Secure Dropbox

by

Gu Juntao, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

August 2013

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Gu Juntao

August 21, 2013

Permission to Lend and/or Copy

Acknowledgments

I would like to show my greatest appreciation to my supervisor, Dr. Hitesh Tewari, for his suggestions and expertise throughout the project. Also, I would like to thank my parents, Mr. Peiqi Gu and Mrs. Xuelan Yang, for their unwavering love and encouragement. Finally, I would like to thank my best friend and love, Zhong Miao, for all of her support.

Gu Juntao

University of Dublin, Trinity College

August 2013

Secure Dropbox

Gu Juntao, M.Sc.
University of Dublin, Trinity College, 2013

Supervisor: Tewari Hitesh

Recently there have been increasing requirements about utilizing cloud technology for data maintaining and management especially when reliable, stable data storage and remote data access required. Since the cloud storage service is usually designed to be available to users over the Internet, it essentially facilitates the data sharing as well. However, security concerns are growing as the most commonly cited reason why users, particularly those enterprise users who are information confidentiality critical, are not interested in SaaS, of which cloud storage services like Dropbox or Google Drive are typical instances. The use of those cloud storage service is actually exposing the sensitive data to the service vendors.

This research project aims to investigate the feasibility of solving the data exposing issue and achieving the goal of secure data sharing via encryption approaches. It was decided to develop an application to explore this topic. A client end encryption tool would be implemented based on Python and Python accredited cryptography related modules. What is more, a key management system (KMS) with features of file encryption management, user profile management and sharing information management would be deployed to realize the function of everywhere use and secure data sharing. Dropbox would be selected as the instance of SaaS cloud storage service for demonstration purpose so that Dropbox core API would be used.

The result of the project indicated the idea of the hybrid application of symmetric and asymmetric cryptography algorithm could effectively protect the data stored on Dropbox and potentially could be implemented. The user testing session at the end of the project also strongly indicated that the encryption tool Secure Dropbox could tremendously gain users confidence in terms of storing their confidential data on a cloud storage service like Dropbox.

Contents

Acknowledgments **iv**

Abstract **v**

List of Tables **xi**

List of Figures **xii**

Chapter 1 Introduction **1**

1.1 Motivation 3

1.2 This Report 5

Chapter 2 State of the Art **6**

2.1 Introduction 6

2.2 Cryptography 7

2.2.1 User-level Cryptography 7

2.2.2 System-level Cryptography 7

2.3 Secure Storage in the Cloud 8

2.3.1 Security in Dropbox 9

2.3.2 Security in Google Drive 9

2.3.3 Storage Security Risks in the Cloud 10

2.4 Searchable Symmetric Encryption 11

2.5 FileMap 12

Chapter 3 Design **14**

3.1 Application of Cryptography 16

3.2 Key Management Service 18

3.2.1 KMS User Authentication 19

3.2.2 KMS RSA Key Pair Management 20

3.2.3 KMS Doc Key Management 20

3.2.4 KMS Sharing Information Management 20

3.2.5 Local KMS 21

3.3 Right Cryptography Algorithm 21

3.3.1 Symmetric Cryptography 21

3.3.2 Asymmetric Cryptography 22

3.3.3 Cryptographic Hash Function 24

3.4 Integration with Dropbox 26

3.5 C/S or B/S 28

Chapter 4 Implementation **30**

4.1 Implementation Dependencies 30

4.1.1 Secure Dropbox Client 30

4.1.2 Secure Dropbox KMS 32

4.2 Communication 35

4.3 Secure Dropbox KMS Implementation 37

4.3.1 Database Design 38

4.3.2 KMS User Management 39

4.3.3 KMS Key Management 40

4.3.4 KMS File Sharing Management 41

4.4 Secure Dropbox Client Implementation 42

4.4.1 Secure Dropbox UI 43

4.4.2 Dropbox Handler 46

4.4.3 Cryptor Handler . . . 48
4.4.4 KMS Handler . . . 49
4.4.5 Configuration Module . . . 50
4.4.6 Secure Dropbox Module . . . 50

Chapter 5 Analysis **53**
5.1 Evaluation Approaches . . . 53
5.2 Performance Testing . . . 54
5.2.1 Testing Schema . . . 54
5.2.2 Performance Evaluation . . . 54
5.3 User Experience . . . 55
5.3.1 Session Procedure . . . 55
5.3.2 None IT Background Users Feedback . . . 56
5.3.3 IT Background Users Feedback . . . 57
5.4 Discussion of Feedback . . . 58

Chapter 6 Conclusion **60**
6.1 Results . . . 60
6.1.1 Was An Effective User End Encryption Tool Developed? . . . 60
6.2 Criticism . . . 61
6.2.1 Design Criticism . . . 61
6.2.2 Implementation Criticism . . . 61
6.2.3 Testing Criticism . . . 62
6.3 Future Work . . . 62
6.3.1 Version Control and File Recovering . . . 62
6.3.2 User Profile Management . . . 62
6.3.3 Improvement of Sharing Mechanism . . . 63
6.3.4 File Sharing Refreshing . . . 63
6.3.5 File System Level Encryption . . . 63

6.3.6 Configuration Interface 64

6.3.7 Multi-platform Implementation 64

6.3.8 Better Local KMS Mechanism 64

Appendix A Implementation Module Dependence **65**

Appendix B Task Description **66**

Appendix C Questionnaire **68**

Bibliography **70**

List of Tables

List of Figures

2.1 Customer Architecture . . . 12

3.1 Secure Dropbox Architecture . . . 15

3.2 Encrypted File Storage and Reading . . . 17

3.3 Encrypted File Sharing . . . 18

3.4 KMS Architecture . . . 19

3.5 Strength Equivalence . . . 23

3.6 Hash Table . . . 24

3.7 Hash Table with Parameters . . . 25

3.8 Dropbox Core API . . . 27

3.9 Cryptography Deployment . . . 29

4.1 Path Representation Schema . . . 32

4.2 Platform Independent Coding Style . . . 32

4.3 Amazon EC2 Server Monitoring . . . 33

4.4 Amazon EC2 Security Customization . . . 34

4.5 Amazon EC2 SSH Procedure . . . 35

4.6 Restful WebService Programming . . . 36

4.7 HTTP POST Request Generating . . . 37

4.8 KMS Interfaces . . . 38

4.9 KMS Database Design . . . 39

4.10 Hashed Passwords . . . 40

4.11 Sharing Notification 41
4.12 Sharing Notification Implementation 41
4.13 Secure Dropbox Client Architecture 43
4.14 Running Environment Detection 44
4.15 Secure Dropbox Commands 45
4.16 Command Print Information 45
4.17 Command Print Information II 46
4.18 File Dialog Implementation 46
4.19 Application Settings of Secure Dropbox 47
4.20 Dropbox OAuth Service 48
4.21 Return Value of Dropbox Core API 48
4.22 AES Key Generation 49
4.23 KMS Data Package Generation 49
4.24 User Instance in Local KMS 51
4.25 RSA Key Pair Generation 52
4.26 Replacements of Newline Characters 52

5.1 Testing Environment 54
5.2 Secure Dropbox Cryptography Performances 55

Chapter 1

Introduction

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. It is by definition composed of three service models:

- Cloud Software as a Service (SaaS) provides computation capacity by running software on a cloud infrastructure. These applications are remotely accessible to users. The users are not supposed to manage the cloud infrastructure hardware but only utilize the services. Different cloud consumers' applications are organized in a single logical environment in the SaaS cloud to achieve economies of scale and optimization regarding issues like speed, security, availability, disaster recovery, and maintenance [1] [2]. Those cloud storage services which this research mainly aims at, Dropbox and Google Drive for instance, are representative examples of SaaS.
- Cloud Platform as a Service (PaaS) is a development framework and service hosting platform which allows users to develop cloud services by providing runtime environment, R&D toolkits, SaaS application APIs, storage and middleware/OS support. An example instance of PaaS is the Google Application Engine.
- Cloud Infrastructure as a Service (IaaS) refers to on-demand provisioning of in-

frastructural resources on the cloud, usually in terms of VMs [3]. The capability provided to the consumer is about provision processing, storage management, network configurations and other fundamental computing resources. The customers are able to deploy and run any software in the cloud [1] [3]. Amazon EC2 is a representational IaaS cloud platform.

Cloud storage refers to the service on the cloud which is delivered as remote storage infrastructure. It can be accessed from any terminal connected to the Internet. It also offers additional data-related functionalities regarding to data management and maintenance [4]. Any operations in which the core tasks that a cloud computing system processes are relevant to big data storage and maintenance, it could be defined as a cloud storage system. Most hardware configuration and software deployment inside are optimized aiming for providing data storage service. Just as it is a service that provides interfaces in terms of data manipulations rather than utilizing certain physical storage devices directly, cloud storage is by nature an application of SaaS.

For personal users, cloud storage is an extension of local storage with good cost performance. Also it could be an ideal replacement of portable storage devices like flash drive or portable hard drive by which stable data storage is not guaranteed. Moreover, it is also a more reliable local disk backup given its essence of good cost efficiency, remote accessibility and reliability. Besides, from the enterprise-level users perspective, cloud storage reduces the investment they should have made in terms of design, management and maintenance of device clusters to build a systematic big data storage service. Lastly, the data sharing is another thing that boosts up data usage efficiency via facilitating the real time data exchange and decreasing the workload about data collection especially relating to frequently reused data resources.

As a highly developed commercial-level cloud storage product and innovator in this field, Dropbox is becoming the industry leader with a market share of 14.14% in 2011 [5]. Google Drive, another popular cloud storage service which is built based on application framework by Google and well integrated with other Google services (e.g. Gmail as a major way of acting user or system activity notification) is also increasing dramatically. It is routine for cloud storage service vendors to provide multiple ways of getting access to the service from different platforms for ubiquitous usage. For example, desktop client, web application and portable device application.

1.1 Motivation

Alongside the high speed development of the cloud storage industry, security anxiety in the age of the cloud is becoming a significant issue. There are frequent reports of web servers being attacked: either because data residing on servers is of interest to hackers, or because the hackers want to leave their mark of exploiting. While robustness and availability are main concerns of security, confidentiality is equally important [6]. Data storage security problem is an important aspect of Quality of Service(QoS) [7].The security features of cloud storage technology are always compared with traditional ways like local disk or RAID. For personal users, the reasons for using the cloud storage service or not are simple and straight forward. For instance, would people put their security sensitive data like bank account information on the cloud? Also, even if service carriers claim that they deploy secure encryption scheme throughout, is it definitely sure that the employees of service carriers will abide strictly by the terms of service and never try to access users data unconsciously by technical approaches which could be easily performed internally? In addition, some personal users with computer expertise could have concerns about whether the continuous on service cloud storage server might represent an attractive attack target and therefore be riskier than local storage. Lastly, personal users, particularly those who use cloud storage service for free, have concerns about who will bear responsibility for any loss of personal data whilst it is stored on the cloud. Such problems relating to security confidence restrict users to only storing unimportant data on the cloud as redundancy and utilize the feature of portability. Even if the cloud storage service provider could be

Google or Dropbox, it is paradoxical that people only achieve redundancy for important data because the security concerns hinder it.

For enterprise users, except for the same concerns that might be taken into consideration as a personal user, there are more severe problems to be dealt with. According to the survey hosted by Intel in May 2012 [8], the concerns of IT professionals regarding public cloud (opposite to the private cloud and traditional IT storage solutions like local disk or RAID. The storage service provided by Google Drive and Dropbox discussed in the paper are typical instances of public cloud) were: Firstly, the lack of measurement of security capabilities of cloud storage service. Despite the security methods deployed which claim to be secure, the providers hide the implementation detail and the only transparent approach to users is simply identity based access control. It is not sufficient to gain their confidence towards the security of the storage system. 57% of the survey participants thought it was the most significant problem concerned when using such service among all the concerns. 55% of the participants considered a lack of control over data as another major concern because of the invisible abstracted resources and shared storage infrastructure in the cloud. Only 36% of the participants thought the cloud storage service lack of transparency/ability to conduct audits which indicates that the continuously improving of comprehensive query interfaces about storage service itself makes it acceptable by gaining the level of transparency. Compliance with regulatory mandates is another key problem concerned by an average of 78% participants. Due to the opaque security and operability of cloud storage, the security of storing legislation sensitive data might not be fully measured and leads to compliance problems. Such laws vary from country to country. One of the advised solutions is making full use of cloud storage for enterprise data is to classify the data with different priorities and keep them in internal infrastructure and external storage services respectively.

The goal of the project is to investigate the feasibility of gaining users confidence towards the security of cloud storage service by utilization of a hybrid encryption mechanism in the local host. Also to evaluate the advantages of information sharing based on asymmetric cryptology over the symmetric cryptology. To achieve this goal, it was decided to develop a desktop client which performs local file management and cryptology operations and a server, which functions as a key management service, user management system and a file sharing pool. The combination of the two components will implement the entire procedure as a prototype.

On completion of the project, the production will be demonstrated to information security experts and several testers who do not have Computer Science background will be invited. The evaluation will be mainly about performing user acceptance tests for assessing the availability and user experience of the software.

1.2 This Report

This report will contain a review of the state of the art technologies in the fields of cloud storage, cloud security approaches and other potential solutions those are relevant to this project. Some other popular approaches will also be discussed, together with justification for their inclusion.

Important design decisions made during the undertakings of this course project will then be explained in detail, along with the reasons that justify these decisions. There will be followed by a detailed outline of both design and implementation of key components of the software. A systematic analysis of the software prototype will be given based on not only the expertise of the IT security professionals but also comments and suggestions by routine Dropbox users. Assessment of this application will focus on the theoretical correctness, availability and user experience based on the feedback from beta users. Additionally, potential future works including implementing such an application on portable platform and evaluating the possibility of performing a file system level encryption will be discussed. Finally, the conclusion of the whole project will be presented.

Chapter 2

State of the Art

2.1 Introduction

Information Security is one of the key issues that slows down the development of cloud computing and its feature of complications with data privacy and protection continue to hinder taking the market [9]. Although the cloud storage services potentially offer several security advantages with their standardized accessing interfaces and benefits of scale (i.e., the same investment of security infrastructures on centralized server sets would provide better capability with regard to issues like data filtering, update management and deployment of information security policies) and agility of reactions to attacking events or other security-threatening behaviors. However, in some other ways, due to its essence of the multi-tenancy model [2], problems are caused by sharing physical storage resources with other users. Also its nature of SaaS implies merely the accessing and auditing interfaces are open to users makes the hidden internal mechanism not trustworthy to users. Furthermore, security approaches towards cloud storage must consider interactions and mutual effects among multiple network objects (e.g., other applications or users) [10] that lead to the invalidation or rejection of traditional local secure storage methods. Modern security technologies and some mainstream cloud storage security solutions in particular are considered as referable potential solutions to this problem. Their features, impacts and also reasons why some of these security approaches are not appropriate in certain application scenario will be discussed in this chapter.

2.2 Cryptography

Cryptography provided secrecy for information sent over channels where eavesdropping and message interception was possible [11]. In order to conceal sensitive contents, cryptography is always exploited on plain text in the storage media concerning not only those intelligible contents but also metadata that might be utilized for deducing the corresponding contents.

2.2.1 User-level Cryptography

A typical way of encrypting the file or data is through some encryption tools, such as enigma under FreeBSD, crypt in UNIX or applications with encryption modules embedded. The basic goal of information concealing achieved in this way if used properly. However, none of them is entirely satisfying in the matter of security, generality and availability [3]. Manual errors like forgetting to delete the plain text file after encryption or inappropriate key management will make results far from expectation. As a conclusion, if encryption is too close to the user level, the high frequency of human interaction caused errors is not worth the cumber for practical and everyday usage [12].

2.2.2 System-level Cryptography

Generally to avoid the manual flaws of user level encryption, the cryptographic related modules should be designed to serve as infrastructures of the system. The key of designing this system is mainly about specifying the component to be cryptographically protected in accordance with priority level. Storage and communication risks are most urgent factors to be concerned about.

Physical media like hard drives could be well protected by customizing the hardware design or additional firmware functionalities. Disk controllers could be used for encrypting entire disks or individually ciphering file blocks with a specified key. In this way the procedure of encryption is completely transparent as long as the key is determined to the hardware [12]. It seems to be an ideal alternative of user-level cryptography but the problem happens when performing data management like sharing or backup. Data sharing would not be done until the encryption key exchange has been made via unreliable medium. Playing backup of a disk without encrypting raw contents is equivalent to exposing a decrypted version of ciphers while applying same encryption mechanism gives rise to not only extra expense of backup procedure but other unreliability concerns. For example, backup is periodically played to ensure the availability by creating data redundancy. However, an extra cryptography application will make the availability of cryptography modules a necessary condition prior to the availability of content backup itself, which changes the essence of backup drastically.

Moreover, such mechanism would not protect data commuting with the disk so that it is not sufficient to secure data in remote physical storage entities. Encrypted network connections between storage entities could be a solution for securing the data exchange. The cryptography could be utilized in the form of end-to-end encryption communication protocol and cryptographic authentication [12]. However, some specialized hardware might be required and these extra cryptographic operations will doubtlessly cause a significant penalty of network performance.

2.3 Secure Storage in the Cloud

File systems on the cloud, due to their web application circumstance, lead to extra security concerns in comparison to local file system. Except for enhancement of its service availability, features like constant monitoring and auditing mechanism will be well redesigned and integrated into the system to achieve better security performance [13]. Most service providers offer access control mechanism as a basic security issue. Some service carriers (e.g., Dropbox) use a combination of identity based access control and encrypted data storage while others (e.g., Google Drive) just stick to concentrating on better access control, auditing performance and revolutionary redesign of the traditional file system, Google File System for instance. Also, certain modern solution like two-step verification which refers authentication security to additional verification steps via text, voice call or other similar approaches. It adds an extra layer for secure storage thanks to carriers multi-platform features.

2.3.1 Security in Dropbox

Dropbox uses Amazon's Simple Storage Service (S3) as storage infrastructure. All files stored online by Dropbox are encrypted during transmission and before storage with Secure Sockets Layer (SSL) and AES-256 bit encryption respectively [14]. Amazon S3 does not encrypt data before storage but a service named Server Side Encryption (SSE) is provided for users to encrypt files and perform key management on the cloud.

2.3.2 Security in Google Drive

Without using any local encryption on the physical storage infrastructure, Google improves its file system dramatically about the access control and auditing. In Google File System, there are some innovative specific fields as part of file metadata for multi-user management and data sharing. For example, rather than generating physical replicas of file or file relocation record, the field "exportLinks" allows data sharing by generating a sharing URL which is irreversible and ready to be published. Another field "writersCanShare" indicates the writers permission of sharing the file with others, which actually controls the permission leakage.

Google Drive also offers another access control mechanism called authorization scope. Authentication scopes imply the permissions users are required to authorize for the following operations. For example, when requiring for authorization with CGI: /auth/drive.readonly indicates that a positive authentication result will allow read-only access to file metadata and file content.

2.3.3 Storage Security Risks in the Cloud

Privacy concerns for some internal reasons always exists as long as the content is stored in plain text on the cloud or encrypted with a key that is known to someone who conducts the encryption. Dropbox has been criticized for storing authentication information on disk in plain text. It actually indicates that Dropbox's terms of service are contradicted to its privacy implementation although the company claims that employees of Dropbox are not able to access users files or profiles [15]. On the contrary, employees of Dropbox can effortlessly hack the system and access users data unconsciously [16]. This kind of attacks cannot be protected by traditional security technologies as long as it is technically possible [17].

External attackers would also try to hack the storage service providers system to access to stored data and such attacking could be prevented by traditional approaches [18]. Undoubtedly, it will raise the data security risk if stored plain texts are exposed once the system has been exploited. Given the fact that these day and night services with considerable potential profit are more attractive to attackers, they are essentially more information secure risky. Some services claim that the stored data is all well encrypted but tragedy happens when key management are played in the same bucket and easily to be discovered by hackers.

Anyway, storage on the cloud in plain text, no matter how unbeatable the security design and implementation are, is highly risky no matter it is facing an internal or external attack. Customer data is uncontrolled and could be leaked to unaccredited parties potentially in this way [18].

2.4 Searchable Symmetric Encryption

The security of encrypted data storage, assuming the private key could be kept properly and confidentially, could be guaranteed. However, there could be an availability performance penalty that routine private key encryptions might prevent searching over encrypted data. Users would lose the ability to selectively locate fragments of expected data at the same time [19] [20]. Searchable symmetric encryption (SSE) allows user to use the third party storage service of its data, meanwhile keeping the capacity of selective searching over data pools [21].

The system prototype was described as a possible architecture for a cryptographic storage service in the paper from Microsoft [21]. The system was composed of three parts: a data processor (DP), a data verifier (DV) and a token generator (TG). The data processor handles data prior to being sent to the cloud; Data verifier verifies if the data in the cloud has been modified without authorization based on auditing information; Token generator generates tokens that grant the permissions to the cloud storage provider and enables it to get segments of expected data on customer. Additionally, a credential generator who works based on certain access control mechanism is implemented. It works by issuing credentials to the various parties in the system which will enable them to decrypt encrypted files according to the policy.

The key features of a cryptography based storage service let customers take control of their data. Its security properties are derived from cryptography, a trusted issue but not those unreliable human factors like legislation or physical security. The advantage with regard to security make the data is always encrypted and data integrity can be audited any time so that security attacking poses little, or say, no risk for the customer [19]. Furthermore, the utilization of symmetric searchable encryption, which leaves the indexes available but sensitive contents encrypted, improves the availability of encryption based security storage.

The whole process illustrated in this paper is expressed as follows:

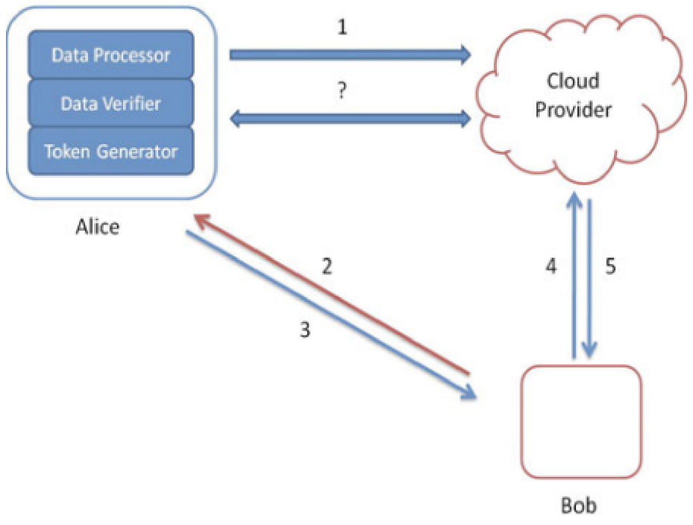


Figure 2.1: Customer Architecture [19]

1. Alices data processor prepares the data (encrypted with symmetric searchable encryption) before sending it to the cloud.

2. Bob asks Alice for permission to search for a keyword.

3. Alices token and credential generators send a token for the keyword and a credential back to Bob.

4. Bob sends the token to the cloud.

5. The cloud uses the token to find the appropriate encrypted documents and returns them to Bob.

6. At any point in time, Alices data verifier can verify the integrity of the data.

2.5 FileMap

FileMap is a file-based map-reduce system for data-parallel computation [22]. It is designed and implemented around several application scenarios like file-based and data replication. Its idea of data replication mechanism which is similar to RAID4 or RAID5 means the file is split in the physical separated media and stored. Files could be split into segments and stored in different entities which are both logically or physically separated, or a combination of them. For example, to perform secure cloud storage in logical separated entities, certain file could be fragmented into n segments and stored in one providers cloud storage service with n identities (i.e., n segments stored in n accounts of Dropbox respectively). While storing files in physical separated entities, as same as the first scenario, files also have to be divided into n segments but the difference is storing them via different cloud storage providers services (i.e., n segments stored in Google Drive, Dropbox and SkyDrive separately). Given the fact that some providers providing encrypted storage service like Dropbox, a pseudo searchable encrypted cloud storage prototype with could be made by:

1. Regenerate the file and adding indexes information in the metadata and fragment it into index and content segments.

2. Store the index segments in plain text stored cloud storage service but with better access control mechanism deployed like Google Drive.

3. Store the rest n content segments in n accounts of encrypted cloud storage service like Dropbox.

Chapter 3

Design

Current storage systems secure data either via encrypting data on the wire, or through data encryption on the physical disk [23]. Cryptographic protection of single file instances could prevent data modification or leakage during storage [24]. Secure Dropbox is designed as a client end encryption tool over the Dropbox service. The main idea is providing a file encryption service with symmetric cryptography where key is not known to Dropbox and, in the meantime, the file encryption key is well protected with asymmetric cryptography for secure file sharing. It is proposed to strengthen the way in which the Dropbox protects users file content security from attacks especially those from internal Dropbox. Internal attack is theoretically easy to be performed as Dropbox encrypts those files with keys known to them. Also the security against external attack has been fortified since files will be encrypted twice, Dropbox and Secure Dropbox respectively. Secure Dropbox is proposed to be a C/S architecture system. The server end will work as a key management service (KMS) which mainly processes key management requests and storage request while the client end running on a user's host computer will perform cryptographic computations which are resource consuming. Secure Dropbox performs file operations like uploading, downloading and sharing via Dropbox API or the Dropbox official client which is essentially built with Dropbox API as well. The user interface of Secure Dropbox will be designed as a file system operation interface. Encryption within this service will be transparent to the users. The very security of Secure Dropbox service is based on users proper usage of Secure Dropbox account information. Two application modes will be provided: using regular mode when there is Internet access while in local mode when there is no Internet access. The different operation permission schema is granted to users.

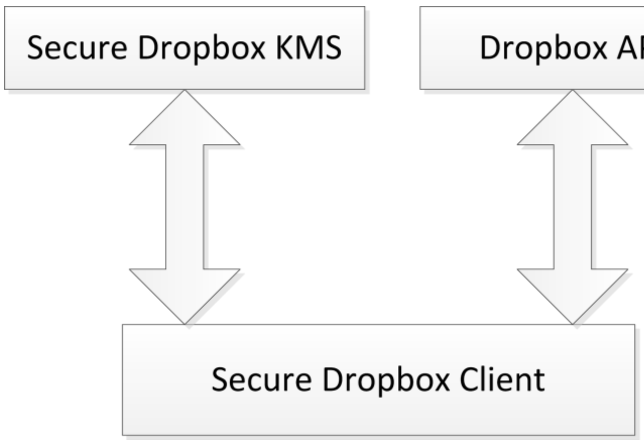


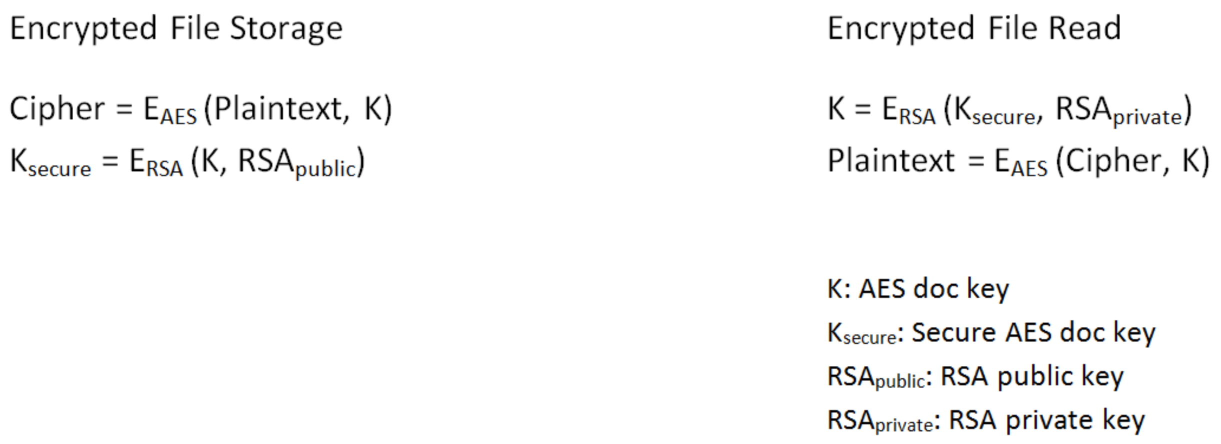
Figure 3.1: Secure Dropbox Architecture

This chapter will discuss some major design decisions and implementation challenges during the Secure Dropbox project. Firstly, the overall design of Secure Dropbox will be proposed. Secondly, the cryptography application mechanism to guarantee the security of Secure Dropbox will be explained and justified. Furthermore, the reason for making choices about ways of performing file operations, systematic integration with Dropbox and a platform on which to construct the application will be discussed in regard to building a platform independent application. Moreover, the decisions made in terms of building a C/S architecture application but not in B/S architecture will be explained. Some other design features are more intuitively understandable by explaining from the implementation aspect so more detail of those parts will be proposed in the next chapter or just briefly mentioned in this chapter. Lastly, the design decisions and other principles proposed to implement Secure Dropbox are not Dropbox oriented specifically but could be adopted when designing any application that protect users file in the cloud and provide a reliable file sharing in the cloud storage system.

3.1 Application of Cryptography

Dropbox claim that they are using Secure Socket Layer (SSL) on data transmission and performing AES 256-bit encryption with users data. It seems sufficient against external attack but only on condition that the attacking does not come to key management service (KMS). However, these mechanisms make no sense of protecting users data from internal attack, which is performed by employees of Dropbox who have access permission to KMS. The behavior, or say the ability, of accessing to users data has been admitted in their terms of service [25] that Dropbox will remove the encryption from file data and deliver them to law enforcement on governments requirement. Nevertheless, because of internal attackers proficient understanding of Dropbox system and some inappropriate access permission configurations, internal attacks could even be done unconsciously. These security concerns have been certified that several internal attack events are reported. Client end encryption is one of those alternatives recommended by Dropbox if users are more willing to conceal their data and care less about losing some Dropbox features like version control or data recovering [25]. More importantly, when protecting users data from internal Dropbox attack with client end encryption, it actually disables the file sharing infrastructure provided by Dropbox totally.

Anyway, cryptography is still going to be the very insurance for the security of Secure Dropbox. Generally, Secure Dropbox is designed to make Dropbox internal attack impossible and also to provide a cipher file sharing service. To achieve this goal, Secure Dropbox will apply a combination of symmetric cryptography and asymmetric cryptography. The symmetric cryptology like AES or DES will be used for file content encryption given the fact that they are much faster than asymmetric cryptography and sufficient security offered. The asymmetric cryptology is used for protecting the symmetric encryption key and providing a secure key transmission service. In the following example, the procedure of secure storage is going to be illustrated with AES and RSA as instance of symmetric and asymmetric cryptography respectively:



Encrypted File Storage

$\text{Cipher} = E_{AES}(\text{Plaintext}, K)$

$K_{secure} = E_{RSA}(K, RSA_{public})$

Encrypted File Read

$K = E_{RSA}(K_{secure}, RSA_{private})$

$\text{Plaintext} = E_{AES}(\text{Cipher}, K)$

K: AES doc key

K_{secure}: Secure AES doc key

RSA_{public}: RSA public key

$RSA_{private}$: RSA private key

Figure 3.2: Encrypted File Storage and Reading

In the file encrypted storage procedure, plain text is ciphered in AES with a random generated AES file encryption key. The doc key is later ciphered in RSA as well with file owners RSA public key. In encrypted file read procedure, firstly the ciphered doc key K_{secure} should be deciphered in RSA with file owners RSA private key. Having retrieved the raw doc key K, the cipher could be decrypted in AES with K. The procedures of sharing from Secure Dropbox users is mainly about ciphering the doc key K and transferring it to the sharing recipient. K should be ciphered by file owner but could only to be deciphered by sharing recipient, which is a typical scenario for asymmetric cryptography application. In the following example, Alice wants to share a file with Bob:

Sharing procedure as shown above actually does not perform any operation on the file instance but only cipher the key for sharing purpose. The doc key is stored in a secure way at the very beginning. To get the raw doc key, Alice has to decipher K_{secure} in RSA with her own RSA private key. In order to enable Bob deciphering the key ciphered by Alice, she should encrypt the raw doc key K in RSA with Bobs RSA public key, which is open to all the users in the Secure Dropbox system. $K_{sharing}$ is generated in this way and then transmitted to Bob via secure tunnels. After receiving the $K_{sharing}$, Bob will decrypt it in RSA with his own RSA private key and then get the raw doc key. Now the plain text of the shared encrypted file could be retrieved by decrypting in AES with the raw doc key.

Encrypted File Sharing

$K = E_{RSA}(K_{secure}, RSA_{private}\_A)$

$K_{sharing} = E_{RSA}(K, RSA_{public}\_B)$

Encrypted Shared File Reading

$K = E_{RSA}(K_{sharing}, RSA_{private}\_B)$

$Plaintext = E_{AES}(Cipher, K)$

K: AES doc key

K_{secure}: Secure AES doc key

$K_{sharing}$: AES doc key to share

RSA_{public}_A: RSA public key of Alice

$RSA_{private}\_A$: RSA private key of Alice

RSA_{public}_B: RSA public key of Bob

$RSA_{private}\_B$: RSA private key of Bob

Figure 3.3: Encrypted File Sharing

Respecting managing the RSA key pair of Secure Dropbox user and doc key of each file, a detailed design description will be given in the coming section.

3.2 Key Management Service

In the Secure Dropbox application scenario, there should be an infrastructure posting all Secure Dropbox users RSA public key for sharing sponsor to achieve during the encrypted file sharing procedure. In addition, given the essence of Secure Dropbox (a client end file encryption tool) and Dropboxs cloud storage nature, the requirement of everywhere use should be met. It means as long as users could get access to Dropbox, they are allowed to use the Secure Dropbox service like downloading the desired doc key chain and RSA key pairs. To achieve the goal, besides generating a local copy of essential data like Secure Dropbox users authentication information, RSA key pairs, doc keys and file sharing information, these records should also be stored on a server which is able to be accessed anytime anywhere.

Dropbox stores file encryption key in its own database makes it not an accredited storage service to users. However, in order to avoid same security concerns towards

Dropbox about its encryption mechanism, some improvements should be made. To make Secure Dropbox reliable, the authority of decrypting file should be only granted to owner but no one else even the KMS administrator. A possible alternative is storing all sensitive data (e.g. RSA private key and doc key) on the server confidentially so that the everywhere usage could be realized safely. Therefore, the only entry (assume it could be something like an access token or password) of decrypting all these sensitive data should never be stored on the server but only held by users themselves.

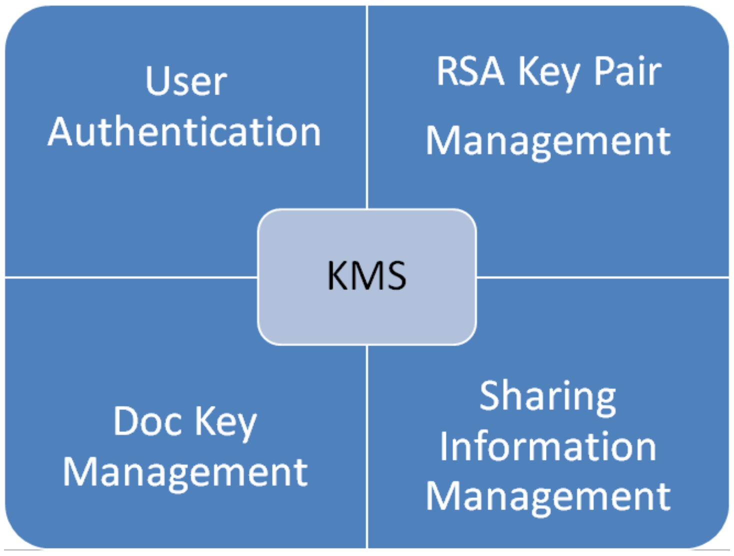


Figure 3.4: KMS Architecture

3.2.1 KMS User Authentication

User authentication information is a basic component in any web application instance as long as access control is required. The classic way applied in web application saves users authentication could be adopted: for the minimum usage, a plain text username and a hashed password should be saved. The authentication procedure will match the username and password hash value. The hashed password is generated in the client and only the hash value with algorithm indicator prefixes will be uploaded to KMS and stored. To avoid performing any suspicious behavior, plain text password should be hashed in client once it is received in the terminal and uploaded for authentication. There is no plain text password involves in the KMS server end. The hash value matching could be done at client as well but it requires a corresponding KMS interface for users to fetch hash algorithms salt, iteration times and hash value.

3.2.2 KMS RSA Key Pair Management

The RSA key pair could be generated either on KMS or client although it would potentially bring about better user confidence when everything is generated in the client and uploaded after encryption. In KMS, RSA public key is stored in plain text so that it could be reached by any user who wants to process the doc key before sharing the file. An RSA private key should be stored in cipher. It is encrypted in symmetric cryptography with users own password or token as encryption key and uploaded once generated in client end. Any request about fetching Secure Dropbox users RSA public key should be allowed.

3.2.3 KMS Doc Key Management

For each file encrypted by Secure Dropbox, a unique file encryption key will be generated. Before uploading to KMS, it is a necessity to encrypt the doc key in RSA with file owners RSA public key. The documents name will be stored in plain text for easy indexing purpose. In this way, any operation on this encrypted file, no matter reading or sharing, could only be initiated by the file owner. It is because both procedures require file encryption key that is only known to file owner. The RSA private key is involved in decryption in corresponding to the encryption with RSA public key that from the same key pair.

3.2.4 KMS Sharing Information Management

The key component of sharing data is still the processed doc key. The processing procedure should be performed on the client. The sharing recipient will be able to get the processed key and decrypt it with own RSA private key. Besides it should include sharing metadata generated by Dropbox sharing API: a URL indicates the entry of the file content and a sharing expiration timestamp indicates when the URL will be closed for access.

3.2.5 Local KMS

There are two modes for Secure Dropbox: regular mode when Internet access is available and a local mode when Internet access is not available. A local copy of KMS information related to the Secure Dropbox user is generated for local usage like providing user authentication information, RSA key pair information and file encryption keychain. Since there is no Internet access, User can neither share a file with other Secure Dropbox users nor read shared files from other Secure Dropbox user. Users own files stored in the local file system are still accessible given the design decision made about using the Dropbox official application as file container. The local RSA key pair and doc keychain make it possible to decrypt any encrypted files. Apparently this local copy should be protected and encrypted as well.

3.3 Right Cryptography Algorithm

3.3.1 Symmetric Cryptography

In the symmetric (private-key) cryptography procedure, encryption and decryption are performed with same key [26]. Accordingly, encryption key sharing becomes the precondition of the sharing the symmetrically encrypted information. Theoretically, as it is significantly faster in comparison with asymmetric algorithm with the same security level, symmetric encryption algorithm is usually used for big data protection.

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [27]. NIST claimed that AES with 128-bit keys provides adequate protection for classified information up to the SECRET level definition. Increasing AES applications have changed to the AES-256 in which more rounds hash and longer key are applied and fulfill a TOP SECRET level by definition (Both definitions of SECRET and TOP SECRET are advised by CNSSP-15 [28]). Numerously there are 3.4×10^{38} combinations to guess for AES-128 while this number is squared and comes to an incredible 1.1×10^{77} for AES-256.

As Dropbox is using AES-256 for encryption in transit and encryption in rest, seemingly it would be ideal to deploy an AES-256 encryption in Secure Dropbox for file encryption as well. As claimed by NIST [29], the encryption strength of AES-256, which has an equivalent security level as RSA-15360, will be sufficient until 2030. In Secure Dropbox, AES-256 will be the most direct protection of all data include encrypted file storage, encrypted local storage of users file encryption key chains and other confidential data in server end as long as content concealing is required.

However, since the guaranteed security totally depends on a proper key usage and storage, it is highly risky to transmit the unprotected file encryption key via unreliable media like the Internet. So, despite of those application layer protocols for secure data transmission like SSL, hybrid encryption which combines the symmetric encryption and asymmetric encryption would be a better way to directly protect the symmetric encryption key against risks during key exchange.

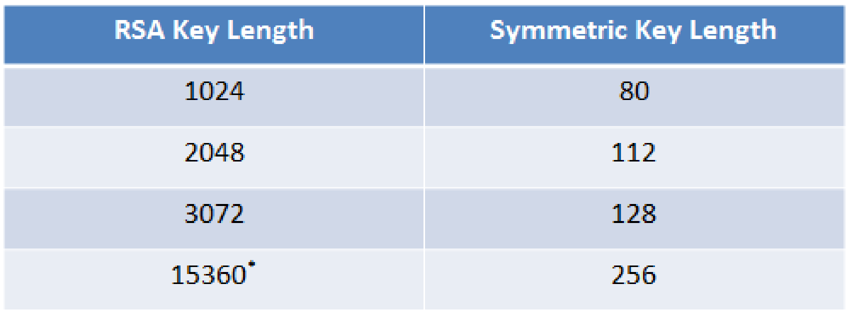
3.3.2 Asymmetric Cryptography

Practically, the security of cryptography nowadays is no longer guaranteed by concealing the cryptography algorithm itself but lies on the encryption key strength and mechanism of key protection. While sharing the symmetrically encrypted data will inevitably involve key exchange on unreliable public tunnels where interception and distortion are happening, the asymmetric cryptography turns out to be a better option.

In asymmetric cryptography, the encryption and decryption are performed with mathematically linked public key and private key respectively. Generally speaking, the public key is for encrypting the plain text while the private key is used for decryption purpose. The key pair is generated by a trusted PKI. The key pair, especially the private key, is distributed far less often than symmetric encryption application scenario. It essentially reduces the security threats brought by frequent key exchange. As a prerequisite of encrypted information sharing, the public encryption key is widely distributed and accessible by anyone who wants to activate information sharing. However, the private key will be only known to the encrypted information recipient. As a result, there is only transmission of encrypted data which could be considered as secure rely on the strength of asymmetric cryptography algorithm and its key length.

Nevertheless, due to the different mathematical natures in comparison with symmetric cryptography, asymmetric algorithm is remarkably less efficient. Most applications of asymmetric cryptography are limited to small data encryption or as a component in hybrid encryption. For example, to share a large encrypted data, the data content is encrypted in the faster symmetric cryptography such as AES while the relatively short AES key is protected with asymmetric cryptography before transmission.

RSA is an algorithm for asymmetric cryptography based on the presumed difficulty of factoring large integers. As claimed by RSA Security, there is a mapping relationship of security strength between RSA and AES in terms of the encryption key length. The correspondence is listed as follows:



RSA Key Length	Symmetric Key Length
1024	80
2048	112
3072	128
15360*	256

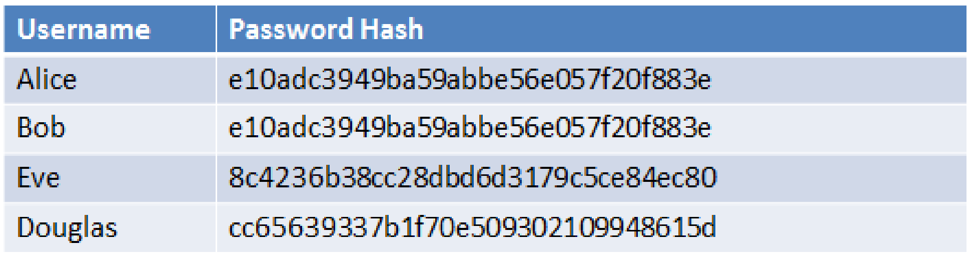
Figure 3.5: Strength Equivalence [30] [29]

RSA Security also claimed that 1024-bit keys are likely to become unsafe sometime between 2006 and 2010 while the 2048-bit keys are sufficient until 2030. Thus the RSA key length of 3072 bits will be required since 2030 [30]. RSA with 2048-bit or longer key should meet the requirement of file encryption key security of Secure Dropbox.

3.3.3 Cryptographic Hash Function

Cryptographic hash functions perform irreversible encryption on an arbitrary length plain text and returns a fixed length cipher. The hash digest is fixed for same input but tremendously changed as long as changes happen to content, no matter how trivial it is [31]. These one-way hash functions provide a rapid method for detecting any change made to the content and are also used for generating individual digest of content [31]. In web application context, users confidential information is usually stored in the form of hash digest in case of being exposed when server is exploited.

Since the hash function by definition has the same output with the same input, hash collision attack is easy to perform. For example: Alice sets her password as “123456” while Bob coincidentally sets his password as “123456” too. Consequently, same password hash values will be generated and stored. If Alice happens to get the password hash table that stored in the server, she might recognizes that Bobs password hash value is exactly the same with hers and further realizes that Bobs plain text password could be the same as Alices:

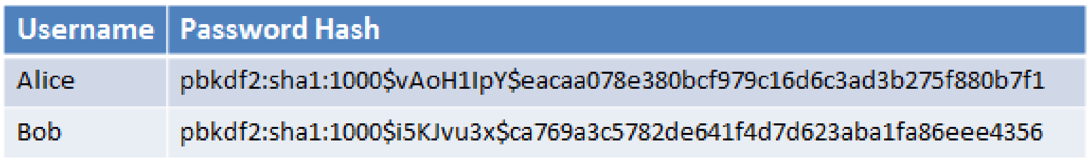


Username	Password Hash
Alice	e10adc3949ba59abbe56e057f20f883e
Bob	e10adc3949ba59abbe56e057f20f883e
Eve	8c4236b38cc28dbd6d3179c5ce84ec80
Douglas	cc65639337b1f70e509302109948615d

Figure 3.6: Hash Table

Also, if Alice has the dictionary of most frequently used password, she could try to hash each password in the collection and match to see if the same hash value exists. Accidentally she might find that Eves hashed password could be achieved by hashing “654321” with certain algorithm.

To protect the password hash value against hash collision attack, a random generated salt and configurable hash function iteration time could be applied when performing the content hashing. Randomly generated salt acts as a part of content to make same content distinct. The stored hash value is the result of hashing the combination of primary content and salt. Also the adoption of different iteration with different round times leads to a more sophisticated mapping relationship between plain text and hash value. The following example illustrates how a hash table with salt and iteration information stored in the server:



Username	Password Hash
Alice	pbkdf2:sha1:1000$vAoH1IpY$eacaa078e380bcf979c16d6c3ad3b275f880b7f1
Bob	pbkdf2:sha1:1000$i5KJvu3x$ca769a3c5782de641f4d7d623aba1fa86eee4356

Figure 3.7: Hash Table with Salt, Iteration and Algorithm Indicator

As shown above, there is additional information attached to the hash value. Take first record for example:

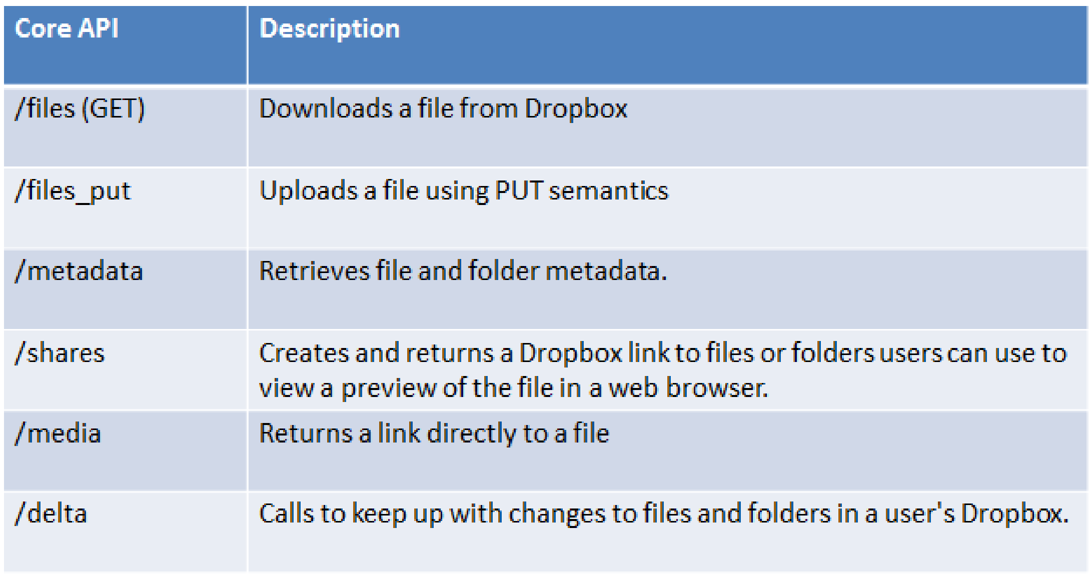
- pdkdf2 indicates the librarys name via which the hash algorithm is used.
- sha1 indicates the algorithm used to generate the hash value.
- 1000 indicates there are 1000 iterations when performing the password hashing.
- "vAoH1lpY" is the salt added to the plain text password before it is hashed.

The hash value to be matched should be generated with the above parameters. In this example, the SHA1 generates a 160-bit long digest of the content. It could be considered as a sufficient secure hash function since it is still widely used in mainstream security protocols like TLS and SSL.

3.4 Integration with Dropbox

Dropbox provides several kinds of APIs for Dropbox developers. For example, The Dropbox Core API, which is recommended as an ideal API set by Dropbox for server-based applications with programming interface for file reading and writing provided. It is also claimed as the most direct way to access Dropbox. Furthermore, it includes some advanced functionality interfaces like content search, version control and restoring file service for developers to performing low-level controls. The Dropbox Sync API, which provides file system-similar programming interface, encapsulates functionality implementations like synchronizing and notification of remote changes. It mainly orients to mobile platform programming. With respect to implementing Secure Dropbox, not only file synchronization operation like uploading and downloading, but also some low-level operations like file sharing, sharing URL generation and file metadata checking are required. To use Dropbox Core APIs, an access token obtained during Dropbox OAuth is essential. The access token will allows Secure Dropbox, a third-party application, to use the Dropbox API with granted permission but never get disclosed with any Dropbox user account information. The implementation details about how to perform the indirect authentication will be proposed in the implementation chapter. Some key Dropbox Core APIs are listed as follows:

Except exploiting these Dropbox APIs as advised, there is another way to use Dropbox service synchronizing files through the Dropbox official application. Using Dropbox on the host computer is just like using any other folder in the file system. However, the files you drag or copy into Dropbox folder will be automatically synchronized and consequently synchronized to other terminals like mobile devices which are linked to Dropbox account. In this way, the Core API is still used but indirectly because the underlying interface for Dropbox official applications is implemented in Core API. The story with regard to file operation could be tremendously simplified if Secure Dropbox use the Dropbox client application as target folder and perform all file operations on it. Correspondingly,



Core API	Description
/files (GET)	Downloads a file from Dropbox
/files_put	Uploads a file using PUT semantics
/metadata	Retrieves file and folder metadata.
/shares	Creates and returns a Dropbox link to files or folders users can use to view a preview of the file in a web browser.
/media	Returns a link directly to a file
/delta	Calls to keep up with changes to files and folders in a user's Dropbox.

Figure 3.8: Key Dropbox Core APIs Functionality Description

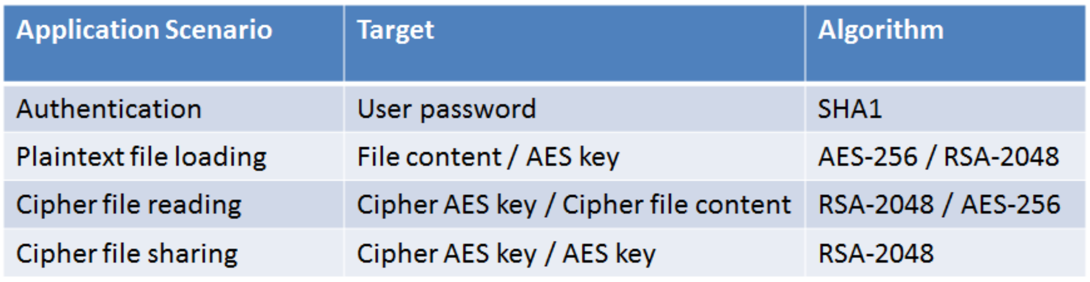
every encrypted file written into the Dropbox client folder will be uploaded automatically but without any Dropbox API invoked explicitly. A commercial software product with the same design and implementation relating to the hybrid usage of Dropbox APIs is Boxcryptor [32]. Boxcryptor has a driver level encryption which is more efficient than user space encryption. However, the nature of client end encryption disables the sharing service infrastructures provided by Dropbox since file shared that way would be nonsense to human reader since the data is encrypted. Boxcryptor made their own sharing service which partially depends on Dropbox core API service like getting raw content of encrypted file via "/media" interface and sharing after decryption.

Although the Dropbox client folder eases the complexity when design the file operation module of Secure Dropbox, the file sharing feature of Dropbox client has been disabled. To analyze from API invokings perspective, "/share" function of Dropbox core API returns the URL refers to file or folder entity encapsulated with html information so there could be a rendered display of certain entity while the "/media" function returns the URL refers to a raw file content text. Apparently, if a file has been encrypted by Secure Dropbox and uploaded to Dropbox, it would be accessible by authenticated user but not meaningful until these encrypted data have been through the decryption service provided by Secure

Dropbox. A potential solution to such a problem is a combination usage of Dropbox client and Dropbox API. For instance, the shared information could be accessed by firstly get the encrypted raw data via "/media" interface and decrypted in Secure Dropbox while other synchronization operations are still based on the Dropbox application. Another concern derives from the lack of access controls on Dropbox official application. The permission to use Dropbox application is incorrectly granted in the following scenario: Alice logins to Secure Dropbox service with her Secure Dropbox account and performs some file operations at her laptop. Bob comes to Alice and asks for a temporary use. When Bob logins with his Secure Dropbox account and uses Secure Dropbox, the file operations are actually performed on Alices Dropbox account. Dropbox does not open API to reset the Dropbox client account information of Dropbox application.

3.5 C/S or B/S

As so long as there is a cryptographic operation happens in the server end, there will be encryption key involved inevitably. It also potentially indicates that the server is allowed to do anything with the encryption key like storing or distributing it. So, to make Secure Dropbox security confident to users, KMS of Secure Dropbox will not be allowed to involve any cryptography procedure. Apparently a B/S architecture, where client is designed to be light while the server is fully responsible for all computations, will not be considered as a reasonable decision. It could be the foremost reason why a C/S designed architecture is more appropriate. However, to reduce the workload of server, in C/S architecture client is assigned with more tasks. Also only the processed data will be submitted to the server. Besides the key involvement issue, in Secure Dropbox the cryptography is the most resource consuming procedure especially when data to be processed is huge. All these tasks done by Secure Dropbox client end could significantly reduce the servers workload. Based on this design, KMSs work has been simplified to only process IO requests and perform local storage procedure. Where the encryption potentially involved in Secure Dropbox is illustrated as follows:



Application Scenario	Target	Algorithm
Authentication	User password	SHA1
Plaintext file loading	File content / AES key	AES-256 / RSA-2048
Cipher file reading	Cipher AES key / Cipher file content	RSA-2048 / AES-256
Cipher file sharing	Cipher AES key / AES key	RSA-2048

Figure 3.9: Cryptography Application Scenarios in Secure Dropbox

Chapter 4

Implementation

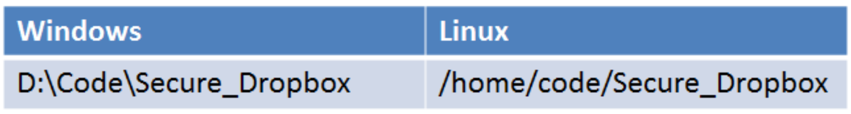
The previous chapter explained some major design decisions made during this project and this chapter will describe the architecture and implementation details of the resulting application of Secure Dropbox. The implementation language, relevant libraries and implementation platform will be introduced at first. Then the communication pattern of Secure Dropbox KMS and Secure Dropbox client will be explained. The implementation of Secure Dropbox KMS, the Secure Dropbox client and their various components, will be illustrated in detail.

4.1 Implementation Dependencies

4.1.1 Secure Dropbox Client

Both client end and KMS of Secure Dropbox are implemented in Python. To implement the file encryption functions, driver level encryption will be more efficient than user space encryption but might encounter with compatibility problems on different platforms. One reason to choose Python as implementation language is it is an efficient programming language that widely supported to run on most mainstream operating systems like Windows, Linux/Unix and Mac OS X. Moreover it has been ported to the Java and .NET virtual machines already. Program once written could be executed everywhere as long as Python interpreter has been installed. It is a high-level programming language and its libraries and syntax regulation allow developers to implement certain functions with fewer lines of code than in lower level programming language like C. It increases the readability of code and boosts up the programming procedure. Although always being claimed as a scripting language, the object-oriented feature makes Python competent in large scale program implementation as well. Another reason to choose Python is because of its relatively stable version history. Java has 51 different releases for Java SE 6 from 2007 to 2011 and 25 releases for Java SE 7 from 2011 to 2013. However, Python has only 8 versions for Python 2.x in the last 13 years and the latest Python 2.7 has been stable since July 2010. As Secure Dropbox is designed to be a C/S architecture system, a frequently changed implementation platform may cause compatibility problems and make software update routines complicated. Also, Python is the advised programming language by Dropbox to make use of their Core API. Especially, some advanced features are currently only supported by the Python API release like OAuthV2 which performs an indirect Dropbox Authentication during Python API programming. With regard to server end implementation, Pythons powerful web development frameworks like Django or web2py provides mature web development toolkits and libraries to use. Django even automatically provides a well generated back end application console for the web application which reduces development workload tremendously. In conclusion, Python could be an ideal programming language to implement the Secure Dropbox Project. The implementation platform becomes less important when programming in Python. Pythons interpreter mechanism insures that as long as there is no operating system specified features used in code, the program could be executed everywhere with exact same outcomes. An example of operating system specified feature could be illustrated with the following example which was actually encountered during the Secure Dropbox course project:

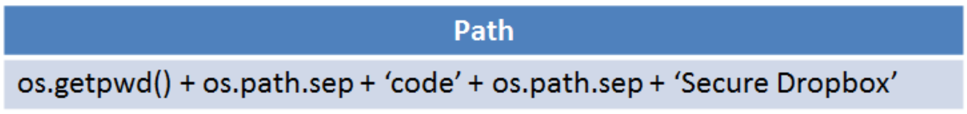
The two different representations indicate that Windows and Linux are using different symbols as path separator. Assume current working directory is D: in Windows and /home on Linux. For both of them there is a involving some file operations upon a



Windows	Linux
D:\Code\Secure_Dropbox	/home/code/Secure_Dropbox

Figure 4.1: Representation of Path in Different Operating System

certain file in the folder Secure_Dropbox. When specifying the file path, either "/" or "\" may leads to compatibility issues on different operating systems and file path exceptions. Such problems can be avoided. For example, in Python the path could be generated as follows:



Path
os.getpwd() + os.path.sep + 'code' + os.path.sep + 'Secure Dropbox'

Figure 4.2: Platform Independent Coding Style

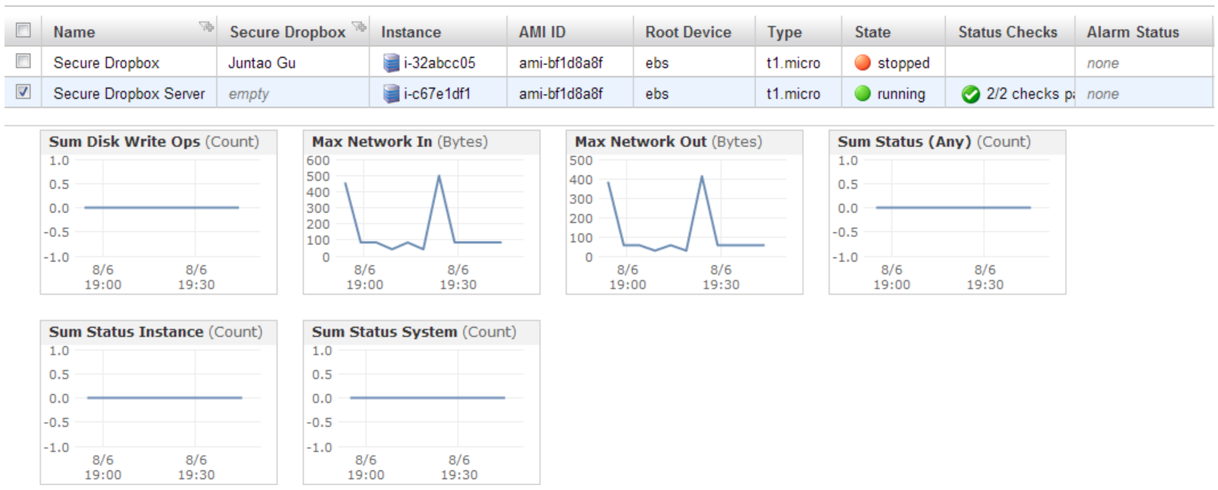
In this way, the variable "os.path.sep" will be replaced with "\" on Windows or "/" on Linux by Python interpreter. To make Python program platform independent, any operating system individual features should not appear in the code. It is advised to make more use of the Python OS or other libraries. Secure Dropbox was developed in Windows 7 64-bit operating system but designed as multi operating system usage software. Python 2.7.5 64-bit which was released on May 15th, 2013 was adopted for implementation of both KMS and client. All cryptography modules are generated based on accredited Python cryptography libraries like PyCrypto or M2Crypto.

4.1.2 Secure Dropbox KMS

Secure Dropbox KMS is implemented as a Restful WebService given those desired features. The WebService implementation is based on the Python bottle library which provides ready to use Restful interface for the application.

KMS has been deployed on Ubuntu Server 13.04. Ubuntu Server is currently the most popular guest server platform on the worlds leading public clouds, regarding the total number of instances running or the diversity of customized images available. It offers a complete solution for building highly available, flexible and secure server application production with stable and efficient storage, networking and computation capabilities.

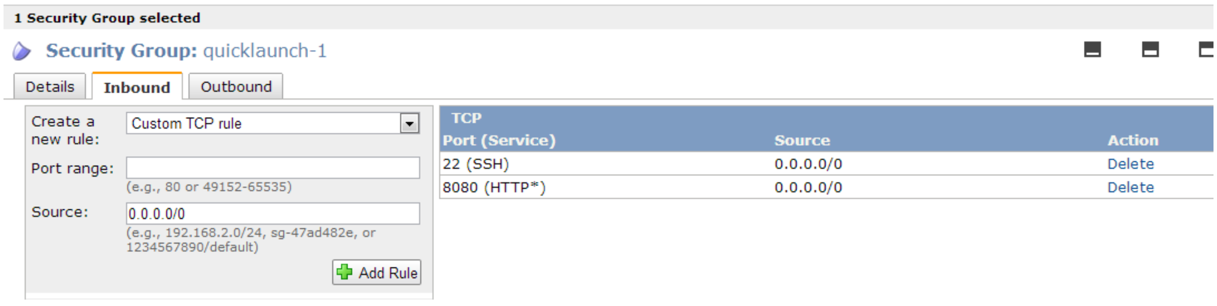
The Ubuntu Server instance with Secure Dropbox KMS Running is currently deployed on the Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 is a computing platform which provides resizable compute capacity in the cloud. AS a representational Cloud Infrastructure as a Service (IaaS) cloud platform, Amazon EC2 rents virtual machines with specific computation resources. Also it provides the developers and administrators with the capability of provision processing, storage management, network configurations and other fundamental computing resources to facilitate application deployment and running. Although Ubuntu Server provides monitoring infrastructures, the administration console provided by Amazon EC2 is more intuitive and comprehensive. A sample server monitoring interface is as follows:



☐	Name	Secure Dropbox	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status
☐	Secure Dropbox	Juntao Gu	i-32abcc05	ami-bf1d8a8f	ebs	t1.micro	● stopped		*none*
☑	Secure Dropbox Server	*empty*	i-c67e1df1	ami-bf1d8a8f	ebs	t1.micro	● running	✓ 2/2 checks pa	*none*

Figure 4.3: Amazon EC2 Server Monitoring

Amazon EC2 also offers an intuitive network access control interface. For example, only port 22 is open by default to capacitate the SSH access from any host. Network data flow has been categorized as inbound and outbound direction. Both directions are allowed to be customized by EC2 administrator. For instance, since Secure Dropbox KMS is a restful WebService, all the inbound http requests should be allowed to go through the access control. To customize the schema, the administrator needs to choose the inbound tag, select the rule as custom TCP rule, set any available port number and allow address of 0.0.0.0/0 which means no access control rules on this port. Configuration interface as follows:



Security Group: quicklaunch-1

Create a new rule: Custom TCP rule

Port range:
(e.g., 80 or 49152-65535)

Source: 0.0.0.0/0
(e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

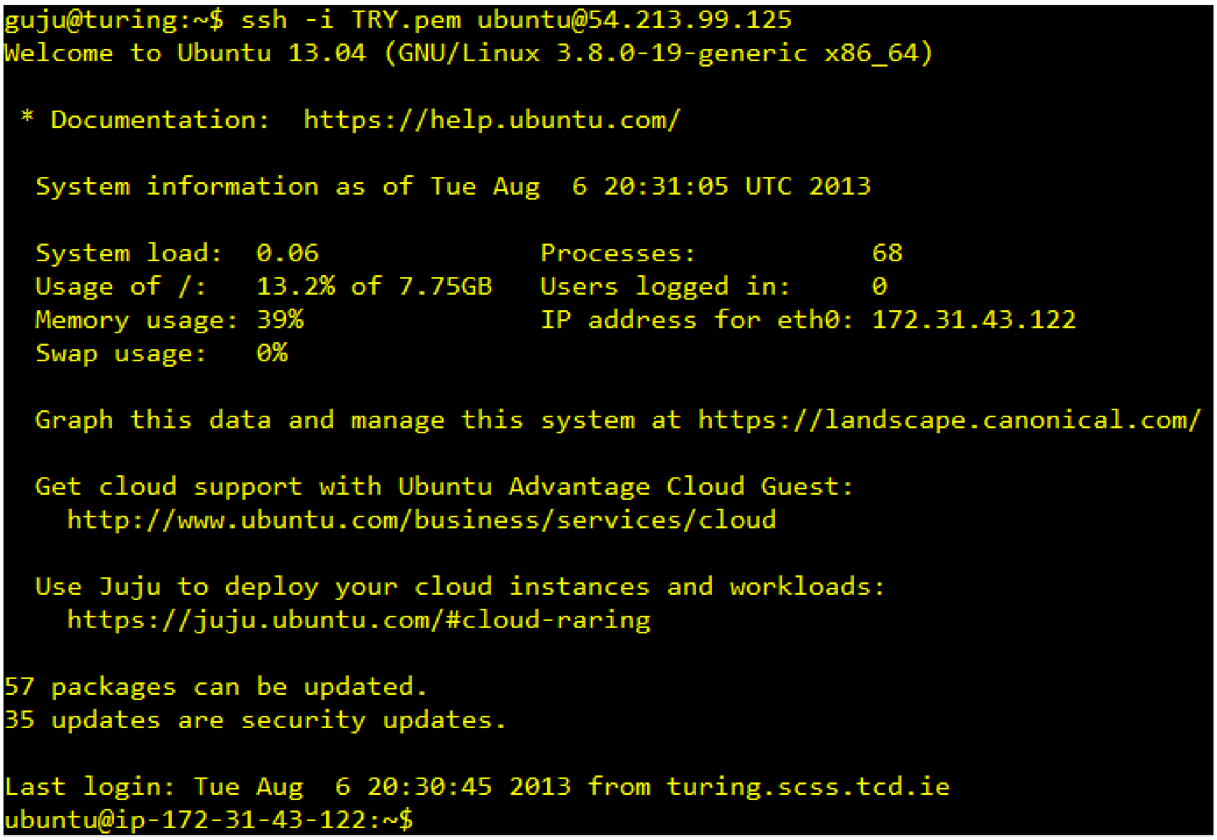
TCP

Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete
8080 (HTTP*)	0.0.0.0/0	Delete

Figure 4.4: Amazon EC2 Security Customization

To SSH to the Ubuntu Server instance on EC2, An identity based authentication is required. For each cloud application deployed on EC2, a private-key file which associated with the instance will be delivered to the administrator. User is by default granted with root permission if the private-key file is authorized. The following figure indicates the procedure of SSH to Amazon EC2 instance from TCD SCSS Turing service. The SSH command includes a private-key file TRY.pem, a default username "ubuntu" and the elastic IP Address of Ubuntu Server Instance. The prompt from the Ubuntu Server instance have no expression about any Amazon EC2 information. Some basic monitoring information like current process number, CPU load and Memory usage will be displayed once login succeeds:

Secure Dropbox KMS uses SQLite as database. SQLite is a SQL database engine delivers file based storage service. It is suitable for building a lightweight disk-based database that does not require a separate server process. The standard python module sqlite3 provides a SQL interface compliant with the DB-API 2.0 specification and a built connection instance that could access to the SQLite database directly.



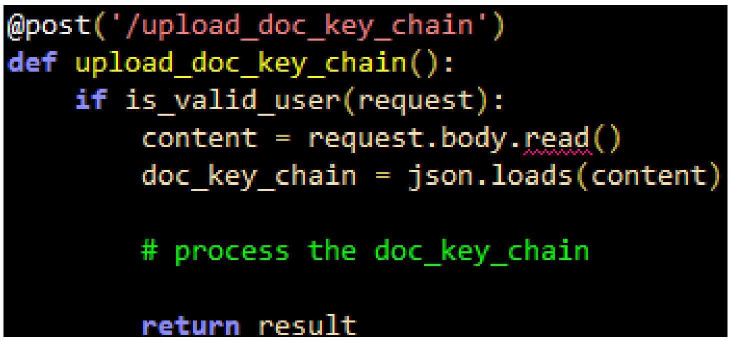
```
guju@turing:~$ ssh -i TRY.pem ubuntu@54.213.99.125
Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-19-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Aug  6 20:31:05 UTC 2013

  System load:  0.06              Processes:           68
  Usage of /:   13.2% of 7.75GB   Users logged in:     0
  Memory usage: 39%               IP address for eth0: 172.31.43.122
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

  Use Juju to deploy your cloud instances and workloads:
    https://juju.ubuntu.com/#cloud-raring

57 packages can be updated.
35 updates are security updates.

Last login: Tue Aug  6 20:30:45 2013 from turing.scss.tcd.ie
ubuntu@ip-172-31-43-122:~$
```

Figure 4.5: Amazon EC2 SSH Procedure

4.2 Communication

Except the encrypted file instance itself, all information data involves in Secure Dropbox client are downloaded from KMS during initialization and any newly generated data have to be synchronized and permanently stored in KMS. For example, before authentication procedure, the password hash algorithm, salt and iteration time information have to be fetched and the locally hashed password has to be uploaded to KMS again for matching purpose. The communication between the Secure Dropbox client and KMS is performed in REST-style. A typical REST-style architecture generally consists of client end and server end. The client initiates by sending requests to the server. Having received the request, the server will process it and return the corresponding responses to the client. REST requests and responses are generated aimed at the transmission of representations of data resources. Resource to be sent can be essentially any comprehensible and meaningful concept that able to be addressed by both ends. The representation of certain resource is typically a formatted text or document that contains the state or value. In Python programming, JSON is the standard data format that always used for formatting the data to be sent.

To build a REST WebService Server, The python web framework Bottle is used in this project. The Python bottle is a fast, simple and lightweight WSGI micro web-framework for Python. To generate a request routing, a tag with http request method type and desired routine name are proposed as follows:



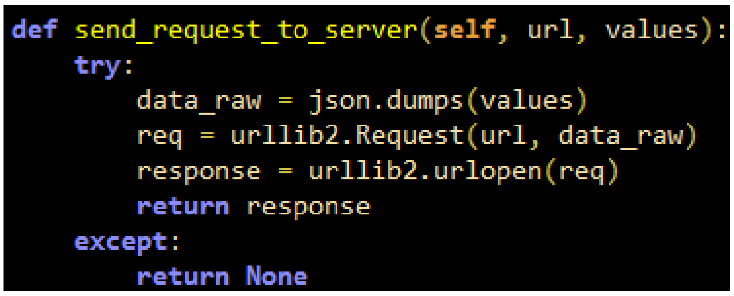
```
@post('/upload_doc_key_chain')
def upload_doc_key_chain():
    if is_valid_user(request):
        content = request.body.read()
        doc_key_chain = json.loads(content)

        # process the doc_key_chain

        return result
```

Figure 4.6: Restful WebService Programming with Python Bottle

- This function will process any POST requests with URL points to the CGI /upload_doc_key_chain.
- The function "upload_doc_key_chain()" is called when such a request arrives.
- The resource content need to be transmitted and processed is encapsulated in the HTTP body.
- The resource content has been formatted in JSON so it could be retrieved again if loaded in JSON format.
- The return value is directly returned to the requester.

In the client end, the request is generated as building an http request with required resource data. It could be done with the support of urllib2. It is a standard module in

Python 2.7 with definitions of functions and classes which helps in fetching URLs. To call the remote KMS APIs, the target URL should be specified as the routine name that predefined in WebService end. It is simplified by urllib2 to generate an HTTP request and send it to KMS:



```
def send_request_to_server(self, url, values):
    try:
        data_raw = json.dumps(values)
        req = urllib2.Request(url, data_raw)
        response = urllib2.urlopen(req)
        return response
    except:
        return None
```

Figure 4.7: HTTP POST Request Generating

- The data to be transmitted has to be formatted into JSON
- An HTTP POST request is generated with data and target URL as parameters
- The request is sent via "urllib2.urlopen()" method and response data received as return value.

All communication sessions are initiated by the Secure Dropbox client. Also these data exchanges are designed with no state so that any communication step is atomic and finished in a single session. The request about fetching data will get a response of data instance and request about uploading KMS data will get an error code depends on if the operation succeeds or not.

4.3 Secure Dropbox KMS Implementation

KMS plays a key management processor role in Secure Dropbox. There are three different types of tasks running in KMS: user management, key management and file sharing management. User management processes users registration and login requests. Key management mainly performs CRUD operations upon encryption key related issues. File sharing management is responsible for file sharing notification and performing a time task to handle the expired file sharing information. The implementation of some important features and WebService interface will be explained.

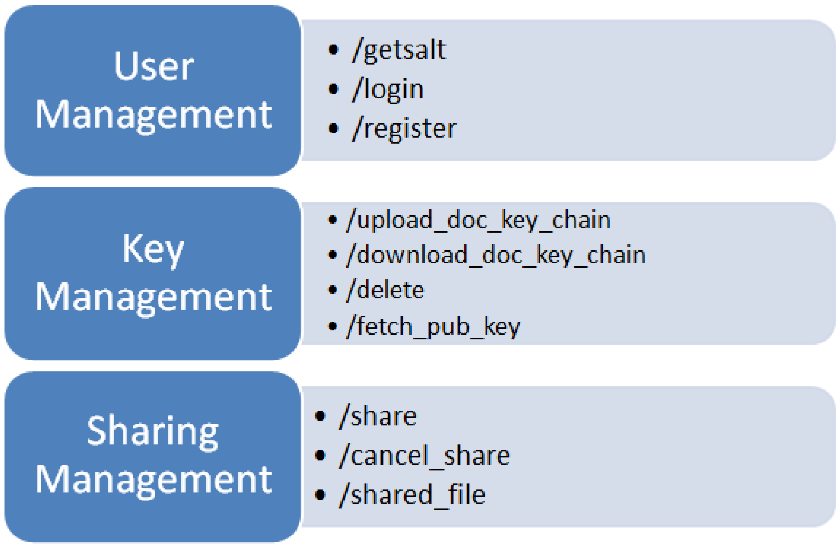


Figure 4.8: KMS Interfaces

4.3.1 Database Design

There are three tables in database of Secure Dropbox KMS. The “user” table stores users username, password and RSA key pairs. Users RSA public key is recorded in plain text while the private key is stored with encryption. The token is a timestamp to indicate the last login time. The “key_chain” table includes each users file encryption keychain which is composed by a unique doc_id and doc_key. The doc_id is guaranteed to be unique since it includes the file owners username which is unique defined as public key in the user table. Each file has its doc_id and corresponding doc_key which is essentially an AES-256 key. The doc_key is encrypted with the owners private key. The “sharing_pool” table consists of sharing information like sharing sponsor and recipient. The doc_key in sharing_pool is encrypted with sharing recipients public key so that the shared doc_key could be decrypted with the recipients private key. The field URL and expires is generated by Dropbox “/media” API. The URL points to the raw encrypted file content and the field expires indicates when the URL turns into not accessible. There is a scheduled task handles the records and deletes those expired records in “sharing_pool” table because certain sharing information becomes useless after getting expired. Logically they should not be able to searched and displayed to the user.

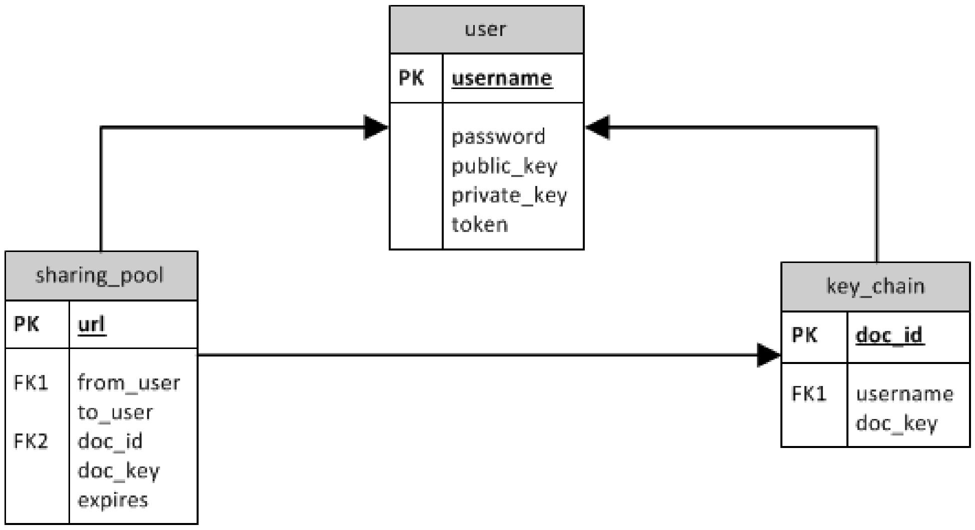


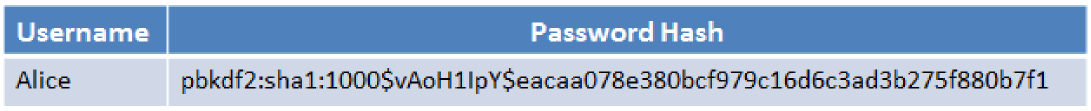
Figure 4.9: KMS Database Design

Although stored in the database of KMS, all these data are generated and uploaded by a Secure Dropbox client. Some essential fields are encrypted with the key not known to KMS but only known to account owner.

4.3.2 KMS User Management

The interface “@post(/getsalt)” returns password hash algorithm parameters like algorithm name, salt value and iteration times to Secure Dropbox client. Thus client is able to hash the plain text password with the same parameters as how the hashed password value stored in KMS database was generated. It is called before client performing the login procedure. The function will return a Python dictionary variable like {“algorithm”: “sha1”, “iteration”: 1000, “salt”: “vAoH1lpY”} to the invoker client.

The interface “@post(/login)” processes login requests. A login request includes a username and hashed password value. A token of current timestamp will be generated



Username	Password Hash
Alice	pbkdf2:sha1:1000\$vAoH1IpY\$eacaa078e380bcf979c16d6c3ad3b275f880b7f1

Figure 4.10: Hashed Passwords in Database

and returned if login information is authenticated or an error code for failed login trials. The users RSA key pair is returned as well if login succeeds.

The interface "@post(/register)" processes registration requests. All the user information includes username, password and RSA key pair are generated and partially encrypted by the Secure Dropbox client. The KMS server will only check if all the fields meet certain requirements and then perform the storage procedure. Error code returned as registration result.

4.3.3 KMS Key Management

Uploading a new file via Secure Dropbox client is consists of two steps: The first one is encrypting the file and synchronizing it to Dropbox. The second one is uploading the encryption key and other data that involved in the first step to Secure Dropbox KMS. KMS key management module is implemented oriented to the second step.

The interface "@post(/upload_doc_key_chain)" receives newly generated doc_id and corresponding doc_key. The record will be refreshed if there is one with same doc id exists in "key_chain" table. Otherwise the new doc_id and doc_key will be added to the "key_chain" table. Error code returned as uploading result.

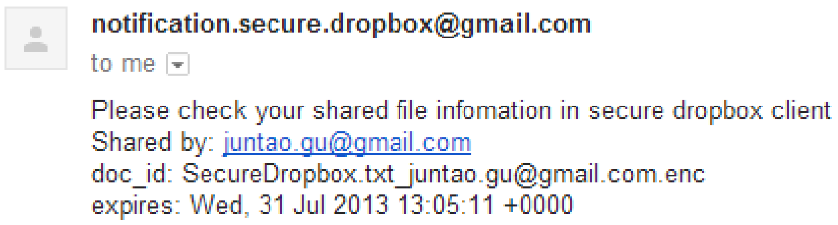
The interface "@post(/download_doc_key_chain)" returns the keychain which belongs to the user in a Python dictionary format with doc id as key and doc key as the value.

The interface "@post(/delete)" receives doc id to be deleted. Not only deleting the record in table key_chain, it will also trigger the deletion of the corresponding records in the "sharing_pool" table. For example, Alice has a file doc1_Alice.txt and shared it with Bob. If Alice deletes the record of doc1_Alice.txt in the "key_chain" table, the sharing information with Bob about this file will be deleted as well. Error code returned as deletion result.

The interface “@post(’/fetch_pub_key’)” returns the RSA public key of the user whose name is specified in the request. It happens before users want to share files with the potential sharing recipient. The sharing sponsor has to fetch the recipients public key to encrypt the file encryption key before generating that sharing record in “sharing_pool” table. Although the public key is stored in the “user” table, it is still designed as part of key management since the RSA key pairs are not involved in user management.

4.3.4 KMS File Sharing Management

The interface “@post(/share)” receives file sharing information and stores it into the sharing_pool table. If there is same sharing record in the database then only the doc_key field will be refreshed to the new file encryption key. Otherwise a new record will be added. An email notification towards the sharing recipient will be created by this method. The example sharing notification is generated as follows:

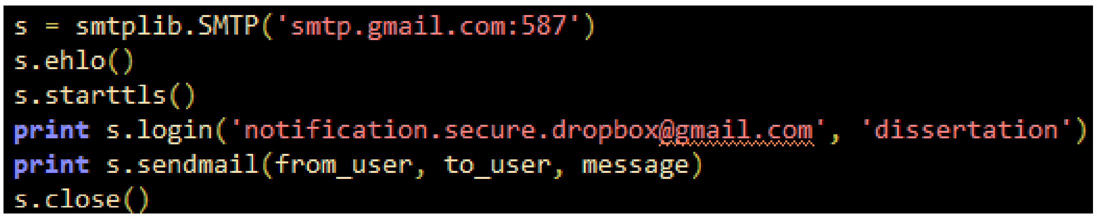


notification.secure.dropbox@gmail.com
to me

Please check your shared file infomation in secure dropbox client
Shared by: juntao.gu@gmail.com
doc_id: SecureDropbox.txt_juntao.gu@gmail.com.enc
expires: Wed, 31 Jul 2013 13:05:11 +0000

Figure 4.11: Sharing Notification

The notification function could be implemented based on Python smtplib module which defines an SMTP client session object that can be used to send mail.



```
s = smtplib.SMTP('smtp.gmail.com:587')
s.ehlo()
s.starttls()
print s.login('notification.secure.dropbox@gmail.com', 'dissertation')
print s.sendmail(from_user, to_user, message)
s.close()
```

Figure 4.12: Sharing Notification Implementation

- SMTP (‘smtp.gmail.com:587‘) generates a SMTP instance encapsulates an SMTP connection to Gmail.
- “ehlo()” identifies the local machine to an ESMTP server.
- “starttls()” puts the SMTP connection in Transport Layer Security (TLS) mode.
- “login()” logins on the Gmail SMTP server with authentication information.
- Send the mail via “sendmail()” method. The message has been generated in the context.

The interface “@post(/shared_file)” returns all the sharing records where the to_user field is the user who is calling this method. A Python dictionary includes doc id, processed encryption key, sharing sponsor, access URL and expiration is returned. This function should be called before the sharing recipient wants to read those shared files.

Since the file sharing URL generated by Dropbox will by default expire three hours later, the sharing recipient would get no access to the URL then. To reduce the confusion, a scheduled task is running as a thread in the server which refreshes the information in sharing_pool periodically. The task will delete the records in which the timestamp in expires field is later than current timestamp. The refreshing period is by default configured as 600 seconds and configuration interface is open to administrators.

4.4 Secure Dropbox Client Implementation

The secure Dropbox client performs cryptography related computations and communicates with both Dropbox and Secure Dropbox KMS simultaneously. In short, according to the architecture diagram above, the Secure Dropbox UI module is responsible for human computer interaction and Application initialization. The secure Dropbox module is the middle ware which controls the data communications and cryptography operations. The judgment of Secure Dropbox module is performed in Secure Dropbox UI module

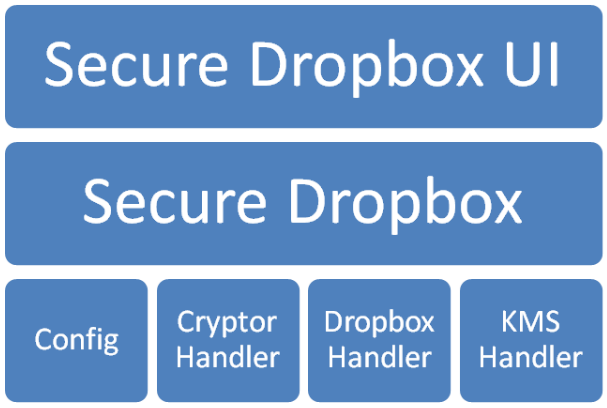


Figure 4.13: Secure Dropbox Client Architecture

during application initialization. The configuration module includes application environment configuration parameters and it is open to Secure Dropbox User. Cryptor Handler includes AES-256, RSA and other essential cryptography algorithm encryptor instances. Dropbox Handler creates a session handler which is used in Secure Dropbox module when Dropbox file operation or other communication is required. KMS Handler includes the method to communicate with Secure Dropbox KMS. Cryptor Handler is used in KMS Handler as well since some communications between client and KMS requires encryption. Since Secure Dropbox module is an integration of other infrastructure modules, the implementation details will be introduced at last after explaining of those infrastructures.

4.4.1 Secure Dropbox UI

Secure Dropbox is implemented as a command line based application since it performs better platform independency when GUI is not involved. Besides IO function, Secure Dropbox clients running environment prerequisites are detected and configured during UI initialization. Also Secure Dropbox running mode is decided. The detection procedure is as follows:

If the Dropbox official client is not installed on this computer, the Secure Dropbox client is not allowed to start since no file synchronization operation to Dropbox could be done. The Secure Dropbox will be configured as local mode if either Secure Dropbox

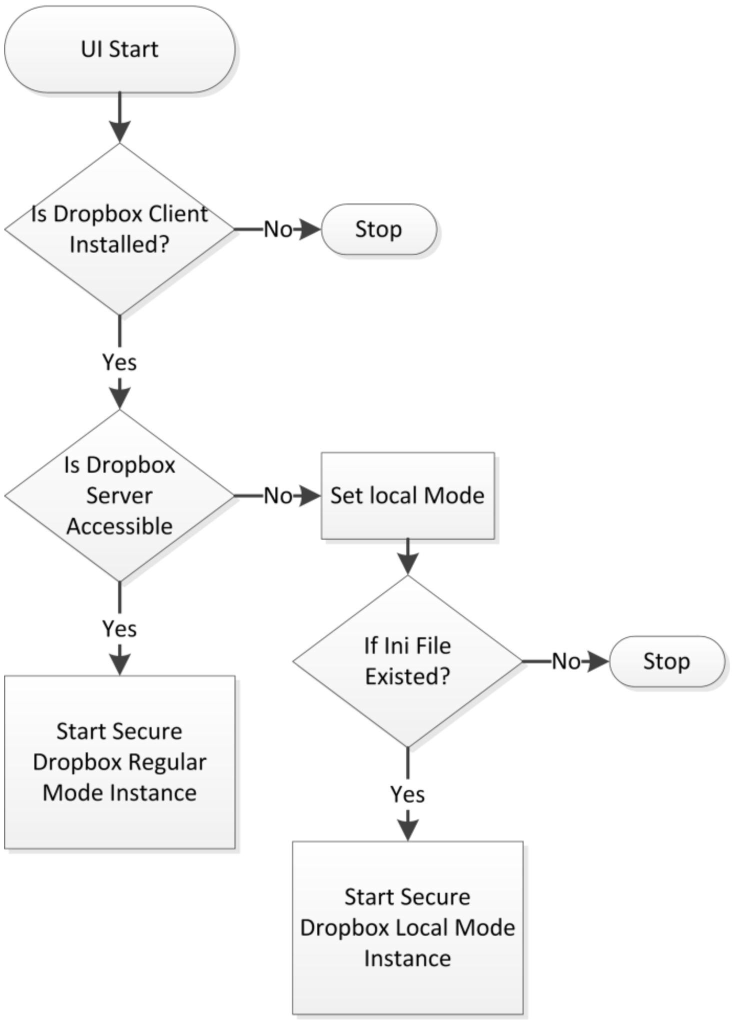
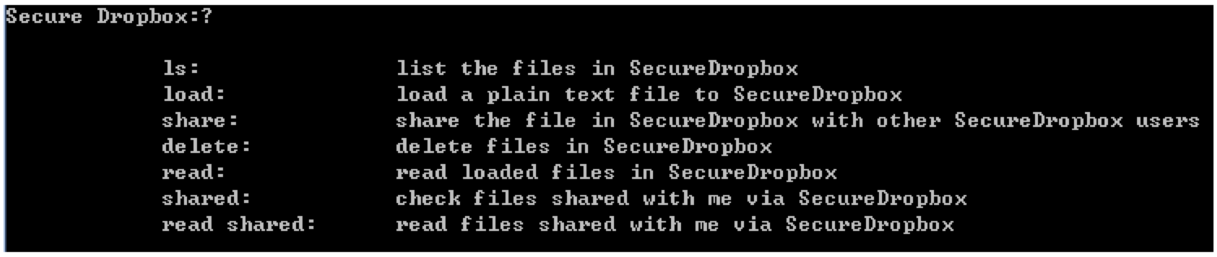

UI Start
Is Dropbox Client Installed?
No
Stop
Yes
Is Dropbox Server Accessible
No
Set local Mode
Yes
If Ini File Existed?
No
Stop
Start Secure Dropbox Regular Mode Instance
Yes
Start Secure Dropbox Local Mode Instance

Figure 4.14: Secure Dropbox Running Environment Detection

KMS or Dropbox server is not reachable. In this situation, if any ini file which works as a local KMS exists, the client will be configured and started in local mode. Otherwise both remote and local KMS will be considered as not available and consequently the Secure Dropbox service is not available. If both Dropbox server and Secure Dropbox KMS are both accessible, client will be initialized in regular mode.

The supported command is different under different mode. For example, under local mode, there are only two commands supported: “ls” to print local file list and “read” to read local files. However, the supported operations in regular mode are listed as follows:

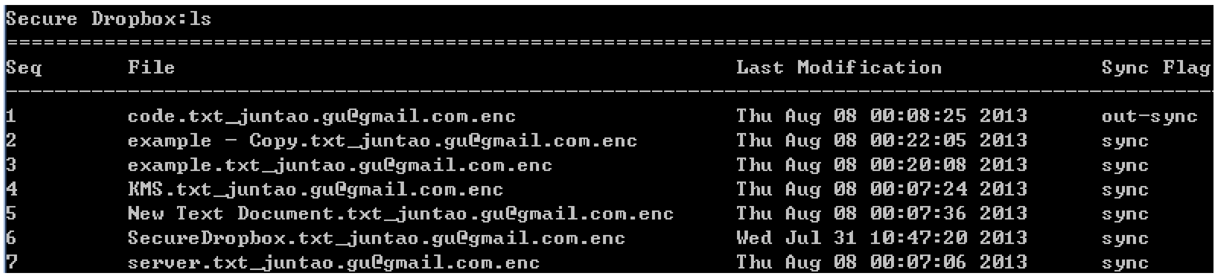
The following two examples are print information of command “ls” and “read”. The “ls” command lists all local files in the folder named Secure Dropbox in Dropbox appli-



```
Secure Dropbox:?

           ls:                list the files in SecureDropbox
           load:              load a plain text file to SecureDropbox
           share:             share the file in SecureDropbox with other SecureDropbox users
           delete:            delete files in SecureDropbox
           read:              read loaded files in SecureDropbox
           shared:            check files shared with me via SecureDropbox
           read shared:       read files shared with me via SecureDropbox
```

Figure 4.15: Secure Dropbox Commands

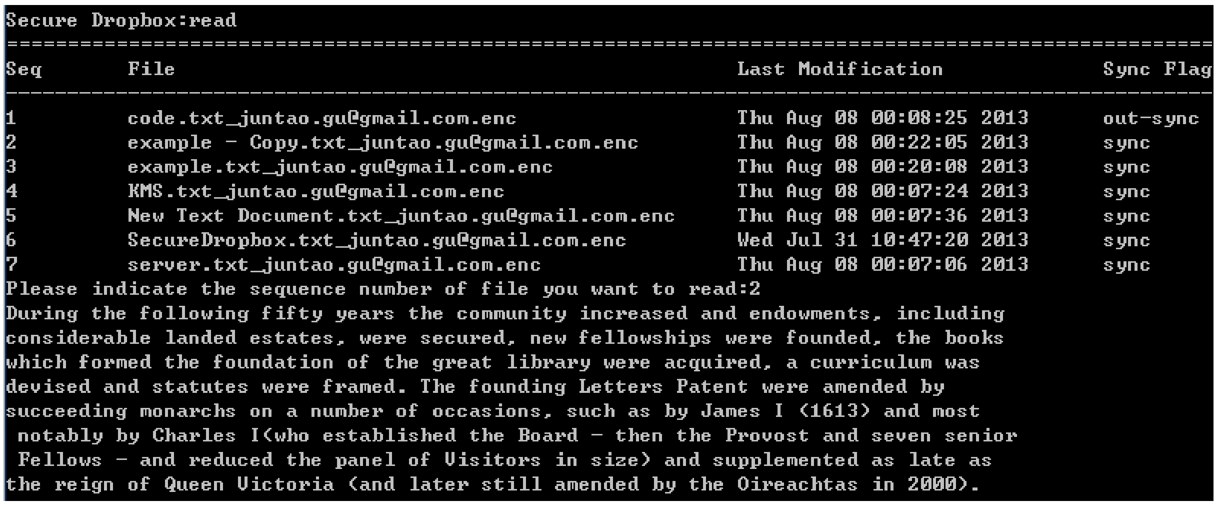
cation. The sync flag field indicates if the encrypted files encryption key could be found in KMS. It might be inappropriate operations that lead to files out-sync. The out-sync file could not be read since no encryption key could be fetched and applied. The second diagram shows the print information of "read" command. A "read" command automatically invokes the "ls" function to display the local file list at first and then show further prompts to guide user to input the sequence number of desired files to be read. It accepts sequence number and reduces the possibility of misoperations by typing file names. The file is stored in cipher locally but decrypted and printed in the same console. It means that the file encrypted by Secure Dropbox could be only read from the Secure Dropbox client. Most user interfaces are designed as the following styles:



```
Secure Dropbox:ls
================================================================================================
Seq       File                                                Last Modification          Sync Flag
------------------------------------------------------------------------------------------------
1         code.txt_juntao.gu@gmail.com.enc                    Thu Aug 08 00:08:25 2013   out-sync
2         example - Copy.txt_juntao.gu@gmail.com.enc          Thu Aug 08 00:22:05 2013   sync
3         example.txt_juntao.gu@gmail.com.enc                 Thu Aug 08 00:20:08 2013   sync
4         KMS.txt_juntao.gu@gmail.com.enc                     Thu Aug 08 00:07:24 2013   sync
5         New Text Document.txt_juntao.gu@gmail.com.enc       Thu Aug 08 00:07:36 2013   sync
6         SecureDropbox.txt_juntao.gu@gmail.com.enc           Wed Jul 31 10:47:20 2013   sync
7         server.txt_juntao.gu@gmail.com.enc                  Thu Aug 08 00:07:06 2013   sync
```

Figure 4.16: "ls" Command Print Information

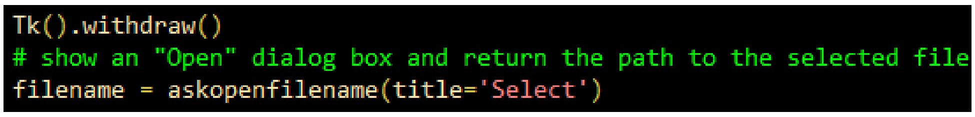
A file loading dialog is provided to choose the file which user wants to upload to Dropbox via Secure Dropbox. It is implemented based on TkInter, a standard Python interface to the Tk GUI toolkit and available on most UNIX platforms, as well as Windows. It facilitates the procedure of specifying the file to be loaded under command line. The return value of the file dialog is a full path of the selected file. Secure Dropbox currently only supports cryptographic operation of text files so any file selected without the suffix



```
Secure Dropbox:read
=========================================================================================
Seq       File                                              Last Modification          Sync Flag
-----------------------------------------------------------------------------------------
1         code.txt_juntao.gu@gmail.com.enc                  Thu Aug 08 00:08:25 2013     out-sync
2         example - Copy.txt_juntao.gu@gmail.com.enc        Thu Aug 08 00:22:05 2013     sync
3         example.txt_juntao.gu@gmail.com.enc               Thu Aug 08 00:20:08 2013     sync
4         KMS.txt_juntao.gu@gmail.com.enc                   Thu Aug 08 00:07:24 2013     sync
5         New Text Document.txt_juntao.gu@gmail.com.enc     Thu Aug 08 00:07:36 2013     sync
6         SecureDropbox.txt_juntao.gu@gmail.com.enc         Wed Jul 31 10:47:20 2013     sync
7         server.txt_juntao.gu@gmail.com.enc                Thu Aug 08 00:07:06 2013     sync
Please indicate the sequence number of file you want to read:2
During the following fifty years the community increased and endowments, including
considerable landed estates, were secured, new fellowships were founded, the books
which formed the foundation of the great library were acquired, a curriculum was
devised and statutes were framed. The founding Letters Patent were amended by
succeeding monarchs on a number of occasions, such as by James I (1613) and most
 notably by Charles I(who established the Board - then the Provost and seven senior
 Fellows - and reduced the panel of Visitors in size) and supplemented as late as
the reign of Queen Victoria (and later still amended by the Oireachtas in 2000).
```

Figure 4.17: "read" Command Print Information

.txt will be ignored. Also, for access control purpose, the file does not include current users account name will be ignored as well. The loading dialog only accepts selection of file instance but not the folder. It is implemented as follows:



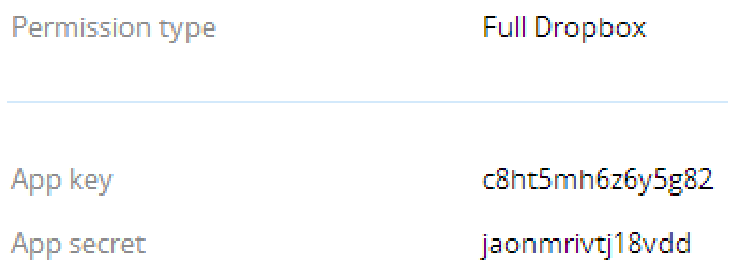
```
Tk().withdraw()
# show an "Open" dialog box and return the path to the selected file
filename = askopenfilename(title='Select')
```

Figure 4.18: File Loading Dialog Implementation

4.4.2 Dropbox Handler

Dropbox Core API is used in Secure Dropbox. Only applications that have registered and integrated with Dropbox application console could get an access token to use these APIs. The App key and App secret are generated after registration and they are also key identities when applying for the access token. There are two permissive types of operating Dropbox. Full Dropbox indicates everything inside users Dropbox folder is accessible by this application while Application Folder mode indicates the application can only access the specified folder with full permission. Secure Dropbox is using the Full Dropbox permit. Application settings of Secure Dropbox are listed as follows:

Dropbox Handler generates an access token via OAuth service of Dropbox. This is an



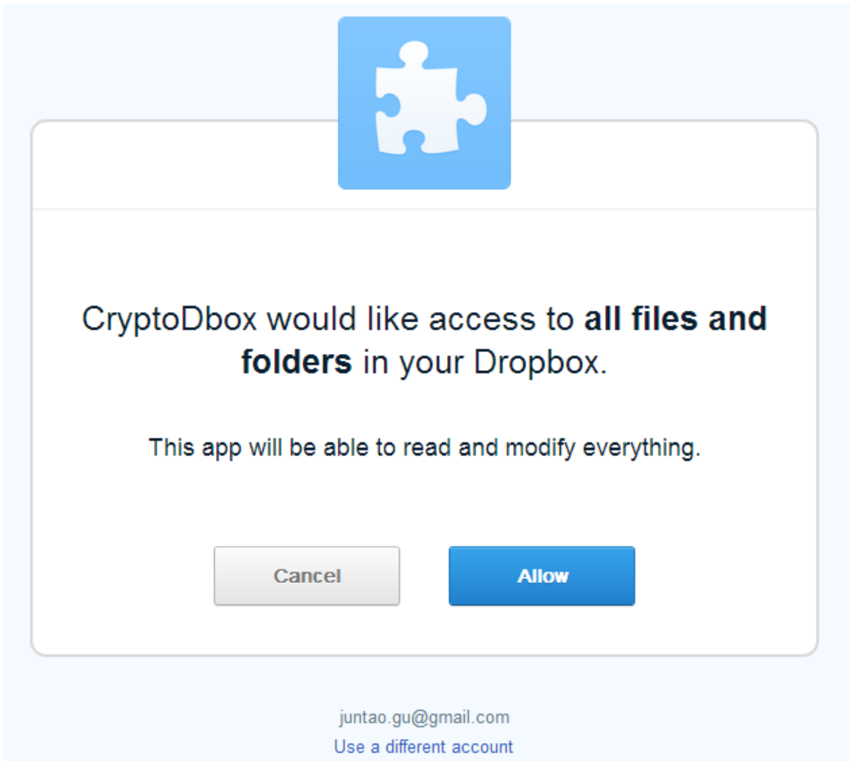
Permission type	Full Dropbox
App key	c8ht5mh6z6y5g82
App secret	jaonmrivtj18vdd

Figure 4.19: Application Settings of Secure Dropbox

authorization framework that allows a third-party application to obtain access permission for HTTP services that is using this framework. For example, to operate Dropbox, rather than inputting the Dropbox username and password to the third-party application, the user still authenticates on Dropbox and then an access token will be granted to this application if authentication succeeds. Significantly it reduces the security risks for Dropbox since it is difficult to audit if the third-party application code records the users username and password. In Secure Dropbox, the Dropbox authentication web page will pop up automatically and wait for users authorization action. The permission type is indicated as follows:

Access token will be granted after clicking Allow button. Although Dropbox Core API provides powerful interfaces that sufficient to implement any file operations, Secure Dropbox still relies on Dropbox official application with respect to all file operations for simplification of implementation and robustness of file synchronization mechanism. For now, only file sharing function in the Secure Dropbox client is performed through the Dropbox Core API by calling the "/media" interface. It gets a temporary authenticated URL for the target file. A Python dictionary with a URL and expiration information like:

The expiration time is consistent with the validation of OAuth access token to avoid the situation that the user whose last login has been expired but still able to get access to the file. Secure Dropbox stores this information and notifies sharing recipient. The sharing recipient gets file content by accessing the specified URL and read it after decryption


CryptoDbox would like access to **all files and folders** in your Dropbox.

This app will be able to read and modify everything.

Figure 4.20: Dropbox OAuth Service

```
{
'url': 'https://dl.dropbox.com/0/view/wvxv1fw6on24qw7/file.mov',
 'expires': 'Thu, 16 Sep 2011 01:01:25 +0000'
}
```

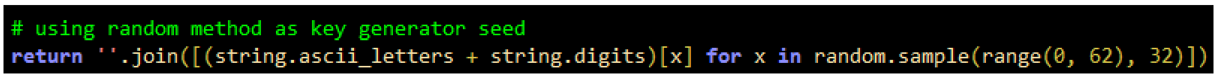
Figure 4.21: Return Value of Dropbox Core API

procedures.

4.4.3 Cryptor Handler

Cryptor handler includes implementation of AES-256. It also contains an encapsulated file encryption tool based on AES-256 that is frequently used in the client. The AES-256 is based on interfaces provided by PyCrypto module which also covers some other cryptography algorithms for the Python programming. AES-256 in Secure Dropbox is in CBC mode and configured with both trunk size and initial vector length in 16-bit.

RSA encryption/decryption handler is implemented based on M2Crypto. M2Crypto is another popular cryptography library for Python with better padding implementation of RSA algorithm. In Secure Dropbox, pkcs1_padding mechanism is adopted. AES-256 key generator is implemented in Cryptor Handler while RSA key pair is not generated here since logically it belongs to the registration procedure. Algorithm of generating random AES key is based on plain text file content as random seeds and implemented as follows:

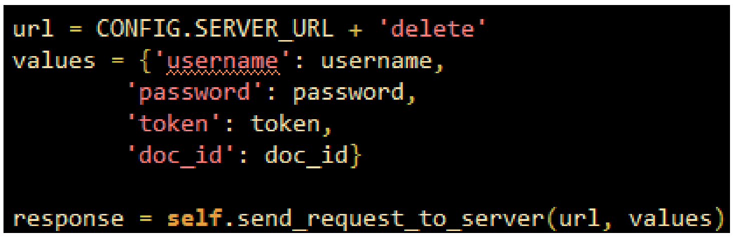


```
# using random method as key generator seed
return ''.join([(string.ascii_letters + string.digits)[x] for x in random.sample(range(0, 62), 32)])
```

Figure 4.22: AES Key Generation

4.4.4 KMS Handler

KMS Handler implements communication interfaces with Secure Dropbox KMS. Important features and implementation dependencies has been introduced in the Communication section. It includes a Cryptor instance for some cryptography operation involves in this module. Typically most methods in KMS Handler are related to generating data package to be sent to KMS based on data transferred from Secure Dropbox module. For example, to make a file deletion request to KMS, data package is generated and sent as follows:



```
url = CONFIG.SERVER_URL + 'delete'
values = {'username': username,
          'password': password,
          'token': token,
          'doc_id': doc_id}

response = self.send_request_to_server(url, values)
```

Figure 4.23: KMS Data Package Generation

- The URL is created with a fixed prefix “CONFIG.SERVER_URL” which includes KMSs IP address and listening port. “delete” specify the interface to call.

- Value to be sent is wrapped as a Python dictionary.

- "send_request_to_server" is called for sending requests. Return value is the response from KMS.

KMS interface invoker methods are implemented correspondingly. Some methods are called by Secure Dropbox Module directly while some else are performing as reusable infrastructure methods like "send_request_to_server()".

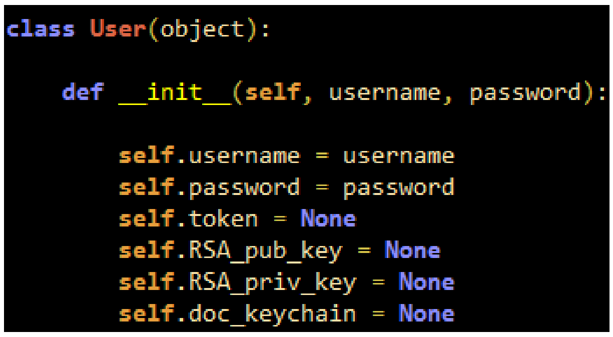
4.4.5 Configuration Module

The configuration module includes Dropbox application settings, running environment settings, cryptography parameters and communication error code. The same error code schema is defined in the KMS side as well so these indicators could be recognized by each other. Dropbox application settings include App key, App secret and access type. Server URL is specified with the combination of IP address, listening port of KMS and the other running environment include like default local location of encrypted files and local KMS file. Cryptography parameters like SHA1 iteration times and encryption block size is defined as well. Secure Dropbox users can change these macros in the configuration file to customize individual security requirement and other settings.

4.4.6 Secure Dropbox Module

SecureDropbox is the main function class in the application. This module includes the user information handler, the Dropbox handler, the KMS handler and the Cryptor handler. The functions provided by these handlers are integrated into Secure Dropbox module to enable Secure Dropbox performing as desired. For example, a file uploaded to Dropbox through Secure Dropbox is encrypted by "encrypt_file()" method of Cryptor handler, uploaded to Dropbox via access token generated by Dropbox handler. Moreover, its encryption key is uploaded to KMS by KMS handler as well. Secure Dropbox is created and manipulated by the Secure Dropbox UI instance. During initialization, it will create a folder in the Dropbox application named Secure Dropbox and all operations done by Secure Dropbox will be only in this folder.

Secure Dropbox has two operation modes. The regular mode works when a normal Internet connection is made. The local mode only supports reading local files. Besides different operations, the user authentication procedure is essentially different. The authentication process in regular mode is performed as described previously while local mode authentication is performed based on the local KMS file. This KMS file is created when login with regular mode and refreshed based on corresponding data in KMS. It actually stores an instance contains user account information, file encryption keychain and RSA key pair information. When the KMS server is not reachable, this information could perform as a read-only local KMS.

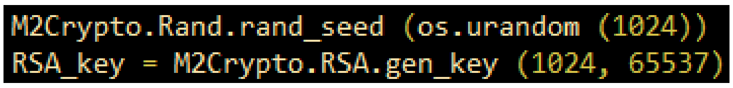


```
class User(object):

    def __init__(self, username, password):

        self.username = username
        self.password = password
        self.token = None
        self.RSA_pub_key = None
        self.RSA_priv_key = None
        self.doc_keychain = None
```

Figure 4.24: User Instance in Local KMS

The local KMS file is created by dumping a User class instance into a file and encrypting it with the users password. Inside the class instance, the password is stored in hash value and “doc_keychain” is encrypted with “RSA_priv_key” which is also ciphered with an AES key derived from password. In Python, to dump an object into a file, the module pickle is widely used. It implements a fundamental solution for serializing and de-serializing objects in Python. To read the local KMS file, a right password as decryption key must be used. Otherwise, the decrypted result of local KMS file will be meaningless that no user instance could be fetched. The input username and password will be matched with that in KMS file. Encrypted file encryption keychain and RSA key pairs will be decrypted and fetched after a successful authentication.

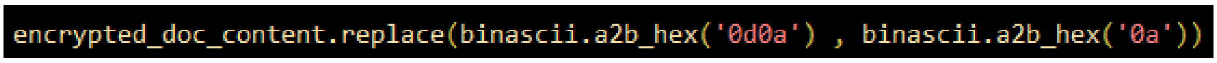
The RSA keys are produced in the Secure Dropbox module with certain interfaces of M2Crypto before uploading registration record. It is made and stored in pem format and then fetched and uploaded with private key encrypted. Coding implementation as follows:



```
M2Crypto.Rand.rand_seed (os.urandom (1024))
RSA_key = M2Crypto.RSA.gen_key (1024, 65537)
```

Figure 4.25: RSA Key Pair Generation

“/media” is used for generating file sharing record in Secure Dropbox while it causes different characters coding style on diverse of operating systems. In operating system like UNIX or other Unix-like operating systems, the newline character is coded as LF (“0x0a”). However, in Windows it is encoded as LF+CR (“0x0d0a”). The URL generated by “/media” points to the file data instance that is located on Unix-like file system on Dropbox which changes the windows coding style into Unix coding style. Since the encryption is done within Windows environment, the binary level change like losing several bytes in the cipher will lead to drastic influences on decryption result. To solve this problem, any “0x0d0a” should be replaced with “0x0a” in ciphers binary content as follows:



```
encrypted_doc_content.replace(binascii.a2b_hex('0d0a') , binascii.a2b_hex('0a'))
```

Figure 4.26: Replacements of Newline Characters

Chapter 5

Analysis

Since the security of Secure Dropbox relies on the nature of different cryptography which has been justified, this chapter will mainly outline the methodologies taken in evaluating the Secure Dropbox in terms of computation performance and user experience. The methods of data collection and analyses are explained and an overview of evaluation result will also be illustrated.

5.1 Evaluation Approaches

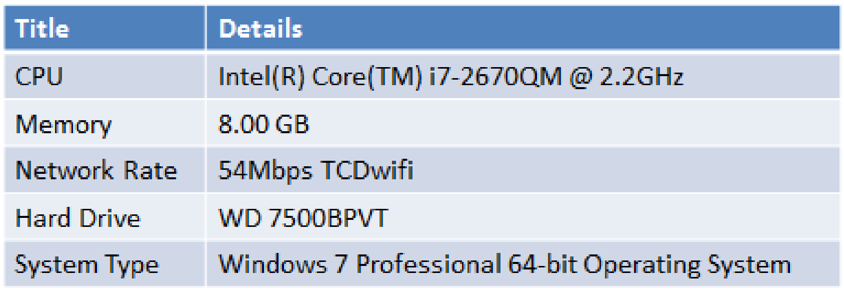
Feedbacks of the evaluation were sought from two categories of evaluators: Dropbox users with IT background and Dropbox users without any professional understanding of IT or computer science and work in IT unrelated industries. Performance evaluation involves the efficiency of cryptography operations where network situation sometimes matters. Efficiency of cryptography involved procedures will be evaluated according to time consumed to play cryptography operation. Network environment counts when using sharing service.

Secure Dropbox was designed to fortify the security confidence of Dropbox users. It is difficult to emulate the scenario that Dropbox is under internal attack practically so the design philosophy and security provided are explained from a mathematical perspective and reasoning of its working procedure. The result data is extracted from the participants oral expression about their thoughts of this application. Also, the frequency of misoperations is recorded with application context to illustrate the usability of Secure Dropbox.

5.2 Performance Testing

5.2.1 Testing Schema

Performance of Secure Dropbox reflected in file encryption and decryption time consumption. The test is conducted under the hardware configuration of:



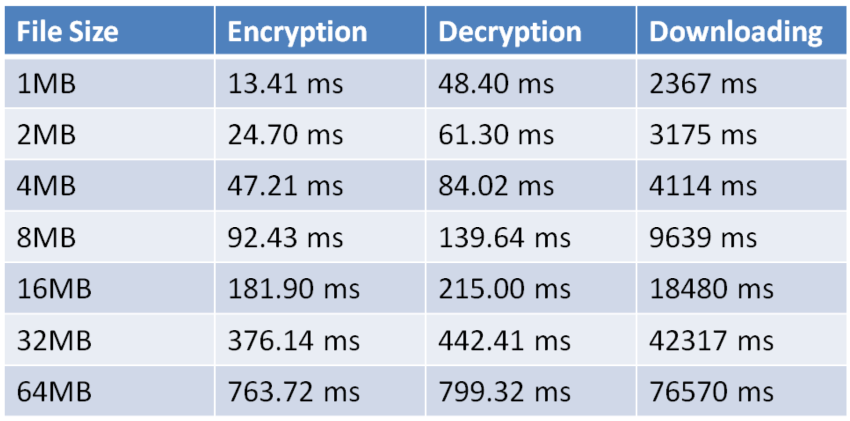
Title	Details
CPU	Intel(R) Core(TM) i7-2670QM @ 2.2GHz
Memory	8.00 GB
Network Rate	54Mbps TCDwifi
Hard Drive	WD 7500BPVT
System Type	Windows 7 Professional 64-bit Operating System

Figure 5.1: Testing Environment

To test the performance, several text files with different length from 1 MB to 64 MB have been created. Encryption time consumption is achieved during loading file into the Secure Dropbox procedure. The decryption time consumption is achieved from both procedures of reading local file and shared files. Downloading time is recorded during the shared file reading procedure. For values in the final performance record, each line is determined depends on average value of 100 times repeated test upon same file by performing Python scripts automatically. The results are listed as follows:

5.2.2 Performance Evaluation

On the basis of testing result listed above, Secure Dropbox could encrypt a 1 MB file within an average of 13.41 ms while this number comes to 763.72 ms when encrypting a 64 MB



File Size	Encryption	Decryption	Downloading
1MB	13.41 ms	48.40 ms	2367 ms
2MB	24.70 ms	61.30 ms	3175 ms
4MB	47.21 ms	84.02 ms	4114 ms
8MB	92.43 ms	139.64 ms	9639 ms
16MB	181.90 ms	215.00 ms	18480 ms
32MB	376.14 ms	442.41 ms	42317 ms
64MB	763.72 ms	799.32 ms	76570 ms

Figure 5.2: Secure Dropbox Cryptography Performances

file. It could be observed that the decryption roughly costs 35 ms more than encryption no matter what the file size is. The different time consumption could be attributed to the AES decryption preparation works like unpadding the cipher and extracting initial vector. However, in comparison of the time consumption of file synchronization operation when reading shared file, the decryption time period which takes almost single-digit-percentage only could be ignored. For example, it costs more than 1 minute to download a 64 MB file but the file could be decrypted in 800 ms. The result could be concluded as satisfying concerning local file cryptography operation while the network situation becomes the bottleneck when performing large shared file reading.

5.3 User Experience

To make the user experience evaluation a generic result, Dropbox users with or without IT background are both chosen as participants. In this section the feedback received from both of them will be summarized and discussed.

5.3.1 Session Procedure

Application configuration was conducted with delivering of packed Python egg of Secure Dropbox and readme file. Assistances were given when necessary. Users tested Secure

Dropbox on their PC or laptop. All the participants finished their sessions individually. During the session, participants were encouraged to finish several proposed tasks with Secure Dropbox client on their own. Participants with IT background were additionally assigned with tasks like Secure Dropbox configuring and performance evaluation. After user experience evaluation, participants were required to fill out a short questionnaire for collecting improvement comments. Users behavior confidence and time consumption on each task is recorded as well.

5.3.2 None IT Background Users Feedback

Most participants without IT background asked for help during Secure Dropbox configuration regarding command line operations and terminologies that involved. During testing, most of them were not skilled in command line operation style at first but got accustomed to it after several trials. Some participants forgot to press enter after finishing the OAuth procedure in the browser and instead just ignored the prompts and kept waiting in command line. All participants performed the file loading operation easily. Participant complained about that changing the source file requires another file loading operation is troublesome and easy to be forgotten. These users also commented with improvement suggestions that it would be better to make writing function integrated into the Secure Dropbox client as well so that extra file loading operations will not be essential. However they accepted the reason why this is difficult to make a generic editing interface. It was explained that different files call for different editing interface just like for ppt files Microsoft Office Power Point is required for editing. The most time consuming steps for these participants were registration and login since they were confused when typing the password to register or login but without echo on the screen. Some of them suspended the testing procedure and thought it was a program implementation problem. This is intentionally designed to imitate the way Linux does when inputting password information. Anyway it caused extremely high typo happening. Some participants proposed that there should be a contact list with permanent storage of sharing recipients Secure Dropbox account name in order to share a file. Also an interface to share files with a group of users should be implemented instead of performing sharing operations to each recipient one by one. One user advised that it would gain more confidence about his files security if Secure Dropbox could perform encryption on the same file but not on the copy in a Secure Dropbox folder. There will be only one encrypted file instance. Otherwise he had to look for other approaches to protect the plain text file. Though, he also showed his concern that he was not confident with leaving only one encrypted file instance and worried if encrypted file could not be decrypted properly. Finally, it was advised by most participants that they prefer operations in GUI or application provides a file system style user interface just like Dropbox application.

5.3.3 IT Background Users Feedback

Besides those relatively generic feedbacks, participants with IT background proposed more professional comments. Most of them advised that the file loading dialog should set file suffix filter to only display ".txt" files as Secure Dropbox only supports text file and other file input will be recognized as invalid operation. A few of them concerned that the OAuth procedure via browser and file loading diagram might causes availability problem in pure command line operating system like UNIX. Some participants thought an interface for querying file metadata in regard to cryptography like algorithm, key length and sharing records. Some else participants, who had been informed in advance that the Secure Dropbox disables the version control and file recovering service, advised that an operation logging system should be implemented for problem tracing in case of application errors. Some significant design defects and implementation advices have been reported during performance evaluation. Most participants thought the application crashed when reading the 64 MB shared files because there was only a blinking cursor but no any other progress indication. They advised that there should be a progress bar for time consuming operations like large file downloading or encrypting. When asked to configure the RSA key length, some participants thought the user experience should be improved. Rather than configuring by modifying the Python source code directly, an integrated configuration user interface would not only make configuration convenient and application robustness guaranteed. The configuration interface in application could play parameter checking in case of manual errors like incorrect key length is advised or illegal expressions or values for parameters. Most importantly, a fatal design defect was discovered. Encrypting a large file (i.e., 64 MB) costs less than one second while consumes far more time to synchronize file into Dropbox via Dropbox official client. However, Secure Dropbox is designed to upload file key encryption into KMS as long as the file has been encrypted. That time the file record could be queried in Secure Dropbox client but actually not in Dropbox yet. It works properly when performing a local reading since there is encrypted file instance in local file system while an exception occurs when trying to share the file because the encrypted file in Dropbox folder is not synchronized yet. It is because the /media interface called by Secure Dropbox to generate sharing URL cannot find the specified file which is still under synchronization. Another cryptography application design with security weakness was pointed out that once file sharing is cancelled or its URL is expired, the file encryption key should be changed in the meantime in case of someone could get the encrypted file instance and decrypt it without using shared file reading interface of Secure Dropbox client. There were implementation suggestions about integrating Secure Dropbox into file system so that it could be seamlessly used. For example, command "sdls" (stands for Secure Dropbox "ls") will be recognized as a default system command and it will directly list files loaded to Dropbox through Secure Dropbox.

5.4 Discussion of Feedback

The feedback received from the both kinds of participants with regard to security was primarily positive. They believe a client end encryption guarantees their file not accessible by Dropbox employees. Despite of those who have no concept about asymmetric cryptography, participants felt more confident to share file with Secure Dropbox in comparison of the plain text sharing by Dropbox.

Most of the criticism directed towards Secure Dropbox was concerned against the usability and design defects. Most participants thought the user interface should be improved no matter in GUI or command line for better user experience. The design flaws found during the performance test might be fatal factors to availability and security of Secure Dropbox. Future work on this application will be made based on these feedbacks.

Chapter 6

Conclusion

This chapter draws some conclusions from evaluation results achieved from user testing session and summarizes some criticisms of the project. The future work planned for the Secure Dropbox is then proposed.

6.1 Results

6.1.1 Was An Effective User End Encryption Tool Developed?

The best approach to appropriately illustrate the idea of user end encryption tools and asymmetric cryptography based secure sharing mechanism is to build a working prototype. Secure Dropbox accomplishes one and the most key ideas in accordance with security have been expressed. Secure Dropbox does not invent anything but applied a cryptography combination in a new application scenario. Its security is based on classic, strict, widely used and universally accepted cryptography algorithm. The design makes it zero-knowledge software: Secure Dropbox never has access to the file or even the encryption keys. Statically stored data that should be kept as secrets are unquestionably stored confidentially. It undeniably gains users confidence when using public cloud storage to store their confidential documentations according to the user testing feedbacks. Secure Dropbox users will never work with unencrypted files in their Dropbox which are ready to share with a small conversion upon the encryption key. Secure Dropbox could already be thought as a successful user end encryption tool to some extent.

However, it is still far from being a commercial software product. It provides a higher level of security but at the same time disables some important features of Dropbox. Actually, to most uncommercial users, the stability and fault-tolerance of a cloud storage are often considered as much as security. The trade-off will be absolutely there until these problems have been solved. Now Secure Dropbox is only implemented as prototype.

6.2 Criticism

6.2.1 Design Criticism

Although it explains the main idea of Secure Dropbox, the project is designed only for demonstration purpose. It lacks essential features that could actually make a better idea expression. For example, a logging module is always built in any software. The potential fault-tolerance and error recovering features of Secure Dropbox could have been illustrated by implementing an embedded logging module. It also lacks of consideration about availability such as it was designed to grant user with a minimum file reading permission in Secure Dropbox local mode. Although most timely Dropbox application based synchronization works robustly, no fault recovering or feedback is provided when errors occur. It is because Dropbox application is not designed to be based on by another application so there is no programmable interface provided. Dropbox core API is the only recommended way to implement a third-party application of Dropbox.

6.2.2 Implementation Criticism

The user space cryptography implementation essentially narrows down the supported file types by Secure Dropbox. For this stage only text file is supported and to support a new file type requires considerable efforts. A file system level encryption implementation would solve this problem. Additionally, a more flexible configuration interface should be made like allowing users to configure their own cryptography application schema based on different environment and practical requirements. The user experience should be improved by redesigning the user interface based on some human computer interaction principles and adding practical functionalities.

6.2.3 Testing Criticism

Performance testing lacks of comparison with other cryptography algorithms or under different running environment. Consequently these numbers do not speak much about the performance of Secure Dropbox. In addition, the user experience testing lacks of expert participants although some of them have computer science background. Security experts usually have a better understanding in this area and they are able to propose constructive expertise about such a software product.

6.3 Future Work

6.3.1 Version Control and File Recovering

Version control and file recovering service provided by Dropbox is disabled because there is no corresponding file encryption key version control module in Secure Dropbox. A possible design could be padding the file name with a timestamp and using this file name as the key of the version control table. This table records the history of certain file and its corresponding encryption key.

6.3.2 User Profile Management

With reference to user profile management, only the user registration and login have been implemented. The following jobs will be the implementation of a more generic user profile management module with common features like logout, account cancellation and more importantly password modification. The updating mechanism towards the expired RSA key pair which is also a part of user profile will be implemented as well.

6.3.3 Improvement of Sharing Mechanism

The file will keep using the same key until its source text has been modified and reloaded via Secure Dropbox again. For now file sharing does not change the file encryption key because the sharing URL could be cancelled or automatically expires after certain time slot. Also shared files could only be accessed after valid authentication and through Secure Dropbox user interface control. However, a new file sharing mechanism provides one time encryption key which guarantees a better security for the file. The file encryption key has been known to other users will no longer be available after sharing is cancelled.

6.3.4 File Sharing Refreshing

The file sharing URL generated by Dropbox expires in 3 hours. It is consistent with Dropbox OAuth access token because Dropbox does not allow third-party application who holding an expired access token could still access the shared file. To keep the file sharing until further cancellation, the file sharing record in the database should be refreshed periodically to update the previous URL and expiration timestamp. However, calling "/media" interface requires a valid token which has to be fetched by playing manual authentication. A proper mechanism of keeping the access token valid will call for more investigations.

6.3.5 File System Level Encryption

Secure Dropbox currently supports operation upon text file only because the implemented user space encryption could play file manipulations conveniently upon text files although it is sufficient as a prototype for demonstration. File system level encryption makes cryptography procedure transparent to user space applications. Dragging files into the specific folder with customized file system level system calls will trigger file encryption automatically and vice versa.

6.3.6 Configuration Interface

Now Secure Dropbox configuration could be carried out by changing parameters in Python source code. While an embedded configuration user interface could limit the options for configuration and execute parameter checking before the new configuration taking effect.

6.3.7 Multi-platform Implementation

There are lots of Dropbox users who want to synchronize their files between different terminals. Since Secure Dropbox is implemented with Python, it would not cost much effort to do the application transplantation between different operating systems. Python is also supported in some portable operating systems like Android and iOS. The difference to be concerned will be mainly about the computation and network capacity which impacts the performance significantly.

6.3.8 Better Local KMS Mechanism

Now Secure Dropbox local mode works based on KMS files stored in local file system which is generated after last login. It provides file encryption keychain and RSA key pair required in order to execute reading operation. Nevertheless, a better designed local KMS mechanism should guarantee Secure Dropbox users the same experience seamlessly as the regular mode does. An optimized local KMS mechanism is designed to perform delay tolerant KMS information updating when Internet access revives.

Appendix A

Implementation Module Dependence

Name	**URL**
PyCrypto	https://www.dlitz.net/software/pycrypto/
M2Crypto	http://chandlerproject.org/Projects/MeTooCrypto
Dropbox	https://www.dropbox.com/developers/core/sdks/python
PBKDF2	https://github.com/mitsuhiko/python-pbkdf2
Werkzeug	http://werkzeug.pocoo.org/
Bottle	http://bottlepy.org/docs/dev/

Appendix B

Task Description

1. Please register a Secure Dropbox account with your email address and login with this account.
2. Check the Secure Dropbox folder in Dropbox folder. Open the ini file includes your username with any text editor.
3. Load a text via Secure Dropbox with "load" command.
4. Check the Secure Dropbox folder in Dropbox folder again. You will see a file whose file name is made up with the files name you just loaded, your account name and end up with suffix enc. Open this file and see if it is encrypted.
5. Use read command in Secure Dropbox to read the file you just loaded.
6. Open another Secure Dropbox client. Register another Secure Dropbox account and login with that account.
7. Back to the first Secure Dropbox client. Use "share" command to share the file you just loaded with the new account you just registered.
8. Switch the later opened Secure Dropbox client. Use "shared" command to see if there is file shared with you and use "read shared" command to read the shared file.

9. Open use. Please do any operation you want with Secure Dropbox. Help information will display by using command “?”

Appendix C

Questionnaire

1. I can understand the theory of how Secure Dropbox performing secure storage and secure file sharing through background introduction. 1-5:
2. I think this application is easy to use. 1-5:
3. I think it is more suitable to technical users. 1-5:
4. I think functions in the application are easy to understand. 1-5:
5. I feel more confident when using Secure Dropbox rather than only Dropbox. 1-5:
6. I think I grasp the basic usage by following the task list. 1-5:
7. I think people will try to use this application for security. 1-5:
8. I will use this application frequently. 1-5:
9. Do you have professional IT background? Yes/No:
10. How often do you use Dropbox? 1-5:
11. I concern about security of Dropbox. 1-5:
12. I experienced security problems in Dropbox or any other cloud storage service.Yes/No:

If you have any comments on how you think about Secure Dropbox, please write them here:

Bibliography

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology," 2011.

[2] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 27–33, 2010.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, Apr. 2010.

[4] C. Cachin and A. Shraer, "Trusting the Cloud," *ACM SIGACT News*, vol. 40, no. 2, pp. 81–86, 2009.

[5] OPSWAT, "Security Industry Market Share Analysis," Tech. Rep. June, OPSWAT, 2011.

[6] D. Beaver, "Network security and storage security: symmetries and symmetry-breaking," *Proceedings of the First International IEEE Security in Storage Workshop (SISW02)*, 2003.

[7] P. S. Kumar, R. Subramanian, and D. T. Selvam, "Ensuring data storage security in cloud computing using Sobol Sequence," *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*, pp. 217–222, Oct. 2010.

[8] Intel, "What's Holding Back the Cloud ?," Tech. Rep. May, Intel, 2012.

[9] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, pp. 1–11, Jan. 2011.

[10] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A Logic-based Network Security Analyzer," *Security '05 Technical Program*, 2005.

[11] D. E. Rob, *Cryptography and Data Security.* Addison-Wesley Publishing Company, Inc., 1 ed., 1982.

[12] A. B. L. Blaze, Matt, "A Cryptographic File System for Unix," *1st Conf.- Computer & Comm. Security*, vol. 4, 1993.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, p. 29, Dec. 2003.

[14] Dropbox, "How secure is Dropbox?," 2013.

[15] M. de Icaza, "Dropbox Lack of Security," 2011.

[16] G. Dhillon and S. Moores, "Computer crimes: theorizing about the enemy within," *Computers & Security*, vol. 20, pp. 715–723, Dec. 2001.

[17] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic, "TrustStore: Making Amazon S3 Trustworthy with Services Composition," *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 600–605, 2010.

[18] R. Uppalli and C. Killian, "Analysis of techniques for building intrusion tolerant server systems," *IEEE Military Communications Conference, 2003. MILCOM 2003.*, vol. 2, pp. 729–734, 2003.

[19] S. Kamara and K. Lauter, "Cryptographic Cloud Storage," *IFCA/Springer-Verlag Berlin Heidelberg*, pp. 136–149, 2010.

[20] D. Xiaodong, S. David, and W. Adrian, "Practical Techniques for Searches on Encrypted Data ," *Security and Privacy, 2000. S\&P 2000. Proceedings. 2000 IEEE Symposium on*, 2000.

[21] R. Curtmola, J. Garay, and M. Hill, "Searchable Symmetric Encryption :Improved Definitions and Efficient Constructions," *CCS06, October 30November 3, 2006, Alexandria, Virginia, USA.*, pp. 79–88, 2006.

[22] Mfisk, "Why Map-Reduce?," 2013.

[23] Erik Riedel, Mahesh Kallahalla and R. Swaminathan, "A framework for evaluating storage system security," *FAST*, pp. 15–30, 2002.

[24] L. M. Vaquero, L. Rodero-Merino, and D. Morán, "Locking the sky: a survey on IaaS cloud security," *Computing*, vol. 91, pp. 93–118, Nov. 2010.

[25] Dropbox, "Dropbox Security Overview," 2012.

[26] M. Bellare, a. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pp. 394–403, 1997.

[27] NIST, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," *Federal Information Processing Standards Publication 197 November*, 2001.

[28] National Security Agency, "National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information," *CNSS Policy No. 15, Fact Sheet No. 1*, vol. 6716, no. 15, pp. 1–3, 2003.

[29] Elaine Barker, William Barker, William Burr, William Polk and M. Smid, "Recommendation for Key Management Part 1: General," *NIST Special Publication 800-57 Part 1, Revised*, 2007.

[30] B. Kaliski, “TWIRL and RSA Key Size,” 2009.

[31] R. C. Merkle, “A fast software one-way hash function,” *Journal of Cryptology*, vol. 3, pp. 43–58, 1990.

[32] BoxCryptor, “BoxCryptor,” 2013.