Flexible yet Secure De-duplication Service for Enterprise Data on Cloud Storage

Wen Bing Chuan[†*], Shu Qin Ren[‡], Sye Loong Keoh[†] and Khin Mi Mi Aung[‡]
[†]School of Computing Science, University of Glasgow Singapore, Singapore 737729
[*]Singapore Institute of Technology, SIT@RP Campus, Singapore 737729
[‡]Data Center Technologies Division, Data Storage Institute, A*STAR, Singapore 138932
Email: {2109934c, SyeLoong.Keoh}@glasgow.ac.uk, {Ren_Shuqin, Mi_Mi_Aung}@dsi.a-star.edu.sg

***Abstract*—The cloud storage services bring forth infinite storage capacity and flexible access capability to store and share large-scale content. The convenience brought forth has attracted both individual and enterprise users to outsource data service to a cloud provider. As the survey shows 56% of the usages of cloud storage applications are for data back up and up to 68% of data backup are user assets. Enterprise tenants would need to protect their data privacy before uploading them to the cloud and expect a reasonable performance while they try to reduce the operation cost in terms of cloud storage, capacity and I/Os matter as well as systems' performance, bandwidth and data protection. Thus, enterprise tenants demand secure and economic data storage yet flexible access on their cloud data.**

In this paper, we propose a secure de-duplication solution for enterprise tenants to leverage the benefits of cloud storage while reducing operation cost and protecting privacy. First, the solution uses a proxy to do flexible group access control which supports secure de-duplication within a group; Second, the solution supports scalable clustering of proxies to support large-scale data access; Third, the solution can be integrated with cloud storage seamlessly. We implemented and tested our solution by integrating it with *Dropbox*. Secure de-duplication in a group is performed at low data transfer latency and small storage overhead as compared to de-duplication on plaintext.

I. Introduction

In contrast to traditional storage services with fully trusted infrastructure and management, cloud storage provides tenants with a transparent service, like elastic capacity and flexible accessibility, without the need to manage troublesome infrastructure. Individual users are already enjoying the flexibility, accessibility and data management provided by cloud storage services such as gmail, dropbox and wechat. People are demanding more storage space from service providers to backup [1], share documents, photos and videos with friends [13]. All these benefits are based on the assumptions that individual users trust the service providers or take the risk of exposing their data to the service providers, thus a privacy issue.

For enterprise tenants/users, data growth is tremendous with online business transactions, and it will continue to be so. The demand for outsourcing data storage and management has increased dramatically. The study from TheInfoPro's Wave shows that on-premises private cloud will host 30% and off-premises public cloud will host 15% in IT service by 2015 [7]. Cloud storage space unfortunately, are not free. Major providers like Amazon Cloud Storage service charges consumers annually based on the amount of storage space they purchase on an annual contract. Enterprise and corporate rely their daily business functions on cloud storage. Basic business operations such as data backup or recovery or cloud databases increase the demand of cloud storage services. If redundant data are not managed well, it would take up too much unnecessary storage space, and hence incurring extra cost. This often drive consumers into outsourcing for techniques aimed to minimize space usage. *Data De-duplication* is an attractive technology that reduces storage space and network bandwidth during data transfer in order to cater for the vast amount of redundant data. This technique exploits the content of data file by removing the need of keeping multiple copies of files with the same content through the elimination of duplicates.

Apart from optimizing the storage space, it is important that enterprises have control over the access to their content stored in the third party providers. The continuous report on data leakage caused by inherent loopholes with the Internet has raised concerns with regards to data security on a large scale. Storage providers are inadequate as the only provider of data security. Following reports of data leakage on platform such as Dropbox [3] and iCloud [5], end users have realized that they have a part to play in securing their own digital assets. Data leakage often compromise data confidentiality and integrity, this induces significant impact that could cost an individual his/her reputation or monetary loss, and it could potentially bring down a large corporation in a jiffy. In addition, the multi-tenant nature of cloud computing exacerbates the security vulnerabilities, resulting in the integrity of data is consistently at risk.

Data protection and service cost are two top issues concerned by enterprise tenants, which are extra features demanded from the individual users. Business data is vital to companies, they can get reliable, available, fault-tolerance and performance from cloud service providers, but they cannot take the risk of letting the service provider to scan data or charge expensive service fee. If enterprise tenants were willing to outsource data storage to cloud, they would prefer flexible but secure data storage and management services while keeping the service fee as low as possible. For example, they would like the cloud service to manage their data without knowing the data contents while cutting off all the unecessary cost. We consider the application scenario where an enterprise tenant has a group of users who share data storage through an untrusted service provider as what they did with NAS or SAN. Since the data stored is on untrusted site, all users' data are encrypted before they are uploaded to the cloud.

In this paper, we propose a system that facilitates secure file sharing over cloud based storage in a space and bandwidth efficient manner while leveraging on the availability, flexibility and capacity provided by the cloud. We outline the contributions of this paper as the following:

- A group-enabled De-duplication scheme that allows de-duplicaiton on encrypted data across users.
- Secure sharing within multi-user group is enabled to support fine-grained access control to the cloud storage.
- Scalable and secure enterprise proxy solution that provides transparent data access and protection.

This paper is organized as follows: Section II reviews the literature and related work. Section III describes the problem statement. Section IV presents the system architecture of the proposed proxy-based access control and de-duplication service. In Section V, we present the implementation details, while Section VI describes the evaluation results. We present security analysis of our solution in Section VII. Finally, we conclude the paper with future work in Section VIII.

II. Background and Related Work

A. Space Saving with De-duplication

The advent of big data, mobile computing and social networking generates data deluge and demands for big volumes of storage. Recent survey by Gartner shows that data growth forms high cost for hardware infrastructure in data center. Data de-duplication has been performed by commercial cloud storage services such as Google Drive [4], Dropbox [3] and bitcasa [2] across users to save space. In such a multi-tenant environment, data duplication occurs at high possibility and de-duplication results in substantial economic benefits for cloud provider [14].

De-duplication [11] is a process of identifying redundancy in data content and denying this incoming data if it matches an existing record. Hence, only a unique single copy of the data is stored and will be made available to all the authorized users. Rashid [18] proposed a framework that implements block-level data de-duplication so that files are divided into blocks and de-duplicated. To fully utilize the benefit of data de-duplication, *cross-user de-duplication* is used in practice. It identifies redundant data across different users and then removes the redundancy and therefore saving storage space. The authors also pointed out that an average of 60% of data can be de-duplicated for an individual using cross-user de-duplication technique. Thus, proving that data de-duplication is capable of supporting the integration with cloud storage to provide space efficient storage on a lower cost and bandwidth consumption.

However, there are several security drawbacks in data de-duplication, e.g., the issue of data privacy and integrity. De-duplication cross users can potentially lead to information leakage to malicious users through side channel attacks. Despite the effort invested in improving the existing de-duplication algorithm to provide users with adequate privacy, thus far there is no work that has a solution for an effective and secure combination of the two [18]. Current systems rely primarily upon three main data de-duplication strategies [20], [17] as follows: Firstly, *Whole File* strategy typically utilises a file's cryptographic hash value as an identifier. If two or more files hash are of the same value, they are assumed to have identical content and only stored once. This is the simplest and the most straightforward form of data de-duplication. Secondly, *fixed-sized chunks* where it breaks a whole file into n number of pre-determined fixed-size chunks. Such chunk-level data de-duplication renders more flexibility and efficiency in terms of the depth of de-duplication. Each chunk is stored in a data store. During de-duplication process, each chunk is analyzed and identical chunks will not be stored into the data store, hence saving storage space. This however, requires system to keep track of a list the files and their associated data chunks. When the files are requested, system will compute the chunks into a whole file and have the file returned. One missing chunk will raise issues in this de-duplication strategy. The last, and the most flexible form of data de-duplication breaks files into *variable-length chunks* using a hash value on a sliding window mechanism. By utilizing techniques such as Rabin fingerprint, chunking can be performed very efficiently [20].

According to [17], the location at which data de-duplication is performed is also very important; if the data are de-duplicated at the client side, then it is known as the *source-based de-duplication* or otherwise, *target-based de-duplication*. In source-based de-duplication, the client will perform hash functions on each data segment that needs to be uploaded. These results are sent to the storage provider to check whether such data are already stored; thus only unique segment will be uploaded and stored. While data de-duplication at the client side can achieve bandwidth savings, unfortunately it is prone to side-channel attack as mentioned earlier. On the other hand, if de-duplication happens at the storage provider, it is no longer prone to side-channel attack, but such solution achieves no decrease in communication overheads. This is a very classic example of a trade-off between data security and system performance.

B. Privacy with De-duplication over Convergent Encryption

Convergent Encryption [17] is used to provide data confidentiality in a de-duplication environment. It uses the cryptographic value of data as the encryption key (Convergent Key), therefore identical data will result in identical ciphertext. In essence, data owner derives a convergent key from the original data and encrypts the data with the key. In this context, users do not have to interact with each other for establishing an agreement on the key to encrypt a given data file, overcoming the problem of key sharing and distribution. Convergent Encryption overcomes the limitation of data confidentiality in de-duplication environment in that ciphertext is now distinguishable, making it suitable for cross user de-duplication [17].

However, this scheme suffers from (1) *Confirmation of File attack (CoF)*, where an attacker who has already known the full plain text of the data, he or she is able to verify if a copy of that file has already been stored. (2) *Learn-the-Remaining-Information (LRI)* attack, where the attackers already owned a big part of the original data, and tried to guess the unknown parts by checking if the result of the encryption matches the observed ciphertext. And lastly, (3) *Dictionary Attack*, an

attacker who is able to guess or predict the original file can easily derive the potential encryption key and verify whether the file is already stored in the cloud storage provider or not.

In [16], the authors proposed to add a secret value to the encryption key, by adding randomness and uniqueness, the key is cannot be easily computable. Which now, de-duplication will thus, can only be performed on files of those users to whom they have a possession of the secret. This solution overcomes the weakness of convergent encryption at the cost of dramatically limiting the effectiveness of data de-duplication.

C. Cloud Storage: Dropbox Security

Dropbox, *Google Drive* and *Sky Drive* are some of the most common cloud storage in the market. Known for their convenience and easy to use interface, Dropbox in particular has garnered more than 300 million users worldwide by May 2014. This is expected to increase within the next 6 months [10]. Dropbox has one of the top network storage services and it has since been providing personal data storage and data sharing among multiple users. [8] has made an in-depth analysis on the types of data sharing method provided by majority of the cloud storage providers. The types of sharing method are as follows: (a) *Public Sharing*, data is intended for the public, so there's no access control. A link to the shared folder (called the sharing URL) can be published, giving anyone on the Internet access to the shared documents. (b) *Secret URL Sharing*, file owner shares the data with others by sending them a sharing URL generated by the cloud storage provider. Anyone with this URL can access the data without further authentication or authorization. The data owner is responsible for identifying the URL receivers. This is only applicable to shared files and not folders. (c) *Private Sharing*, file owner must explicitly specify who can access the shared data. The cloud storage providers then authenticates the identity of the named users, usually by requesting that they sign into their account before accessing the data.

In October 2014, IT news giant, Engadget reported that Dropbox has been under a massive hacks which compromises 7 million Dropbox accounts with their credential leaked online [15]. Despite Dropbox quickly implemented counter-measures to detect the account compromised, they are undeniably a single point of failure in the face of all the sophisticated attacks online. Dropbox does not have control over how the secret URL links are shared after they are generated. This can lead to unauthorized re-sharing of the secret URL link [8]. For instance, owner A sends a secret URL link to user B, B although is not capable of inviting others to view this file, but is capable to resharing this URL to others without the acknowledgement of the file owner. This would mean that if an unauthorized user got their hands on the URL, they can potentially access the shared file with the URL. Convenience of file sharing is performed on the cost of data confidentiality and privacy, therefore there is a need for data encryption and access control to overcome the vulnerabilities of Dropbox.

III. Problem Statement

A. Problem 1: Group Key Management for Data Privacy

When storing files and sharing them among users on Dropbox, unencrypted files stored in the cloud is susceptible to illegal access by the service provider and unauthorized users, thus leading to the compromise of data integrity and confidentiality. This problem could be solved by encrypting all files before uploading to the cloud storage. However, the risk has been delegated to the secrecy of the encryption key instead, as illustrated in Figure 1. The encryption key needs to be securely distributed in order to grant access to the files and we have to protect against malicious attackers obtaining the encryption key distributed by the sender to receivers. Consequently, there is a need for a trusted entity to sit in between the users and cloud storage in order to fulfill the new tasks of key distribution and group access control, specifically targeting at the requirements of secure de-duplication within a group.

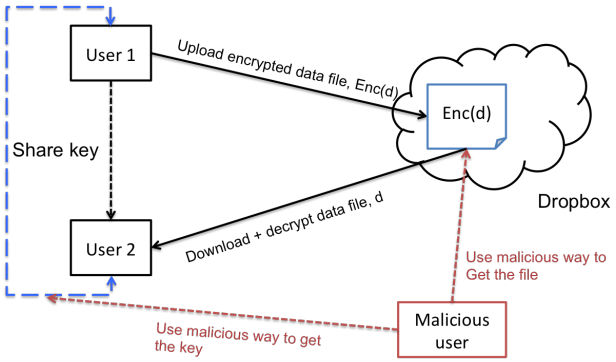

Fig. 1. Cloud Storage security and key distribution problems

B. Problem 2: De-duplication on Privacy Preserved Data

The second problem to address is to detect duplicated data to be uploaded into the same cloud storage without compromising the data privacy, as illustrated in Figure 2. The negative redundant data will take up unnecessary space in the cloud and slow down the network performance. The issue will gradually worsen with the increase of redundant data in the storage. The impact on enterprise users would be more evident compared to a single user.

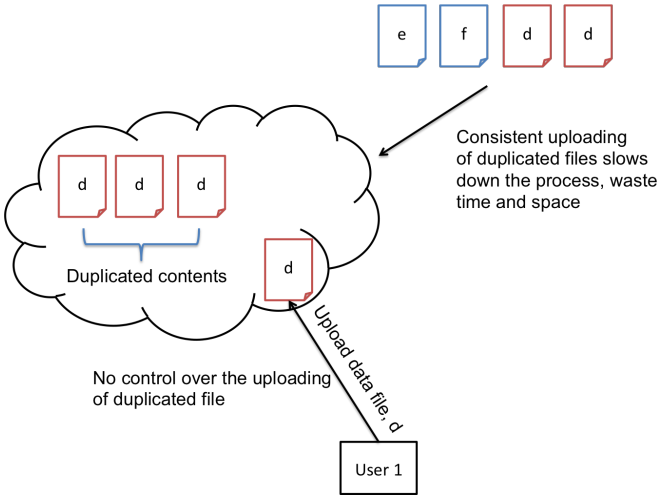

Fig. 2. Redundant and duplicated data problem

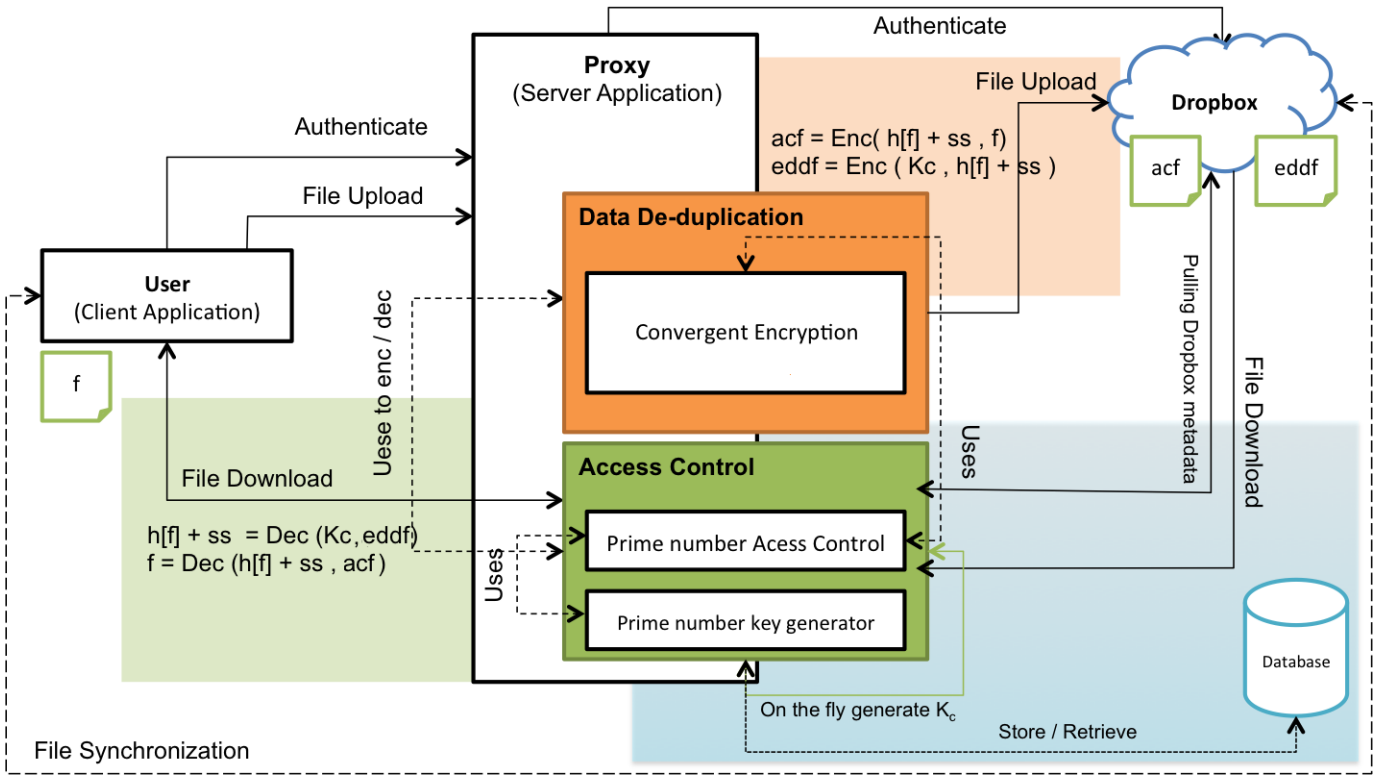

Fig. 3. System Architecture

IV. Proposed Solution: Flexible yet Secure De-duplication Service for Enterprise Data on Cloud Storage

We have designed and integrated our flexible and secure de-duplication service with Dropbox cloud storage. However, the solution is not limited only to Dropbox, it is a genetic security service that is also applicable to other cloud storage providers. As illustrated in Figure 3, our solution consists of three components, namely:

- User/Client application – It provides interfaces to user to operate data upload and download transparently with the function of encryption and decryption.
- Trusted Proxy – It has three main responsibilities, namely: 1) Mediate the communication between the client and the Dropbox Cloud Storage service; 2) Perform group authentication on end users when accessing the shared data on cloud storage; 3) Detect data de-duplication to save storage cost and bandwidth from/to cloud storage.
- Cloud storage server (Dropbox) – It serves as a data store for all data uploaded by the client.

A. System Setup and Assumptions

When enterprises and Internet users turn to cloud based storage as an alternative to on-premises data stores for its powerful data synchronization service, they require convenient file access from multiple users across multiple devices at numerous locations. Although most cloud storage providers support file sharing, it is of utmost important that only authorized users should have access to the original (encrypted) data, whereas unauthorized users should see no hints of the original data. Therefore, it is crucial that a feasible access control mechanism is set in place to manage the access rights and maintain the privacy and integrity of data sharing in the cloud.

The communication channel between the cloud storage and proxy is secured by TLS as required by Dropbox. Additionally, the client application establishes a TLS secure channel with the proxy while uploading or downloading files.

Prior to using this system, users are assumed to have a Dropbox account, and the creation of shared folders and sharing of folders with others are to be performed on the Dropbox website before the trusted proxy can be invoked to generate the corresponding cryptographic materials. Therefore, the system cannot be used to grant access rights to non Dropbox users.

B. Efficient Group Access Control based on Prime Factoring

In order to facilitate secure file sharing, the trusted proxy employs a novel access control mechanism to grant access to the files and shared folders stored in Dropbox based on the hard problem of *prime factoring*. Each authorized user is assigned a prime number, and the group access control key of a file is the multiplication of all the authorized members' prime numbers. Access to the file or folder is granted whenever the group key is divisible by the authorized member's prime number.

Folders and files are viewed as *Resources* in the trusted proxy. Prior to uploading or downloading files, the client application needs to inform the trusted proxy about which resource to manage. Each new resource is associated with a 256-bit

prime number as resource key, denoted as R_k. Similiarly, each member m_i is assigned a 128-bit prime number as member key, denoted as K_{mi}. The control key over group g is denoted as K_{cg} where

$$ControlKeyGen(R_k,K_{mi}) \rightarrow K_{cg} \text{ for } \forall\ m_i \in g$$

The control key, K_{cg} over a resource R, is essentially the *multiplication* of the resource key R_k and the dedicated member keys K_{mi} for which the members have been granted permission to access the resource. The access control service which runs on the proxy checks whether a user is authorized to access the resource by invoking

$$IsMember(K_{cg},\ K_{mi}) \rightarrow \sigma$$

This function checks whether a member, m_i has access rights to the resource managed by the control key, K_{cg}. If K_{cg} is divisible by K_{mi} then member m_i associated with K_{mi} is deemed as authorized, and the output, σ, a boolean value is true, otherwise it is unauthorized with the output of false. Whenever there is a change in membership, such as new member joining or member is revoked from the authorized member list, the group control key K_{cg} is updated through the following:

$$AddMember(K_{cg},\ K_{m_{new}}) \rightarrow K_{cg}^{'},$$
$$\text{where } K_{cg}^{'}=K_{cg}*K_{m_{new}}$$
$$RevokeMember(K_{cg},\ K_{revoked}) \rightarrow K_{cg}^{"},$$
$$\text{where } K_{cg}^{"}=K_{cg}\ /\ K_{m_{revoked}}$$

The control key of a resource, K_{cg} will be re-computed by multiplying the new member key $k_{m_{new}}$ with the old control key K_{cg} when adding a new member; the control key of a resource, K_{cg} is updated by dividing itself by the revoked member's key, $K_{m_{revoked}}$ when permission to access the resource is revoked. Each member is only securely deployed its own member key and the control key is computed on-demand and managed by the trusted proxy. The control key is invisible to the members. More interestingly, the group key update are only relevant to the new member and revoked member, in that the control key generation process is transparent to other members. Existing members do not need to be notified whenever there is a change in the membership. As a result, our scheme provides a very efficient manner to manage and enforce group access control. This control can be further generalized and applied to read and write permissions. Note that we define this group access control key, a big prime number, as a metadata related to the resource. This metadata can be placed in the cloud storage and exists as an object. Due to hard problem of the prime factoring, we can safely assume that its security is difficult to compromise. For applications that require higher level of security guarantee, our system can be adapted to store the membership information in the cloud storage, so that the control key itself is not being stored anywhere but generated in real time on-demand.

For each resource in Dropbox, the trusted proxy obtains the metadata of the resource, and based on the list of members authorized to access the resource, the control key K_{cg} is generated on the fly for each access request to Dropbox. In essence, the control key K_{cg} is used to govern access to the resource in Dropbox.

C. Authentication to Dropbox

The client application as shown in Figure 3 is equivalent to the *Dropbox desktop synchronization application* that allows users to upload, download and share files with other users. It provides an interface for communication between the trusted proxy and the user through the relay of the client's operation back to the proxy. The client application authenticates to the Dropbox via the web browser and delegates its access rights to the proxy to perform access control and data de-duplication. Upon signing in to Dropbox, an access code will be given to the user, submitting the code to proxy will generate an *access token* from Dropbox. This token serves as an identification for subsequent access by the user and proxy. As the *access token* can be used to access the cloud storage, if attackers got hold of it, they will be able to access the files stored in the cloud. Hence, upon logging out from the proxy, this access token will be revoked from Dropbox.

The Dropbox shared folder metadata contains the list of members of the shared folder, the list of files within the shared folders and the owner of the shared folder. As our system uses the Dropbox itself to manage the member list, it is crucial that the proxy and/or the client application do not store any trace of member list. Consequently, whenever the folder metadata is needed, it will be requested and pulled from Dropbox by the proxy so that access control decision can be made in real time to grant access to the cloud storage.

D. Cloud Storage Management and De-Duplication

In addition to the control key, K_{cg} and the resource key, R_k that are being used to enforce access control, a *secret string*, ss for each resource is also generated by the trusted proxy to perform encryption of the resource. When a client application uploads a file to Dropbox, the request is routed to the trusted proxy. A convergent key is generated using the hash value of the resource, $H[f]$ salted with the secret string, *ss*.

$$KeyGen_{CE}(H[f],\ ss) \rightarrow K_{\delta}$$

This convergent key, K_{δ} is used to encrypt the resource. The proposed scheme is different from the conventional convergent encryption in that a *secret string* is added to the hash of the resource in order to add randomness and uniqueness to the key so that the encryption is not deterministic. Hence, *Confirmation-of-File* and *known plaintext attack* can be mitigated in that even if the attacker knew the plaintext and it cannot infer the content of the encrypted files.

The resource is subsequently encrypted with the convergent key, K_{δ} to produce a ciphertext, acf.

$$Encrypt_{CE}(K_{\delta}, f) \rightarrow acf$$

Anyone with possession of the convergent key will be able to decrypt the file. Therefore, the convergent key must be protected and only authorised users are allowed to access the content of the file. The convergent key is subsequently

encrypted using the control key, K_{cg}, and this is stored in the Dropbox as $eddf$.

$$Encrypt_{CE}(K_{cg}, K_{\delta}) \rightarrow eddf$$

Before the resource can be uploaded to the Dropbox, data de-duplication is performed on the encrypted file, i.e., acf. The hash of acf, $H[acf]$ is checked against the database accessible by the trusted proxy to detect whether there is already a copy of the resource in the cloud. If the resource is a duplicate, the upload to cloud will be skipped, otherwise both the encrypted resource, acf and $eddf$ are uploaded to Dropbox for storage.

E. Download of Resource

Access control is realized through division and multiplication of prime numbers. Resource keys and member keys are mapped to the member list obtained from the Dropbox shared folder metadata. The control key, K_{cg} is generated for each access request based on the member list. If it is divisible by the member key, K_{mi} requesting for access, the acf and $eddf$ are downloaded from the Dropbox. If client is deemed unauthorized to view the downloaded content, the download process will be rejected.

Once the user has been authenticated, the control key, K_{cg} can be used to decrypt the convergent key, K_{δ}. Following that, the convergent key is then used to decrypt the resource.

$$Decrypt_{CE}(K_{cg}, eddf) \rightarrow K_{\delta}$$
$$Decrypt_{CE}(K_{\delta}, acf) \rightarrow f$$

The actual requested file is never downloaded until its acf and $eddf$ are decrypted and the client is deemed authorized to view the file content.

V. System Implementation

We have implemented a client application that mimics the *Dropbox* desktop application using Java. In addition, a trusted proxy is implemented and deployed to mediate the communication between the client application and the Cloud Storage, Dropbox. The following sections describe the implementation of the client application and the trusted proxy.

Client Application The client is implemented using Java, and it provides a simple interface for the user to authenticate itself to Dropbox, create a new resource, manage access control, upload resources as well as download resources.

Dropbox Core API The proposed access control and de-duplication service was integrated with Dropbox using its Core API. The core API provides most of the functionalities needed by the trusted proxy, such as authentication of client, retrieval of metadata of files and folders of the client and basic operations to create file, create folders as well as removing them. However, the core API does not allow the proxy to perform the creation of shared folders [9]. The only shared folder related function is to request for its metadata. The metadata of a shared folder however, is returned from Dropbox in JSON format, as there exist no methods to extract the metadata for certain information, unlike of those provided for metadata of a folder. Therefore a class name CURL.java is created to make HTTP request to Dropbox for shared folder

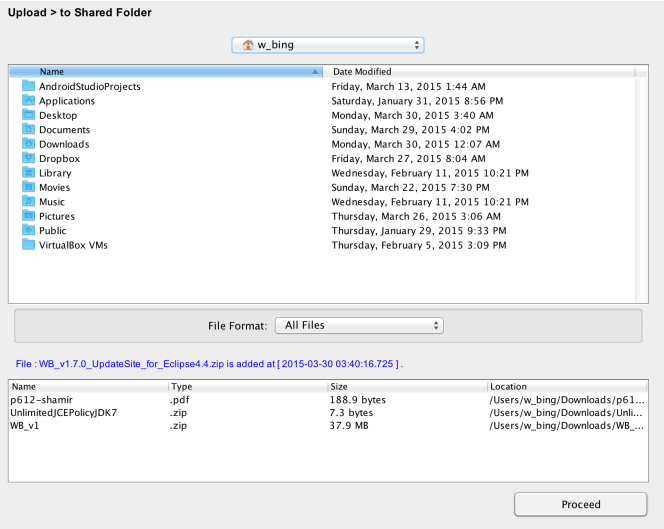

Fig. 4. Interface of the implemented Client Application

metadata as well as extracting information such as owner of the shared folder and members in the shared folder out of the JSON string returned from Dropbox [12].

Cryptography and Access Control We implemented convergent encryption and cryptographic hashing using the Bouncy Castle API for Java [6]. Encryption was based on 256-bit AES, and SHA-256 was used as the hashing function. Convergent encryption was achieved by implementing AES-256 with a static IV. The IV is derived from the first 16 byte of a secret string, the same secret string used to generate the convergent key.

The *Prime Number Access Control* was implemented on the trusted proxy. This was achieved through the use prime number multiplication and division capability in Java. As the prime numbers generated were really big, a BigInteger data type was used.

VI. Evaluation and Results

A. System Performance and Overhead

In terms of performance, we measured the time taken to *upload* and *download* files of variable size to determine the performance overhead incurred by the trusted proxy. We compared the total data transfer time from access control, data encryption/decryption, de-duplication to upload/download with our proxy solution against the existing Dropbox solution. The experiment results are listed in Table I and Table II. Obviously, our solution is slower than the current Dropbox solution for upload as shown in Table I, but the difference is narrowed down with the increase of file size. More importantly, the overhead of 4 sec to 1 min is consumed by the security features provided including confidentiality, access control and secure de-duplication. On the contrary, the performance of our solution is slightly better for resource download. As shown in Table II, the performance between our system and pure Dropbox solution is comparable. This is possibly because the amount of computational workload has been reduced in that data de-duplication check is only applicable to resource upload, while resource download only involves access control

check and decryption. In fact, when performing data de-duplication, more cryptographic functions such as hashing had to be carried out, thus this might have slowed down the overall performance. While for access control, most of the operations are simple mathematical division and multiplication.

File Size	With Proxy	Without Proxy
180 Kb pdf	approx. 7 secs	<3 secs
6.5Mb Audio File	approx 45 secs	<20 secs
40 Mb Video clip	2 mins	approx 1 - 2 mins
>230 Mb Zip file	5 mins	4 mins

TABLE I. RESULTS OF UPLOAD SPEED

File Size	With Proxy	Without Proxy
180 Kb pdf	approx 4 secs	1 - 2 secs
6.5Mb Audio File	approx 30 secs	<20 secs
40 Mb Video clip	2 mins	1 mins
>230 Mb Zip file	approx 7 mins	7 mins

TABLE II. RESULTS OF DOWNLOAD SPEED

An increase in uploading or download time per file will be significant if the client were to be an enterprise user. Processing more files drastically increases the processing time thus might not be feasible. But our solution is scalable, we can add more proxy nodes to perform access control and secure de-duplication because of the secure metadata storage on cloud and on-demand computation of the control key.

B. Storage Space Overhead

By implementing data de-duplication, we can potentially save up a lot of storage space, but concern arises when we are also uploading the access control files into Dropbox. One encrypted data file comes with one access control file, the number of access control files will increase linearly with the number of encrypted data. Table III compares the access control file size against the actual encrypted data file size. The size of the encrypted data has no relevance to the file size of an access control file. This is due to the fact that each access control file stores the decryption key, and the decryption key are of the same size due to the algorithm used. Taking into consideration that the access control file safeguards the integrity of the data, the space taken up by access control files is actually acceptable. But this remains arguable, as a mitigation to this problem is to store the access control file on the trusted proxy, and leave the responsibilities to the proxy, but this can potentially affect the overall system performance.

File Size	Access control file size
180 Kb pdf	64 Bytes
6.5Mb Audio File	64 Bytes
40 Mb Video clip	64 Bytes
>230 Mb Zip file	64 Bytes

TABLE III. COMPARISON OF ACCESS CONTROL FILE SIZES

VII. DISCUSSION AND SECURITY ANALYSIS

This section presents a security analysis of the proposed system and describes the corresponding mitigation strategy.

A. Compromise of File Content

As the proposed system has adapted the use of Convergent Encryption, even if the attacker has knowledge of the file content stored in Dropbox and wishes to launch *Confirmation-of-File attack* (CoF), he or she is not able to do so because all resources in our system are encrypted with its hash value concatenated with a *secret string*, and encoded as K_δ. Thus, without knowning the *secret string*, it is not possible to initiate a CoF attack on the system.

Additionally, the cloud storage is not directly accessible by a client of the system, instead they must first be decrypted by the trusted proxy. The generation of the convergence key is hence performed at the trusted proxy, the clients will not be able to have access to the shared *secret string*. In order to safeguard the *secret string*, other security means such as *threshold cryptography* [19] can be employed to further split the *secret string* into n shares and stored in distributed databases. With this, the *secret string* can only be constructed on-demand when $k-1$ out of n shares needs be gathered, and the attackers have to compromise $k-1$ shares.

B. Compromise of Dropbox Access Code

Upon launching the client application, one will be prompted to request for access from Dropbox. Following that, a line of *access code* provided by Dropbox will need to be entered into the system. This line of code, although is exposed to the public, it can be a threat to the system on its own. This *access code* when copied into the application, Dropbox will be able to authenticate the client and then return the *access token* to the client. To make this API call for authentication, the access code is needed, the access code however is *not* the access token, and it serves a totally different function. Therefore even the access code is compromised, Dropbox knows that this access code is tag to which account, and it will expire after a period of time. Essentially, this access code cannot derive any access token. The only solution to this problem is to ensure that the user is not susceptible to *Shoulder Surfing*, making sure that no one else in the vicinity can gain access to the access code.

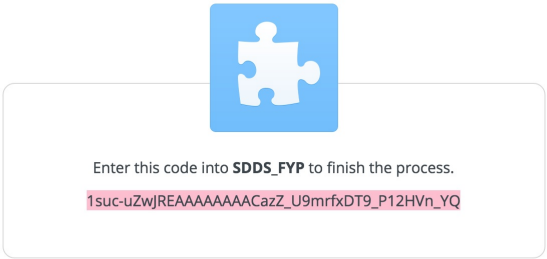

Fig. 5. Requesting access code from Dropbox

C. Compromise of Dropbox Access Token

The access token is crucial for the trusted proxy to communicate with Dropbox, and Dropbox identifies the client through this access token. If the access token is compromised, it will endanger the proxy as well as all the data in the client's Dropbox. The system is unable to deal with compromised token, however we can prevent the compromise from taking place. The access token can only be gained through API calls

made to Dropbox, the entire communication made from proxy to Dropbox is secured using *Transport Layer Security, (TLS)* provided by Dropbox. Upon client exiting the application, and when the session has ended, the application must request the trusted proxy to revoke this access token from Dropbox. Consequently, revoked tokens will no longer be usable after the session has ended.

D. Old and Outdated Access Control Files

Access control files are re-generated upon each request of adding a new member and revoking the membership of current members. As these access control files are automatically synchronized between the client application and the cloud storage, the client can attempt to constantly back-up all the access control files. Whenever a user has his membership revoked from accessing a shared file, although it still holds onto the outdated access control file, its request to decrypt the file will be denied. This is because the trusted proxy would have updated the control key and hence the client will not be able to decrypt the old access control file.

E. Compromise of Convergent Encryption Key

In all cryptographic system, the encryption key is the secret to the file content. The trusted proxy can only prevent encryption key from getting compromised but cannot deal with a compromised key. The encryption key is generated with the file hash value salted with a secret string, and then encrypted with a control key. The trusted proxy employs multi-layer security to protect the convergent encryption key. Therefore, we assume that it is sufficiently difficult to compromise the key. However under any circumstance that the key is compromised, the client will have to remove all files from the shared folder and inform the proxy administrator to mitigate further loss.

VIII. Conclusions and Future Work

This paper has proposed a flexible yet secure data De-duplication service for enterprise tenants to utilize cloud storage in a secure and economical means. With the security concerns, we have proposed a lightweight access control mechanism based on prime factoring hard problem to provide a flexible group control; we have also devised and implemented a practical secure de-duplicaiton method to allow de-duplication to occur on encrypted data within a group, which can effectively tolerate the existing attacks of CoF, LRI and Dictionary attacks. With regards to the concerns of operating costs, our secure de-duplicaiton solution can efficiently save storage capacity on the cloud storage and compare well with the existing solutions. Furthermore, our solution is scalable because of its metadata storage on the cloud and on-demand control key computation. Nonetheless, the performance of the proposed system can be scaled up by deploying multiple proxy nodes, and this is particularly useful for meeting the requirements of the enterprise tenants, where it can better support performance oriented applications for large scale enterprises.

Our prototype on Dropbox has demonstrated the feasibility of combining de-duplication and access control to secure cloud storage. Although the performance penalty for uploading of resources through a single trusted proxy is significantly higher as compared to the existing Dropbox solution, we believe that with multi proxies and multi-threading, the performance can be further improved. Finally, we will further explore the possibility of delegating some tasks on the proxy to the clients without compromising the security in our future work.

References

[1] Amazon Simple Storage Service, http://aws.amazon.com/s3, 2015.

[2] Bitcasa, https://www.bitcasa.com, 2015.

[3] Dropbox, http://www.dropbox.com, 2015.

[4] Google drive, www.google.com/drive, 2015.

[5] iCloud, https://www.icloud.com, 2015.

[6] BouncyCastle. The Legion of the Bouncy Castle. https://www.bouncycastle.org/, 2015.

[7] R. Castagna. Survey finds cloud storage implementation growing but cautious, mar 2013.

[8] C.-K. Chu, W.-T. Zhu, J. Han, J. Liu, J. Xu, and J. Zhou. Security concerns in popular cloud storage services. *Pervasive Computing, IEEE*, 12(4):50–57, Oct 2013.

[9] Dropbox. Core api. https://www.dropbox.com/developers/core, 2015.

[10] K. Hong. Dropbox reaches 300m users, adding on 100m users in just six months. http://thenextweb.com/insider/2014/05/29/dropbox-reaches-300m-users-adding-100m-users-just-six-months/, may 2014.

[11] J. Li. Secure deduplication with efficient and reliable convergent key management. *IEEE Transaction On Parallel And Distributed System*, 25(6):1615–1625, jun 2014.

[12] S. Marx. New in beta: shared folder metadata. https://blogs.dropbox.com/developers/2014/07/new-in-beta-shared-folder-metadata/, 2015.

[13] Megaupload. Megaupload.

[14] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. *Trans. Storage*, 7(4):14:1–14:20, Feb. 2012.

[15] M. Moon. Dropbox passwords posted online and millions more might follow. http://www.engadget.com/2014/10/14/dropbox-log-in-posted-online/?ncid=rss_truncated&utm_source=twitterfeed&utm_medium=twitter, oct 2014.

[16] D. Perttula. Drew perttula and attacks on convergent encryption. https://tahoe-lafs.org/hacktahoelafs/drew_perttula.html, mar 2008.

[17] P. Puzio. Cloudedup : Secure deduplication with encrypted data for cloud storage. *2013 IEEE International Conference on Cloud Computing Technology and Science*, 978-0-7695-5095-4(13):363–370, 2013.

[18] F. Rashid. A secure data deduplication framework for cloud environments. *2012 Tenth Annual International Conference on Privacy, Security and Trust*, 978-1-4673-2326-0(12):81–87, 2012.

[19] A. Shamir. How to share a secret. *Communication of the ACM*, 22(11):612–613, nov 1979.

[20] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller. Secure data deduplication. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 1–10, New York, NY, USA, 2008. ACM.