classifications of videos in terms of motion, density and texture [77]. Foreman and highway are classified as non-complex textured and medium intensity videos. Grandma, Akiyo and Claire are classified as non-complex textured and low intensity motion videos. Mobile is classified as medium texture video. Each video sequence is first encoded into the QCIF, and then the frames are extracted. The videos are then input into the H.264 compression algorithm with a GOP size of 15. Since we are running our POR and POW protocols in the context of data deduplication, the processing unit is the GOP of the video, which is important for uniquely identifying the parts of the video. Each GOP is further broken down into chunks of blocks which are encoded by the error correcting code.

The first frame is the I frame and the rest of the frames are the P frames, which imply a IPPP format. The QP value is set to *28*. The algorithm works on 4×4, 16×16 MB modes for the I and P frames. The rate of compression for H.264 is set to 30 Hz. The

Video sequence	When $b = 15$	When $b = 20$	When $b = 40$
Akiyo	13520	10120	5060
Grandma	39208	29348	14674
Mobile	13520	10120	5060
Foreman	10140	7590	3795
Claire	22308	16698	8349
Highway	90584	67804	33902

Table 6.5: Number of Queries When b the Number of Blocks for the Video Sequences Varies

details of the algorithm can be found int [67]. The system configuration for the experiment is an Intel Core i3 processor with the 2.40 GHz frequency and 4 GB RAM. The underlying operating system is operating on 64 bits, and our schemes have been implemented using the MATLAB R2014a version. Various methods, all implemented in MATLAB, are used for the extraction and processing of the QCIF format videos, as well as for the extraction of the frames. We have simulated the number of challenges for different dimensions of the video using the Communication Systems Toolbox (for Reed Solomon Codes). The results of the experiments are presented in Table: 6.4 and Table: 6.5.

Table: 6.4 shows the percentage of increase in the size of the file after applying the error correction encoding. It also shows the percentage of increase after 20 bit MAC values are appended at the end of the data. Each GOP is composed of blocks of size 20 bytes. The original size of the file is the size of the data which is considered important in order to decrypt/decode the GOP. From Table: 6.4, it can be observed that the first increase in size is in the range of 12%, which is not a huge increase. But, still this volume of additional data has to be stored in order to ensure the security of the video file using the POR protocol. The fact that we are only encrypting and encoding the important parts of the video file and

not the entire file also declines the volume of the data overall. The percentage of increase is almost the same in case of all the videos because we are using the same standard RS encoding on all of the videos, with the same parameters. The same error correction code C(n, k, d) over $GF(2^m)$ is applied to each chunk, where $m = 8$, $n = 256$, $k = 223$, and $d = 32$ (i.e. up to 2 errors can be corrected). It is also observed that the percentage of increase after the MACs are appended to the file is nominal in case of all the files and does not incur much overhead. It should be noted that the values in the Table: 6.4 have been calculated by using 15 as the value of the blocks in each chunk.

Table: 6.5 shows an important characteristic of the proposed scheme in terms of number of challenges or queries that the owner of the file can probe in one stretch. To illustrate this aspect, we consider the example of the Grandma video sequence. In Table: 6.5, it can be observed that the number of queries varies by changing the number of blocks in each chunk. If each chunk has 15 blocks, the obtained number of queries is 39208 in the case of the Grandma video, which is a big enough for the owner of the file to be exhausted before the queries are actually generated. It is also observed that when the number of blocks per chunk increases, the number of queries decreases. Considering the case of the Grandma video, where 14674 queries have been generated using chunks of 40 blocks, the owner of the file can ask a single query for almost 40 years without running out of challenge for one single GOP. It should be noted that the number of blocks can be selected depending upon the security required by the file. The security of the proposed POR scheme will become tighter with higher number of blocks per chunk and weaker with lower number of blocks per chunks. In any case, this much computation and storage is a fair tradeoff between the security and the storage requirements for the proposed secure POR scheme. The security of the POR scheme is further tightened by encrypting the important part of the compressed video sequence and by decoding it.

Moreover, the ECC can tolerate a corruption of up to z-fraction of data blocks in the file F (z is dependent on which ECC is used) [87]. This feature implies that if the CSP deletes or modifies the z-fraction of the file F, the z-fraction of sentinels will also be changed with a high probability. Therefore, if the verifier probes enough sentinels, he/she can detect the modifications performed on the z-fraction of F. The security of the POR scheme also depends on the number of queries per video sequence which we have shown to be a big number (Table: 6.5). The POW protocol is efficient in terms of computational overhead in the sense that it does not require any calculation of the data separately, but uses the data already generated for the POR scheme. For the video files with 40 blocks per chunks, the number of chunks is 253 in the case of the Grandma video sequence, which implies that the height of the tree is $O(logn)$ where $n = 253$. The time complexity to built the MHT is also proportional to $O(logn)$; therefore, the operation of insertion, space, and time required for carrying out the POW protocol using the MHT approach does not incur any computational overhead due to the small size of the important parts of the compression video. The use of hashes in the underlying data structure of the MHT further increases the security of the POW scheme.

6.4 Chapter Summary

We have proposed a secure video deduplication scheme through video compression in cloud storage environments. The proposed scheme is made of three components: H.264 video compression scheme, signature generation from the compressed videos, and selective encryption of the compressed videos. The compressed video obtained from the H.264 algorithm is partially encrypted through convergent encryption, in such a way that the semi-honest CSP or any malicious user cannot have access to it in plain, hence ensuring the security of

the video data from the CSP or any malicious user. The hash signatures are generated in such a way that they are robust enough to identify the GOPs uniquely and allow the CSP to differentiate among other GOPs by comparing the signatures rather than scanning the entire video sequence. Breaking the video into the GOP also increases the probability of capturing more duplicate data and therefore increases the deduplication ratio. Experimental results have shown that: (1) For complex textured videos, the percentage of digital storage space saved by the CSP practicing cross-user deduplication using our scheme is higher than that of the non-complex textured video; (2) For videos that are large in size and high motion intensity, there are substantial space savings for the CSP practicing cross-user deduplication using our scheme; (3) the average time taken to encode the videos is much higher than that taken to calculate the signature; (4) the proposed scheme is secured against the semi-honest CSP since the CSP does not have access to the video compression information required for the recovery of the video. For the POR and POW protocols, through experiments using 6 different video sequences, we have determined how much volume of additional data has to be stored in the cloud storage as a result of the data deduplication process, in order to ensure the security of the video file using the POR protocol. We have also shown that the POW protocol is efficient in terms of computational overhead since it does not require any calculation of the data separately, but uses the data already generated for the POR scheme.

Chapter 7

Conclusions and Future work

7.1 Conclusion

As digital data is growing tremendously, cloud storage services are gaining popularity since they promise to provide convenient and efficient storage services that can be accessed anytime, from anywhere. At the same time, with the advent of cloud computing and its digital storage services, the growth of digital content has become irrepressible at both the enterprise and individual levels. These huge volumes of data need some practical platforms for the storage, processing and availability and cloud technology offers all the potentials to fulfill these requirements. A major issue hindering the acceptance of cloud storage services by users is the data privacy issue associated with the cloud paradigm. Indeed, although data is outsourced in its encrypted form, there is no guarantee of data privacy when an honest but curious CSP handles the management of confidential data while these data reside in the cloud. CSPs are able to maximize data storage space by incorporating data deduplication into cloud storage. Although data deduplication removes data redundancy and data replication by storing only a single copy of previously duplicated data, it also introduces major

data privacy and security issues for the user. In this thesis we have address the issue of the security of the data in the cloud storages when data deduplication is being practiced by the CSP.

First, we have introduced a data deduplication framework, with the goal of preserving to preserve the privacy of data in the cloud while ensuring that the CSP can perform data deduplication without compromising the data privacy and security. This framework is made of five modules. The first module (*File chunking* module) is meant to divide the file into smaller units called chunks. The second module (*chunk encryption* module) is responsible for encrypting these chunks using a secure hash scheme. The third module (*index generation* module) is designed to generate (via B^+ trees) an index of the encrypted units of the data obtained from the second module. The fourth module (*index encryption* module) is responsible for encrypting this index using an asymmetric searchable encryption scheme. This encrypted index will then be used by the CSP to search for the data in the Cloud. Finally, the fifth module (*data and index encoding* module) is responsible for organizing the metadata, the encrypted data chunks, and the encrypted index, in such a way that the user can at anytime verify of the integrity of its data. The highlights of the proposed framework include the fact that the CSP will only have the access to the encrypted data but still will be able to perform operations such as search over the encrypted data and running the proof of storage protocols to ensure the integrity of the uploaded data.

Second, we have proposed a two-level data deduplication framework can be used by the enterprises who utilize the data storage services from the same CSP. By first applying the cross user level deduplication and then the cross enterprise level deduplication, the CSP can maximize the digital space savings and minimize the cost. We start our framework by first defining a secure indexing scheme using $B^\star$ trees and structure the index in such a way that data deduplication can be supported at the enterprise and single user level. We also

use convergent encryption to encrypt the index for enhancing the security. A multi-user searchable encryption scheme for enterprise level deduplication is then designed and the files sharing activity between the users within the enterprise is examined in-depth. We use the private key word search strategy to make the multi user searchable encryption possible at the enterprise level. Our scheme also allows a secure and efficient file sharing mechanism within the enterprise since it a vital requirement for successful operations. In order to verify the integrity of the data in the cloud, a privacy-preserving scheme is designed for assuring the data integrity in terms of proof of retrievability and proof of ownership in the context of cross-user client-side data deduplication for medium-sized and small-sized enterprises. Through our security analysis, we showed that the proposed framework is safe from attacks by internal and external adversaries under certain assumptions. Examples of such attacks include: identifying files, learning the contents of files and poison attacks. Our analysis also showed that the proposed POR and POW protocols are robust enough to catch a malicious user and at the same time offers enough queries in one session for the user to probe from the CSP. Our performance analysis have shown that our framework uniformly distributes the computational load between the users and the servers and at the same time keeps the operational cost minimum by utilizing the information already generated during different stages of the functionalities.

Third, secure image deduplication scheme through image compression compression is proposed, which is made of three components as follows. SPIHT algorithm is applied on the image in order to minimize its size as much as possible. The compressed image is analyzed and only the important information of the compressed data is encrypted. This partial encryption will not only save the computational cost of encryption, but will also enhance the security of the data from the CSP. In order to apply image deduplication, a unique signature of the image is generated from the compressed data so that the CSP can identify

the same compressed images coming from different users. Using this same set of information (generated during compression, encryption and signature generation process),POR and POW schemes for image deduplication are proposed, and their efficiency and security are demonstrated by experiments. The experimental results have shown that our proposed image deduplication framework can identify the minor differences among the images uploaded by different users through their signatures. We have used 6 different images and generate 120 variations of these images to verify our claim . This is justified by the fact that in case the images appear to be identical, the CSP will delete the duplicate data, therefore even slightly different images should be counted as two separate entities. We have also proved that the security of the proposed scheme is tight enough for the CSP to read the underlying image data due to the fact that the encryption is applied over the compression. The efficiency of our proposed POR and POW schemes is attributed to the fact that these protocols are designed in such a way that enough challenges can be generated for the users in one session to probe the CSP. We have run our experiments for different resolutions of the same image and we have shown that the number of challenges increases as the resolution of the image increases. The efficiency of the protocols is also based on the fact that no additional computational cost is incurred to implement the POR and POW on top of our proposed image deduplication scheme.

Fourth, we have proposed a secure video deduplication scheme for cloud environments, which is composed of three main steps. The first step consists of applying the H.264 compression algorithm on the video data, yielding the compressed video. For the processing, we divide the videos are divided into group of pictures (GOP) based on similarity, where each GOP is made of I frames and P frames. This break down enhances the chances of catching more duplicate video data since videos are often huge in size. The second step consists of getting the user to generate a unique signature of the compressed video which will be used

by the CSP to discriminate between the compressed video data originated from the owner of the data and that originated from other users and to ease the deduplication process. The third step consists of getting the user applying a partial encryption on the compressed video - this step is meant to prevent the semi-honest CSP from having access to the plain video data. We run our experiments on 3 different types of 6 videos and showed that the time to encode the videos is much larger than the time to calculate the signatures. The videos are almost always compressed before being upload, therefore our scheme for deduplication (the signature generation) will incur less overheads. We carried out the experiments on the same set of 6 videos to determine the efficiency of our proposed POR and POW protocols and found out that the volume of additional data to be stored in the cloud for the running of the protocols is nominal as compared to the original data. The efficiency is further increased by building our framework in an incremental manner such that each component is using the information generated from its predecessor.

7.2 Future Work

As future work, we plan to design a compressed sensing (CS) based scheme for tightening the security of our proposed deduplication schemes. Using different sampling methods and measurement matrices for different types of data will help minimizing the computational cost for the security of the proposed schemes . In this regard, we plan to study the Non-Deterministic and Non Adaptive Measurement matrices / Encodings along with the Non-Deterministic and Adaptive Measurement matrices / Encodings for deduplication purpose, considering the three different types of data, namely text, video and image. In order to tailor our deduplication schemes in accordance with the requirements of the compressed sensing, we will explore some CS algorithms to find the best fit for our multi media

deduplication. These includes: the Coordinate Gradient Descent Method for l1 regularized Convex Minimization , the Bregman Iterative Algorithms, the Bayesian Compressive Sensing, the Two Step Reweighted l1, the Chambolle's algorithm and the Split Bregman Method.

We also plan to implement a privacy preserving public auditing of the uploaded data in the cloud such that the third party external auditor (TPA) should not learn about the content of the data. We have assumed semi honest CSP and wish to allow the TPA to only audit the data for correctness without learning about it. We will be using the technique of public key based homomorphic linear authenticator (HLA) which will not require the auditor to download the entire data from the cloud. Moreover, we will be incorporating some basic MAC solutions for this issue to support the HLA solutions.

We also plan to use artificial intelligence (AI) techniques (such as neural networks) to identify the duplicates in an efficient and fast manner. This will allow to make more customized and user friendly methods for identifying the duplicates. We also plan to address the scenario where dynamic data is stored in the cloud and the CSP wishes to implement data deduplication in order to save digital space.

Bibliography

[1] "Dropbox hacked." http://www.businessinsider.com/dropbox-hacked-2014-10. Accessed on November 2014.

[2] I. D. Corporation, "The digital universe decade - are you ready?." Online, 2010.

[3] "Can-you-compress-and-dedupe-it-depends." http://storagesavvy.com. Accessed on October 2012.

[4] "A-guide-to-data-de-duplication." http://www.computerweekly.com/feature/A-guide-to-data-de-duplication. Copyright 2015.

[5] "Dropbox hacked." http://www.businessinsider.com/dropbox-hacked- 2014-10. Accessed on November 2014.

[6] "Deduplication internals." https://pibytes.wordpress.com/2013/02/17/deduplication-internals-content-aware-deduplication-part-3/. Accessed on February 2015.

[7] I. S. Storage, "Ibm protectier deduplication." http://www-03.ibm.com/systems/storage/tape/protectier/index.html. Online.

[8] A. Osuna, E. Balogh, A. Ramos, G. de Carvalho, R. F. Javier, and Z. Mann, "Implementing ibm storage data deduplication solutions." http://www.redbooks.ibm.com/redbooks/pdfs/sg247888.pdf, 2011.

[9] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, (New York, NY, USA), pp. 195–206, ACM, 2013.

[10] F. Rashid, A. Miri, and I. Woungang, "A secure data deduplication framework for cloud environments," in *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, pp. 81–87, IEEE, 2012.

[11] F. Rashid, A. Miri, and I. Woungang, "Secure enterprise data deduplication in the cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pp. 367–374, IEEE, 2013.

[12] F. Rashid, A. Miri, and I. Woungang, "Proof of retrieval and ownership protocols for enterprise-level data deduplication," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, pp. 81–90, IBM Corp., 2013.

[13] F. Rashid, A. Miri, and I. Woungang, "Proof of retrieval and ownership protocols for images through spiht compression," in *6th Intl. Symposium on Cyberspace Safety and Security*, IEEE, 2014.

[14] F. Rashid, A. Miri, and I. Woungang, "A secure video deduplication in cloud storage environments using h.264 compression," in *Proc. of the First IEEE Intl. Conference on BigData Computing Service and Applications San Francisco Bay, USA*, IEEE, March 2015.

[15] F. Rashid, A. Miri, and I. Woungang, "Proof of storage for video deduplication in the cloud," in *IEEE 4th International Congress on Big Data, New York, USA*, IEEE, August 2015.

[16] C. Bo, Z. F. Li, and W. Can, "Research on chunking algorithm of data deduplication," *American Journal of Engineering and Technology Research*, vol. 11, no. 9, pp. 1353 – 1358, 2011.

[17] E. Kave and H. K. Tang, "A framework for analyzing and improving content based chunking algorithms," tech. rep., International Enterprise Technologies Laboratory, HP Laboratories Palo Alto, Sept 2005.

[18] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, (New York, NY, USA), pp. 174–187, ACM, 2001.

[19] C. Wang, Z. guang Qin, J. Peng, and J. Wang, "A novel encryption scheme for data deduplication system," in *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*, pp. 265 –269, july 2010.

[20] S. Parez, "Bitcasa: Infinite cloud storage." http://techcrunch.com/2011/09/18/bitcasa-explains-encryption/, Sept 2011. Online.

[21] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin, "Secure anonymous database search," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, (Ner York, NY, USA), pp. 115–126, ACM, 2009.

[22] T. Thwel and N. Thein, "An efficient indexing mechanism for data deduplication," in *Current Trends in Information Technology (CTIT), 2009 International Conference on the*, pp. 1 –5, dec. 2009.

[23] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," in

Proccedings of the 7th conference on File and storage technologies, FAST '09, (Berkeley, CA, USA), pp. 111–123, USENIX Association, 2009.

[24] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, pp. 1 –9, sept. 2009.

[25] J. Dinerstein, S. Dinerstein, P. Egbert, and S. Clyde, "Learning-based fusion for data deduplication," in *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pp. 66 –71, dec. 2008.

[26] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, (Berkeley, CA, USA), pp. 18:1–18:14, USENIX Association, 2008.

[27] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proceedings of the 14th international conference on Financial cryptograpy and data security*, FC'10, (Berlin, Heidelberg), pp. 136–149, Springer-Verlag, 2010.

[28] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *Proceedings of the 5th international conference on Information Security Applications*, WISA'04, (Berlin, Heidelberg), pp. 73–86, Springer-Verlag, 2005.

[29] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, (Berlin, Heidelberg), pp. 535–554, Springer-Verlag, 2007.

[30] J. Baek, R. Safavi-Naini, and W. Susilo, "On the integration of public key data encryption and public key encryption with keyword search," in *Proceedings of the 9th international conference on Information Security*, ISC'06, (Berlin, Heidelberg), pp. 217–232, Springer-Verlag, 2006.

[31] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proceeding sof the international conference on Computational Science and Its Applications, Part I*, ICCSA '08, (Berlin, Heidelberg), pp. 1249–1259, Springer-Verlag, 2008.

[32] T. Fuhr and P. Paillier, "Decryptable searchable encryption," in *Proceedings of the 1st international conference on Provable security*, ProvSec'07, (Berlin, Heidelberg), pp. 228–236, Springer-Verlag, 2007.

[33] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith, III., "Public key encryption that allows pir queries," in *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, (Berlin, Heidelberg), pp. 50–67, Springer-Verlag, 2007.

[34] A. Juels and J. Burton S. Kaliski, "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, (New York, NY, USA), pp. 584–597, ACM, 2007.

[35] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, (New York, NY, USA), pp. 598–609, ACM, 2007.

[36] H. Pang, J. Zhang, and K. Mouratidis, "Enhancing access privacy of range retrievals over B+trees," *Knowledge and Data Engineering, IEEE*, pp. 99–99, 2012.

[37] S. Wu, D. Jiang, B. Ooi, and K. Wu, "Efficient B+tree based indexing for cloud data processing," *The Proceedings of the VLDB Endowment (PVLDB)*, vol. 3, pp. 1207–1218, Sep 2010.

[38] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 79–88, ACM, 2006.

[39] Y. Yang, H. Lu, and J. Weng, "Multi-user private keyword search for cloud computing," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 264 –271, 29 2011-dec. 1 2011.

[40] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt 2004*, pp. 506–522, Springer, 2004.

[41] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log," in *Proceedings of 11th Annual Network and Distributed System Security Symposium (NDSS 2004)*, vol. 6, 2004.

[42] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Communications (ICC), 2012 IEEE International Conference on Communications, ICC'12, Ottawa*, pp. 917–922, June 2012.

[43] B. Furht and D. Kirovski, *Multimedia security handbook*, vol. 158. CRC press New York, 2005.

[44] M. F. E. Shusterman, "Image compression via improved quadtree decomposition algorithms," in *IEEE Transactions on Image Processing*, vol. 3, pp. 207–215, IEEE, Mar. 1994.

[45] F. Keissarian, "A new quadtree-based image compression technique using pattern matching algorithm," in *International Conference on Computational & Experimental Engineering and Sciences (ICCES)*, vol. 12, pp. 137–143, 2009.

[46] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3445–3462, 1993.

[47] P. Singh and P. Singh, "Design and implementation of ezw & spiht image coder for virtual images," *International Journal of Computer Science and Security (IJCSS)*, vol. 5, no. 5, p. 433, 2011.

[48] "Spiht image compression." http://www.cipr.rpi.edu/research/SPIHT/spiht1.html. Accessed on October 2013.

[49] C.-Y. Lin and S.-F. Chang, "Authentication of zero-tree wavelet encoded images," *ADVENT Project Report, Columbia University*, 1999.

[50] K. Martin, *Secure Wavelet-Based Coding of Images, and Application to Privacy Protected Video Surveillance*. PhD thesis, University of Toronto, 2010.

[51] K. Martin, R. Lukac, and K. N. Plataniotis, "Efficient encryption of wavelet-based coded color images," *Pattern Recognition*, vol. 38, no. 7, pp. 1111–1115, 2005.

[52] K. Ravishankar and M. Venkateshmurthy, "Region based selective image encryption," in *Computing & Informatics, 2006. ICOCI'06. International Conference on*, pp. 1–6, IEEE, 2006.

[53] N. Taneja, B. Raman, and I. Gupta, "Partial encryption on spiht compressed images," in *Pattern Recognition and Machine Intelligence*, pp. 426–431, Springer, 2009.

[54] H. Luo, F.-X. Yu, H. Li, and Z.-L. Huang, "Color image encryption based on secret sharing and iterations.," *Information Technology Journal*, vol. 9, no. 3, 2010.

[55] H.-q. YANG, X.-F. Liao, K. W. WONG, W. Zhang, P.-C. Wei, *et al.*, "Spiht-based joint image compression and encryption," *Wuli Xuebao/Acta Physica Sinica*, 2012.

[56] S. I. R. C. Rengarajaswamy, "Spiht compression on encrypted images," in *Proc. of IEEE Conference on Information & Communication Technologies (ICT), Jeju Islan, Korea*, pp. 336–341, IEEE, Apr. 2013.

[57] S. G. Amrita Sengupta, "Compression of encrypted images using chaos theory and spiht," in *IJCA Proc. On Intl. conference on Green Computing and Technology ICGCT*, pp. 10–15, IEEE, Oct. 2013.

[58] X. Zhang and X. Wang, "Chaos-based partial encryption of spiht coded color images," *Signal Processing*, vol. 93, no. 9, pp. 2422–2431, 2013.

[59] J. Lee and B. Jeon, "Fast mode decision for h.264," in *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, vol. 2, pp. 1131–1134 Vol.2, June 2004.

[60] S.-H. Yang and C.-F. Chen, "Robust image hashing based on spiht," in *Information Technology: Research and Education, 2005. ITRE 2005. 3rd International Conference on*, pp. 110–114, IEEE, 2005.

[61] A. Swaminathan, Y. Mao, and M. Wu, "Robust and secure image hashing," *Information Forensics and Security, IEEE Transactions on*, vol. 1, no. 2, pp. 215–230, 2006.

[62] E. Y. Chang, J. Z. Wang, C. Li, and G. Wiederhold, "Rime: A replicated image detector for the world wide web," in *Photonics East (ISAM, VVDC, IEMB)*, pp. 58–67, International Society for Optics and Photonics, 1998.

[63] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," in *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, pp. 178–183, IEEE, 2000.

[64] H. Cheng and X. Li, "Partial encryption of compressed images and videos," *Signal Processing, IEEE Transactions on*, vol. 48, no. 8, pp. 2439–2451, 2000.

[65] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3445–3462, 1993.

[66] I. E. Richardson, *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[67] "H.264 baseline codec v2." http://www.mathworks.com/matlabcentral/fileexchange/40359-h-264-baseline-codec-v2. Updated on February 2013.

[68] B. Coskun, B. Sankur, and N. Memon, "Spatio–temporal transform based video hashing," *Multimedia, IEEE Transactions on*, vol. 8, no. 6, pp. 1190–1208, 2006.

[69] R. K. Vadapalli and P. Bora, "Perceptual video hashing based on 3d spiht coding of 3d dwt coefficients," in *National Conference on Communications (NCC-2008)*, pp. 173–177, 2008.

[70] S. Lee and C. D. Yoo, "Robust video fingerprinting for content-based video identification," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 7, pp. 983–988, 2008.

[71] S. Lee, C. D. Yoo, and T. Kalker, "Robust video fingerprinting based on symmetric pairwise boosting," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 9, pp. 1379–1388, 2009.

[72] K. Saadi, A. Bouridane, and A. Gessoum, "H. 264/avc video authentication based video content," in *I/V Communications and Mobile Network (ISVC), 2010 5th International Symposium on*, pp. 1–4, IEEE, 2010.

[73] J. Scharinger, "Fast encryption of image data using chaotic kolmogorov flows," *Journal of Electronic Imaging*, vol. 7, no. 2, pp. 318–325, 1998.

[74] S. Li, C. Li, G. Chen, N. G. Bourbakis, and K.-T. Lo, "A general quantitative cryptanalysis of permutation-only multimedia ciphers against plaintext attacks," *Signal Processing: Image Communication*, vol. 23, no. 3, pp. 212–223, 2008.

[75] C. Shi and B. Bhargava, "An efficient mpeg video encryption algorithm," in *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*, pp. 381–386, IEEE, 1998.

[76] L. Varalakshmi, G. Sudha, and V. Vijayalakshmi, "Enhanced encryption schemes of video for real time applications," in *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*, pp. 408–413, IEEE, 2011.

[77] Y. Zhao and L. Zhuo, "A content-based encryption scheme for wireless h. 264 compressed videos," in *Wireless Communications & Signal Processing (WCSP), 2012 International Conference on*, pp. 1–6, IEEE, 2012.

[78] S. Lian, Z. Liu, Z. Ren, and H. Wang, "Secure advanced video coding based on selective encryption algorithms," *Consumer Electronics, IEEE Transactions on*, vol. 52, no. 2, pp. 621–629, 2006.

[79] A. Katiyar and J. Weissman, "Videdup: an application-aware framework for video de-duplication," in *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, pp. 7–7, USENIX Association, 2011.

[80] K.D.Bowers, A.Juels, and A.Oprea, "Proofs of retrievability: theory and implementation," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pp. 43–54, 2009.

[81] G.Ateniese, R.Burns, R.Curtmola, J.Herring, L.Kissner, Z.Peterson, and D.Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pp. 598–609, 2007.

[82] G.Ateniese, S.Kamara, and J.Katz, "Proofs of storage from homomorphic identification protocols," in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pp. 319–333, 2009.

[83] G.Ateniese, R.Burns, R.Curtmola, J.Herring, O.Khan, L.Kissner, P.Zachary, and D.Song, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, pp. 12:1–12:34, June 2011.

[84] S.Halevi, D.Harnik, B.Pinkas, and S.Alexandra, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pp. 491–500, 2011.

[85] M.Lillibridge, S.Elnikety, A.Birrell, M.Burrows, and M.Isard, "A cooperative internet backup scheme," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 3–3, USENIX Association, 2003.

[86] H.Shacham and B.Waters, "Compact proofs of retrievability," in *Advances in Cryptology-ASIACRYPT 2008*, pp. 90–107, Springer, 2008.

[87] A.Juels, Jr.Kaliski, and S.Burton, "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pp. 584–597, 2007.

[88] Q.Zheng and S.Xu, "Secure and efficient proof of storage with deduplication," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, CODASPY '12, pp. 1–12, 2012.

[89] S.Kumar and R.Subramanian, "An efficient and secure protocol for ensuring data storage security in cloud computing," *International Journal of Computer Science*, vol. 8, 2011.

[90] Q.Wang, C.Wang, K.Ren, W.Lou, and J.Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 5, pp. 847–859, 2011.

[91] W.K.Ng, Y.Wen, and H.Zhu, "Private data deduplication protocols in cloud storage," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pp. 441–446, 2012.

[92] F. Software, "How data deduplication works." http://www.falconstor.com/library/documents/white-papers/falconstor-vtl-white-papers, Sept 2011. White Paper.

[93] M. Bellare, A. Boldyreva, and A. ONeill, *Advances in Cryptology - CRYPTO 2007*, vol. 4622 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007. Chapter: Deterministic and Efficiently Searchable Encryption.

[94] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, (New York, NY, USA), pp. 598–609, ACM, 2007.

[95] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, (New York, NY, USA), pp. 584–597, ACM, 2007.

[96] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, SecureComm '08, (New York, NY, USA), pp. 9:1–9:10, ACM, 2008.

[97] F.Rashid, A.Miri, and I.Woungang, "A secure data deduplication framework for cloud environments," in *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, pp. 81 –87, July 2012.

[98] Wikipedia, "B-tree." http://en.wikipedia.org/wiki/B-tree. Accessed on April 2013.

[99] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou, "Sam: A semantic-aware multi-tiered source de-duplication framework for cloud backup," in *Parallel Processing (ICPP), 2010 39th International Conference on*, pp. 614 –623, Sept. 2010.

[100] C. Soghoian, "How dropbox sacrifices user privay for cost savings." http://paranoia.dubfire.net/2011/04/how-dropbox-sacrifices-user-privacy-for.html, April 2011. Online.

[101] "Spideroak- zero knowledge data backup, sync, access, storage and share from any device." https://spideroak.com/. Accessed on March 2013.

[102] A. J. Menezes, P. C. V Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press LLC, 1997.

[103] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security and Privacy*, vol. 8, pp. 40–47, 2010.

[104] J. Xu, E. Chang, and J. Zhou, "Leakage-resilient client-side deduplication of encrypted data in cloud storage," tech. rep., Cryptology ePrint Archive, 2011.

[105] F.Rashid, A.Miri, and I.Woungang, "Secure enterprise data deduplication in the cloud," in *Accepted in : IEEE CLOUD 2013, IEEE 6th International Conference on Cloud ComputingPrivacy*, June 2013.

[106] L. Shu and D.J.Costello, *Error Control Coding, Second Edition*. Prentice-Hall, Inc., 2004.

[107] "Computerworld, "data deduplication in the cloud explained, part one",." http://www.computerworld.com/article/2474479/data-center/data-deduplication-in-the-cloud-explained–part-one.html. Last visited Nov 16, 2014.

[108] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 3, pp. 243–250, 1996.

[109] K. Saadi, A. Bouridane, and A. Guessoum, "Combined fragile watermark and digital signature for h. 264/avc video authentication," in *The 17th European signal Processing Conference (EUSIPCO 2009), Glasgow, Scotland*, 2009.

[110] T. Stutz and A. Uhl, "A survey of h. 264 avc/svc encryption," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 3, pp. 325–339, 2012.

[111] N. M. Thomas, D. Lefol, D. R. Bull, and D. Redmill, "A novel secure h. 264 transcoder using selective encryption," in *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, vol. 4, pp. IV–85, IEEE, 2007.