Chapter 4 Implementation

The previous chapter explained some of the 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 library and implementation platform will be introduced at first. Then the communication mechanism of Secure Dropbox KMS and Secure Dropbox client will be explained. The implementation of Secure Dropbox KMS, 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 programming will be definitely more efficient than user mode encryption but consequently might encounter compatibility problems under different platforms.One reason to choose Python as implementation language is it is an efficient programming language that is widely supported to run on most main stream operating system like Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines. Program once written could be executed everywhere as long as Python interpreter has been installed. Python is a high-level programming language and its libraries and syntax allows developers to implement certain function with fewer lines of code than it would be possible in lower level programming language like C, which increase the readability of code and boost up the programming procedure. Although always being claimed as a scripting language, the object-oriented feature makes Python competent in large scale program implementation. Another reason to choose Python because 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 while 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. Since Secure Dropbox is designed as a C/S architecture system, a frequently changed implementation platform may cause compatibility problem and make software update complicated. Also, Python is the advised programming by Dropbox to make use of their Dropbox Core API and some advanced features are currently only supported by Dropbox Python release like OAuthV2 which perform an indirect Dropbox Authentication during Python API programming. With regard to server end implementation, Python’s powerful web development frameworks like Django or web2py provides mature web development toolkits and libraries to use. Django even automatically provide 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. Python’s interpreter mechanism insures that as long as there is no operation system specified features used in code, program could be executed everywhere but with exactly the same result. An example of operation system specified feature could be illustrated with the following example which was actually encountered during the Secure Dropbox course project:

The schema to represent a path under Windows and Linux is like:

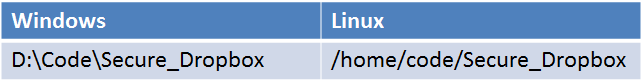


Figure 4.1 Representation of Path in Different Operation System

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 in Linux and for both of them there is a task about performing some file operations upon certain file under code/Secure\_Dropbox. When specifying the file path, either ‘/’ or ‘\’ may cause compatibility problem under different Operation system and lead to file path exceptions. So, to avoid such an operation system specified problem, in Python programmer could generate the path as:

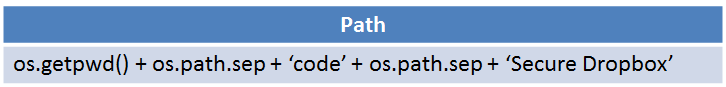


Figure 4.2 Platform Independent Coding

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

4.1.2 Secure Dropbox KMS

Just as Secure Dropbox client, Secure Dropbox KMS is implemented with Python 2.7.5 as well. It is implemented as a simple Restful WebService based on its desired features. The WebService implementation is based on Python bottle library which provides ready to use restful interface for the application. It has been deployed on Ubuntu Server 13.04. Ubuntu Server is currently the most popular guest server platform on the world’s leading public clouds, in terms of 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 compute capabilities.

The Ubuntu server instance with Secure Dropbox Running is currently deployed on Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 is a web service that 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 and provides the consumer with the capability of provision processing, storage management, network configurations, and other fundamental computing resources so that the consumer is able to deploy and run any software on the cloud. Although Ubuntu Server provides monitoring infrastructure, the monitoring console from Amazon EC2 is more intuitive and comprehensive. The sample server monitoring service is provided as:

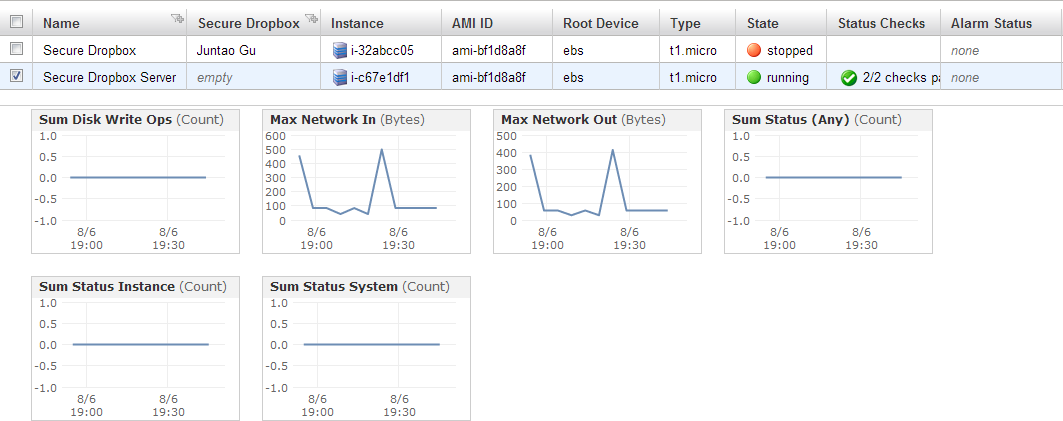


Figure 4.3 Amazon EC2 Server Monitoring

Amazon EC2 also provides 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 divided into inbound and outbound direction but both directions are totally free 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 0.0.0.0/0 by which means no access control rules on this port:

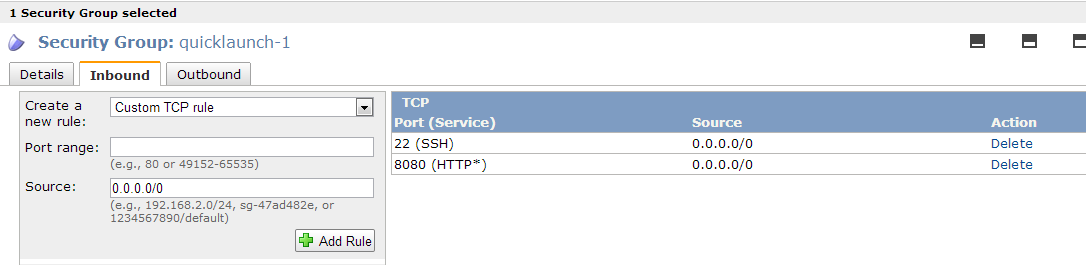


Figure 4.4 Amazon EC2 Security Group Customization

To SSH into Ubuntu Server instance on EC2, An identity based authentication is required. For each cloud application deployed on EC2, a private-key file would be delivered to administrator which associated with the instance. There is no account information based authentication on Ubuntu Server as long as it is deployed on EC2 but the user is by default granted with the root permission if the private-key file is authenticated. The following figure indicates the procedure of SSH to Amazon EC2 instance from Trinity College Dublin School of Computer Science and Statistics’ Turing service. The SSH command includes the private-key TRY.pem, the default username ‘ubuntu’ and the elastic IP Address of Ubuntu Server Instance. The prompts from the Ubuntu instance have no illustration about any Amazon EC2 information so it is actually like using a local Ubuntu Server to server administrators. Some basic monitoring information as current processes number, CPU load and Memory usage has been displayed as well.

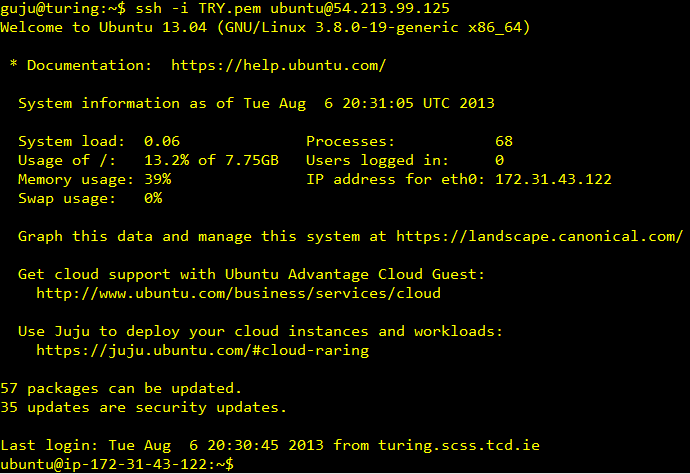


Figure 4.5 Amazon EC2 Instance SSH Procedure

Secure Dropbox KMS uses SQLite as database. SQLite is a SQL database engine provides file based storage service and build 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 could access to the SQLite database directly.

4.2 Communication

Basically, except the encrypted file instance itself, all the information involved in Secure Dropbox client are downloaded from KMS during initialization and any newly generated data have to be uploaded and permanently stored in KMS. For example, before the authentication procedure, the password hash algorithm, salt and iteration times have to be downloaded and the locally hashed password has to be uploaded to KMS again for matching purpose. The communication between Secure Dropbox client and Secure Dropbox KMS is performed in Restful WebService Style. A typical REST-style architecture generally consists of [client](http://en.wikipedia.org/wiki/Client_(computing)) end and server end. The client initiates by sending requests to the server. Having received the request, server will get it processed and return desired or corresponding responses to client. Usually rest requests and responses are generated aimed at the transmission of representations of certain resources. Resource to be sent can be essentially any comprehensible and meaningful concept that may be addressed by both ends. The [representation](http://en.wikipedia.org/wiki/Representation_(systemics)) of a resource is typically a formatted text or document that contains the state or value of certain resource. In Python programming, JSON is the standard which always used for formatting the data to be sent.

To build a Rest WebService Server, The python web framework Bottle [1] is used. Bottle is a fast, simple and lightweight WSGI micro web-framework for Python. To generate a request routine, a tag with http request method type and desired routine name are required as follows:

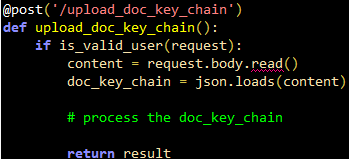


Figure 4.6 Restful WebService Programming with Bottle

This code paragraph indicates:

* This function processes a POST request when the url points to /upload\_doc\_key\_chain
* The function upload\_doc\_key\_chain() would be called when such a request arrives
* The resource content needs to be transmitted and processed is encapsulated in HTTP body
* The resource content has been formatted in JSON so it could be retrieved again if loaded in JSON format
* The return value would be directly returned to client end

In the client end, the request is generated just as building an http request with required resource data. It could be done with the support of urllib2 which is a standard module in Python 2.7 that defines 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:

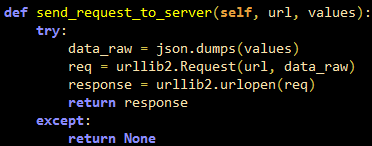


Figure 4.7 HTTP POST Request Generation

This code paragraph indicates:

* The data to be transmitted has to be formatted into JSON format
* HTTP POST request is generated with data and target url
* Request is sent via urllib2.urlopen() method and response data returned

All communication sessions are initiated by Secure Dropbox client end. Also these data exchange is designed with no state that any communication step is atomic and finished in a single session. The request about fetching data would get a response of data instance while request about uploading KMS data would get an error code depends on if the operation succeeded on 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 user’s registration request and login request. Key management mainly performs a CRUD operation upon any key related issues like doc keychain related request and RSA key pair related request. File sharing management is responsible for file sharing notification and a time task to handle the expired file sharing information. The implementation of some important features and WebService interface would be explained.

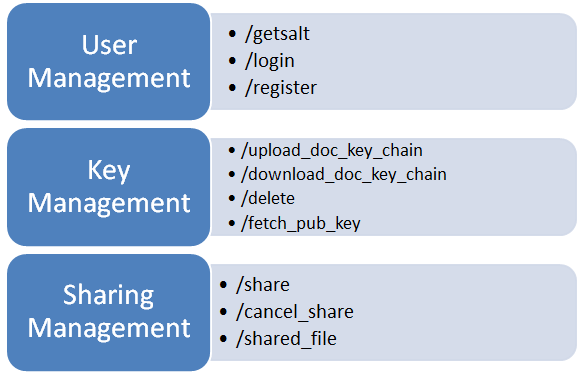


Figure 4.8 KMS Interfaces

4.3.1 Database Design

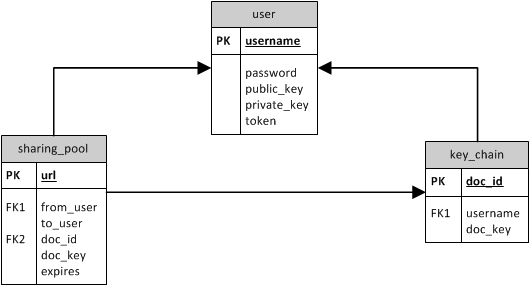


Figure 4.9

There are three tables in database of Secure Dropbox KMS. The table user stores user’s username, password and RSA key pairs. The public key is recorded in plaintext while the private key is stored with encryption. When users require public key of other user’s before sharing a file, it is fetched from this table. Token is a timestamp to indicate the last login time. The key\_chain table includes each user’s doc 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 owner’s username which is unique defined as public key in the table user. Each file has its doc\_id and corresponding doc\_key which is essentially an AES-256 key. The doc\_key is encrypted with owner’s 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 recipient’s public key so that the shared doc\_key could be decrypted by recipient’s 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 would the URL turns into not accessible. There is a scheduled task handler the records and delete those expired ones in sharing\_pool because certain sharing information becomes useless after getting expired and logically they should not be able to searched and displayed to user.

Although stored in the database of KMS, none of them is generated KMS but all generated and uploaded by Secure Dropbox client. Some essential fields are encrypted with the key not known to KMS but only known to account host.

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 so that client is able to hash the plaintext password with the same parameters as how was the hashed password value stored in KMS database generated. It is called before client performing the login procedure. For example, hashed password is stored as:



Figure 4.10 Hashed Passwords in Database

The function will returns 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 username and hashed password value. A token of current time stamp will be generated and returned if login information is authenticated while error code for failed login trials. The user’s RSA key pair would be returned as well if login succeed.

The interface @post (‘/register’) processes registration requests. All the user information include username, password, RSA key pair are generated and partially encrypted by the Secure Dropbox client while KMS server only checks if all the field meets certain requirements and then performs storage procedure. Error code returned as registration result.

4.3.3 KMS Key management

Uploading a new file via Secure Dropbox client has two steps: The first one is encrypting the file and updating it to Dropbox. The second one is uploading encryption keys and other data that involved in 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. If there is record with same doc id exists in key\_chain table, it will get refreshed. Otherwise the new doc\_id and doc\_key will be added to the table key\_chain. Error code returned as uploading result.

The interface @post ('/download\_doc\_key\_chain') returns the key chain which belong to the user in a python dictionary with doc id as key and doc key as 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 corresponding record in the table sharing\_pool. 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 table key\_chain, the sharing information with Bob in terms of this file will be deleted as well. Error code returned as deletion result.

The interface @post ('/fetch\_pub\_key') returns recipient's RSA public key request. It usually happens before user want to share a file with the potential sharing recipient so he has to fetch the recipient’s public key to encrypt the doc key before generating that sharing record in sharing pool. Although the public key is stored in 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 database. If there is same sharing record in the database then only the doc key field will be refreshed to the new doc key. Otherwise a new record will be added into database. An email notification to the sharing recipient would be made by this method. The example sharing notification is generated as follows:



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.

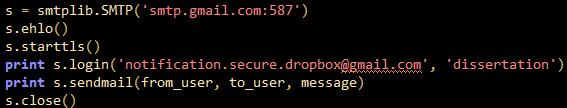


Figure 4.12 Sharing Notification Implementation

* SMTP (‘smtp.gmail.com:587’) generates a [SMTP](http://docs.python.org/2/library/smtplib.html#smtplib.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 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 include doc id, doc key, sharing from, url and expiration would be returned. This function should be called before the sharing recipient want to read those shared file.

Since the file sharing url generated by Dropbox will by default expires three hours later, the sharing recipient would get no access to the url then. To reduce the confusion, a scheduled task has been running as a thread in the server which refreshes the information in sharing\_pool frequently. The task will delete the record in which the time stamp in expires field is later than current timestamp. The refreshing period is by default configured as 600 seconds but configuration interface is open to administrator.

4.4 Secure Dropbox Client Implementation

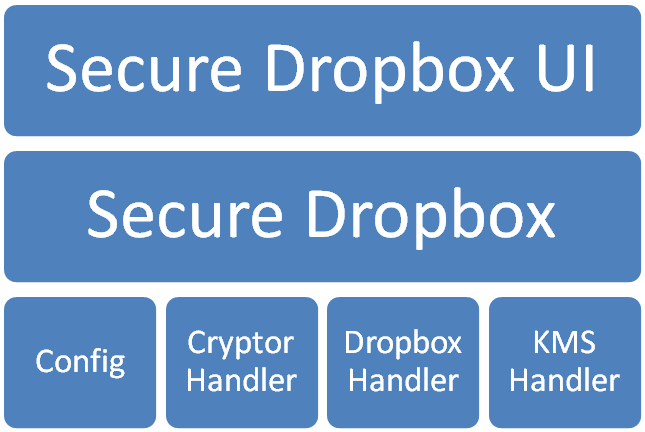


Figure 4.13 Secure Dropbox Client Architecture

Secure Dropbox client performs all cryptography related computation and communicates with both Dropbox and Secure Dropbox KMS simultaneously. In short, according to the architecture diagram above, Secure Dropbox UI module is responsible for human computer interaction and Application initialization. Secure Dropbox module is the middleware which controls the data communications and cryptography operations. Secure Dropbox has two modes: the regular mode is activated when there is Internet access is available while local mode is activated when there is no Internet access or either Dropbox Server or Secure Dropbox KMS is not reachable. The judgment is performed in Secure Dropbox UI module during application configuration. Config module includes application environment configuration 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 and the handler 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 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 independent availability when GUI is not involved. Besides IO function, Secure Dropbox client’s running environment prerequisites are detected and configured during UI initialization and whether Secure Dropbox is running under regular mode or local mode is decided as well. The detection procedure is as follows:

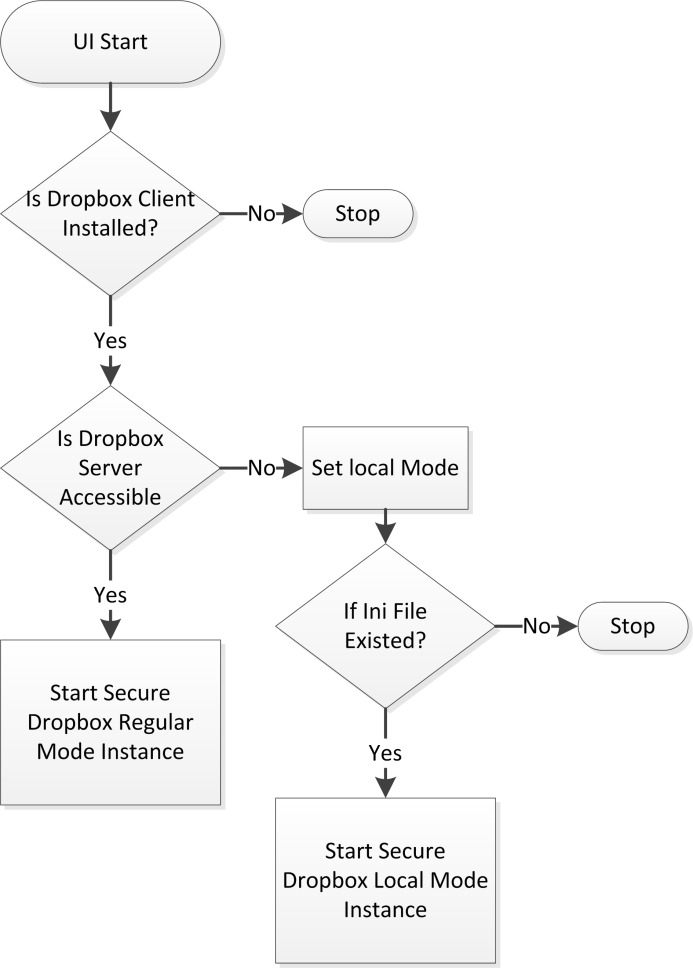


Figure 4.14 Secure Dropbox Running Environment Detection

If Dropbox official client is not installed in this computer, the Secure Dropbox client is not available then since no file synchronization operation to Dropbox could be done. Then as long as one of Dropbox server and Secure Dropbox KMS is not reachable, the Secure Dropbox will be configured as local mode. 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 it is not available since both remote and local KMS is not available and consequently the Secure Dropbox service is not available. If both Dropbox server and Secure Dropbox KMS are 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. It is because when remote KMS is not accessible, any encryption key or file sharing related operation could not be recorded permanently. The supported operations under regular mode are listed as follows:

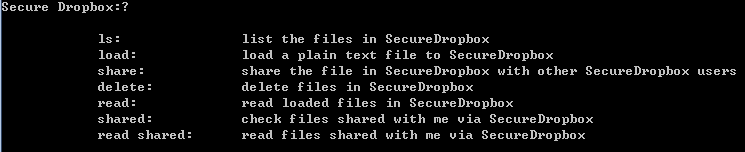


Figure 4.14 Supported Commands by Secure Dropbox

The following two examples are printed information of command ls and read. The ls command lists all local files in a folder named Secure Dropbox inside Dropbox application. The sync flag field indicates if the encrypted file’s encryption key could be found in KMS. It might be inappropriate operations leads to a file is out-sync. The out-sync file could not be read since no encryption key could be fetched and applied. The second diagram shows the printed information of read command. Read command automatically invokes the ls function to display the local file list at first and then show further prompts to guide user input the sequence number of desired file to read. It accepts sequence number and reduces the possibility of misoperations by typing file name. File is stored in cipher locally but would be decrypted and printed in the same console. It means that the file encrypted by Secure Dropbox can be only read from Secure Dropbox client. Most user interfaces are designed as the following styles.

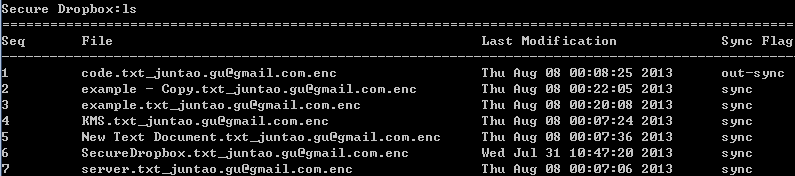


Figure 4.15 ls Command Print Information

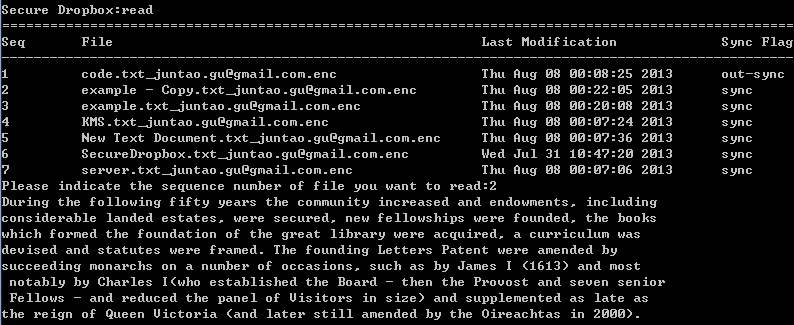


Figure 4.16 read Command Print Information

A file dialog is provided to choose the file which user wants to upload to Dropbox via Secure Dropbox. The file dialog is implemented based on TkInter which is the standard Python interface to the Tk GUI toolkit and available on most UNIX platforms, as well as on Windows. It facilitates the procedure of specifying file to be load under command line. The return value of the file dialog is a full path of the selected file. Because Secure Dropbox currently only support cryptography operation upon text files so if any file selected without the suffix .txt would be ignored. Also, for access control purpose, the file does not include current user’s account name will be ignored as well. The dialog only accepts selection of file instance but not folder. It could be implemented as follows:



Figure 4.17 File Dialog Implementation

4.4.2 Dropbox Handler

Dropbox Core API is used in Secure Dropbox. Only applications that has 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 permission type of operating Dropbox. Full Dropbox indicates everything inside certain user’s Dropbox is accessible by this application while Application Folder mode indicates the application can only access the single specified folder but still with full permission. Secure Dropbox is using Full Dropbox permission. Application settings of Secure Dropbox are listed as follows:

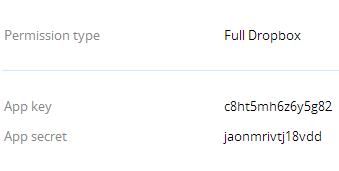


Figure 4.18 Application Settings of Secure Dropbox

Dropbox Handler generates an access token via OAuth service of Dropbox. This is an authorization framework that allows a third-party application to obtain access permission to HTTP services that is using this framework. For example, to operate Dropbox, instead of input the Dropbox username and password to the third-party application, user still authenticate on Dropbox but 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 user’s username and password. In Secure Dropbox, the Dropbox authentication webpage will pop up automatically and wait for user’s authorization action. The permission type is indicated as follows:

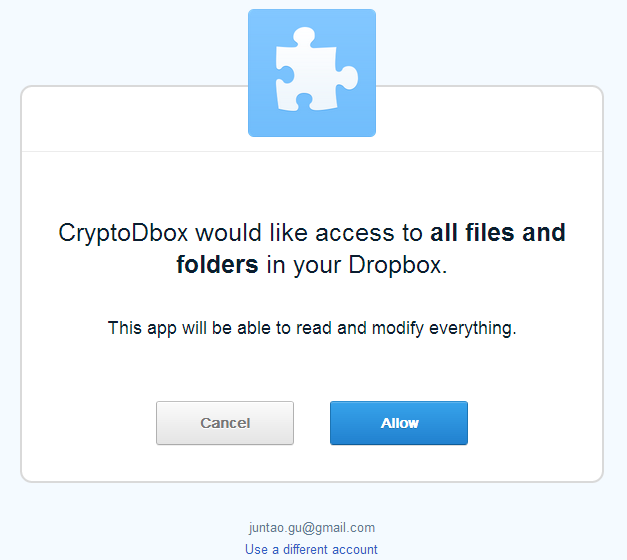


Figure 4.19 Dropbox OAuth Service

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

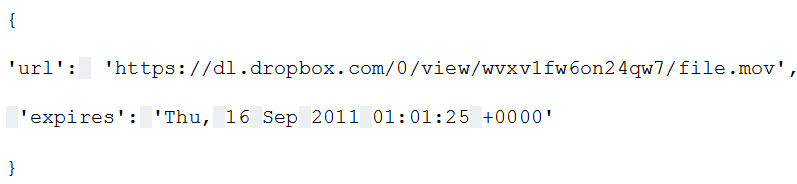


Figure 4.20 Return Value of Dropbox Core API /media

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

4.4.3 Cryptor Handler

Cryptor handler includes implementation of AES-256 and encapsulated as file encryption method that is frequently used in client. The AES-256 is based on interfaces provided by PyCrypto module which contains various cryptography algorithms for the Python programming language. AES-256 in Secure Dropbox is in CBC mode and encrypted with trunk size and initial vector in 16bit. RSA encryption / decryption handler is implemented based on M2Crypto. M2Crypto is another popular cryptography library for Python with better padding implementation in terms of RSA. 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 that should belong to registration procedure. Algorithm of generating random AES key is based on plaintext itself as random seeds and implemented as follows:



Figure 4.21 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 Communication section. It includes a Cryptor instance for some cryptography usage in this module. Typically most method in KMS Handler is related to generate data package to be sent to KMS based on data transferred from Secure Dropbox Module who takes control of it. For example, to make a file deletion request to KMS, data package is generated and sent as follows:

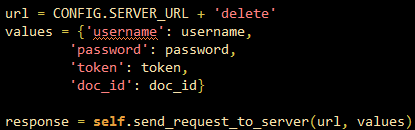


Figure 4.22

* url is created with a fixed prefix CONFIG.SERVER\_URL which includes KMS’s IP address and listening port. ‘delete’ specify the interface to call.
* Value to be sent is wrapped as a Python dictionary variable.
* send\_request\_to\_server is called for sending request and return result.

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 Config module

Config module includes Dropbox application settings, running environment settings, cryptography parameters and communication error code. The same error code schema is defined in 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 and listening port of KMS and other running environment include suffix of encrypted file and ini file. Cryptography parameters like SHA1 iteration times and encryption block size is defined as well. Secure Dropbox users can change these macro in Config 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 control, Dropbox handler, KMS handler and a Cryptor handler for the everywhere usage of cryptography operations. The functions provided by these handlers are integrated in Secure Dropbox class to make Secure Dropbox performing as desired. For example, an file uploaded to Dropbox through Secure Dropbox is encrypted by encrypt\_file() method from Crypto handler, uploaded to Dropbox via access token generated by Dropbox handler and get its encryption key uploaded to KMS by KMS handler. Secure Dropbox is created and manipulated by Secure Dropbox UI instance. During initialization, it will create a folder in Dropbox application named Secure Dropbox and all operations done by Secure Dropbox will be only inside this folder.

Secure Dropbox has two operation modes. The regular mode works when normal Internet connection is made while 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 a local ini file. This ini file is created or refreshed when login under regular mode based on corresponding data in KMS. It actually stored an instance contains user account information, existing doc keychain and RSA key pair information so that when KMS server is not reachable, these information could perform as a read-only local KMS.

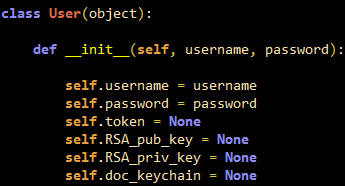


Figure 4.23 User Instance Stored in Ini File

The ini file is created by dumping a User class instance into a file and encrypting it with user’s password. Inside the class instance, password is stored in hash value and doc\_keychain is encrypted with RSA\_priv\_key which is also ciphered with an AES key derives from password-exact the same security mechanism as Secure Dropbox KMS. In Python, to dump an object into a file, the module pickle is usually used. It implements a fundamental solution for serializing and de-serializing object in Python. To read the ini file, a right password as encryption must be used. If the password is incorrect, the decryption result of ini file will be chaos and no user instance could be fetched. The input username and password will be matched with that in ini file and then those encrypted doc keychain and RSA key pairs will be decrypted for using.

RSA keys are produced in Secure Dropbox module since this procedure logically belongs to users account information generation but not encryption. The RSA key pair is created before uploading user registration record with certain interface of M2Crypto module. RSA key pair is made and stored in pem format and then fetched and uploaded with private key encrypted. Coding implementation as follows:



Figure 4.24 RSA Key Pair Generation

/media is used for file sharing in Secure Dropbox while it causes different coding style under diverse of operating systems. In operating system like Unix and any other Unix-like systems the newline character is coded as LF (‘0x0a’) while in Windows it is encoded as LF+CR (0x0d0a). The url generated by /media points to file data instance which is located in Unix-like file system of Dropbox so it changes the windows coding style into Unix coding style. Since the encryption is done within Windows environment so a binary level change like losing several bytes in the cipher will have drastic influence on decryption result. To solve this problem, any ‘0x0d0a’ would be replaced with ‘0x0a’ in cipher’s binary content as follows:



Figure 4.24 Replacements of Newline Characters

[1] <http://bottlepy.org/docs/dev/>