Indian Institute of Technology Bombay

Software Systems Lab

Secure Personal Cloud

Author:
Yash Jain
Yash Khemchandani
Anshul Nasery

Supervisor:
Prof. Soumen Chakrabarti

Team YAY

170050025(33.33%),170050055(33.33%),170070015(33.33%)

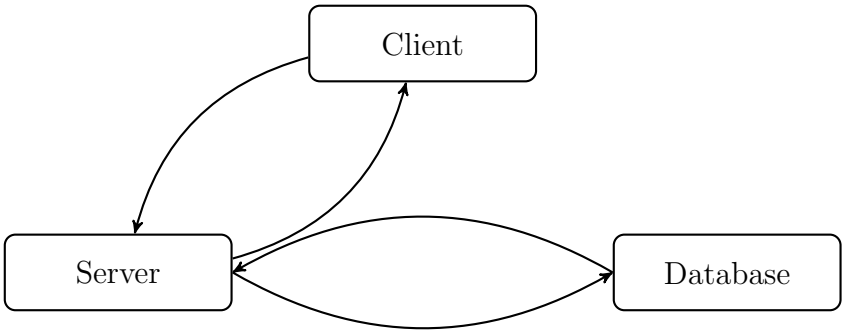
November 24, 2018

Declaration

I acknowledge and understand that plagiarism is wrong. This project is my own work, or my group's own unique group project. I acknowledge that copying some-one else's work, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.

1 Introduction

Secure Personal Cloud is a file storage and sharing system designed with security being the primary concern. The user has complete control over the entire encryption schema. The project consists of a Linux client and a web client. The Linux client does the main task of uploading, downloading and syncing the data between the server and the client, including several other features such as periodic sync, conflict resolution and md5sum check on upload and download both. The data is stored in an encrypted form in the Database. The Web Client is only used for viewing the data stored on the server.



2 Features

2.1 Linux

2.1.1 Syncing

Sync essentially backs up all the data of the user's directory to the cloud, and downloads all of the user's cloud's data to the observed directory.The uploading, downloading and conflict resolution is done through repeated API

calls to access and modify the contents of the database. The APIs are secured through a tokenization system. Syncing first gets all the paths and md5sums of the files of the user stored on the server, then compares them with the files already present in the directory.

SPC then encrypts the files on the machine through `openssl` Linux tool, (which converts the encrypted file to base64 text files) and sends them to the server, along with the md5sum of the unencrypted file and its relative path. Then it downloads the files absent on the machine, and decrypts(using the same tool) them.

2.1.2 Database design choices

Note that the server contains the encrypted data as base64 strings. This was done for 2 major reasons - JSON objects(which are sent through the API) can handle strings easily, and universally bytestreams can be easily converted to base64 strings. Further, the js libraries used for cryptography work well with base64 encoding. Another reason for this design choice was that base64 encoded media can be conveniently rendered in browsers.

We have only stored the md5sum of the unencrypted file. This is to ensure upon downloading that the encryption schema of the downloading client is same as that of the uploading client.

2.1.3 Tokenization of APIs

In the tokenization system, whenever a user first sign in using a linux client, it is assigned a unique (username, token) field generated by the server itself that is stored in the database in Token model as well as locally in a log file. Now for all further API calls the token as well as the username is sent as the header which at server side after authentication with database Token model allows only those API calls to be successful which returns data related to that user.

2.1.4 Conflict Resolution

For conflict resolution, the contents of the server's copy of the file are downloaded, and the difference between the copies is shown to the user(using python `difflib`), who then decides which file to keep.

2.1.5 Race Conditions and Deadlock handling

Before each sync operation a signal is sent to the server confirming that another client of the same user is not syncing at that time, and another signal to lock the database. This lock is released by another signal at the end of each operation. A python script on the server end checks all the locked clients, and unlocks any clients locked for more than 15 minutes

2.1.6 Encryption

All the data on the server is encrypted with a personalized encryption schema and key that user enters/chooses.

We have provided a choice of 3 encryption schemes - AES, DES-3 and RC4 which the user can choose from. The user can either enter their own key, or a key can be randomly generated for them. The key is stored locally as a `pickle` file.

Upon changing the encryption scheme, all files of the user are downloaded from the server, re-encrypted and backed up again. Further, checks are put in place to see the last modified time of the encryption scheme (stored on the server) and the m-time of the local key to prevent bad decryptions.

2.1.7 File sharing

Users can share encrypted files among each other again using personalised encryption schema and key. The sharer's file is downloaded and decrypted locally, then a temporary key is generated for it. Then it is encrypted and uploaded to a buffer table along with its md5sum, path and receiver. This is again done through API calls. The receiver gets a notification about the impending download, and must then get the key from the sender. The file is downloaded and decrypted according to this temp key, and finally backed up(uploaded) to the receiver's cloud with his encryption scheme. The receiver then sends a signal to delete the file from the auxiliary table.

2.1.8 Periodic syncing

The observed directory will be periodically synced with the server. This is achieved with the use of cronjobs. A cronjob is set up which calls runs a python script to fetch the paths of the files on the server. If there are conflicted files, a notification is sent to the user through notify-send. Else,

sync is attempted. To display notifs, the output of the python script is piped to `stderr` and subsequently to the cronjob sending a notification.

2.2 WebClient

2.2.1 Login/Signup

The Web Client uses the inbuilt `Django.contrib.auth` libraries that generates a form in order to facilitate the Login of a user present in the model as well as Signup of a new user. During signup, Django also prevents the use of weak passwords and invalid usernames. Also clicking on back button after the user has logged in ensures that the session of the current user is not lost and same applies when the user has logged out. All of this has been implemented by Cache Control Library of Django. Web-Client also allows changing of the passwords which again is implemented by inbuilt Libraries of Django.

2.2.2 Encryption

The Web Client asks for the encryption schema and key from the user which is stored in Browser Cache and doesn't go in the network to ensure security of the data. The Browser then displays the file structure present on the server corresponding to the user in a neat folder tree view which has been implemented by BootStrap. Clicking on a particular file opens up another webpage that gets encrypted information from the server and decrypts it using CryptoJS Library. Just like Linux Client, the webclient supports three forms of encryption namely RSA, DES-3 and RC4.

2.2.3 Rendering

The encrypted files have been stored on the server in Base64 format and same is the format for the file that has been decrypted from CryptoJS. Using Base64 format, it becomes quite easy to render audio, video, pdf and image files in HTML5. The Web Client is able to render most of the generally used extensions such as mp4, mp3, jpg, png, jpeg etc. The files can also be downloaded from the Web Client. In Chrome, there is a download button present on the screen. In case of FireFox however, the user has to right click on the rendered file and click on Download. PDF can be downloaded on both the browsers directly.

3 Models used

User Model	The default django model which stores username, sha encrypted password, email id.
File Model	Stores owner user (foreign key), path, timestamp of modification, md5sum of file and the base64 encoded file.' Has a unique together constraint on user and path, i.e. one user cannot have multiple files of same name
Shared File Table	Stores sender, receiver, path of file, data and md5sum of the shared file.
User Attributes Table	Contains other information about the user, such as their state of database (locked/unlocked), time of last sync (for resolving deadlocks), whether their data is encrypted yet or not (for detecting first login of user and generating a scheme)
Token Table	Stores the authentication tokens of each user, which are unique for each user across clients. The tokens are used to secure the APIs.

References

[1] Django Tutorials on Youtube: Click Here

[2] Django Login Form : Click Here

[3] PDF rendering in HTML : Click Here

[4] CryptoJS encryption: Click Here

[5] BootStrap Tree View: Click Here

[6] Automatic Javascript Ul Li generator: Click Here

[7] Huge Thanks to people at Stack OverFlow

[8] Our Reliable Search Engine Google