Secure Personal Cloud

Project Report
int_elegance

Chitrank Gupta-170050024
Anurag Kedia-170050030
Jatin Lamba-170050039

Problem Statement

The aim was to create a secure cloud file storage system that would be immune to security issues. It should have multiple client support to enable sync across various devices.
Rest of the report briefs about the technical implementation details, the design choices and the justifications for various features and the work flow related to solve this problem.

Backend

Data Base Management

Djangon version 2.0.5 has inbuilt support for sqlite3. A table contianing user information was maintained using Django-auth library, while another was used to store information about the files uploaded. It contained the hash value, actual file content in binary form, encryption schema used and file path. Encrypted files were stored using Binary Field of Django. A "one to many" relation was maintained between user and file tables.

Rest API

Django Rest Framework was used to implement Rest APIs for our project. This is a secure method of file transfer in the form of JSON objects and at the same time provides a high level veiw of the project for the user. Django serializers were used to process data and handel request

Client Management

Features were added to enable new-user sign-up and login via the webclient. Django Sessions Security was used for session management adding to the security. Linux client interacts with the rest API for exchange of encrypted files. Content veiwing and rendereing allowed only if user logged in with valid credentials.

System Architecture

Figure 1: The Server-client Model

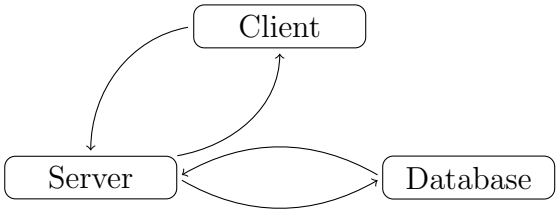
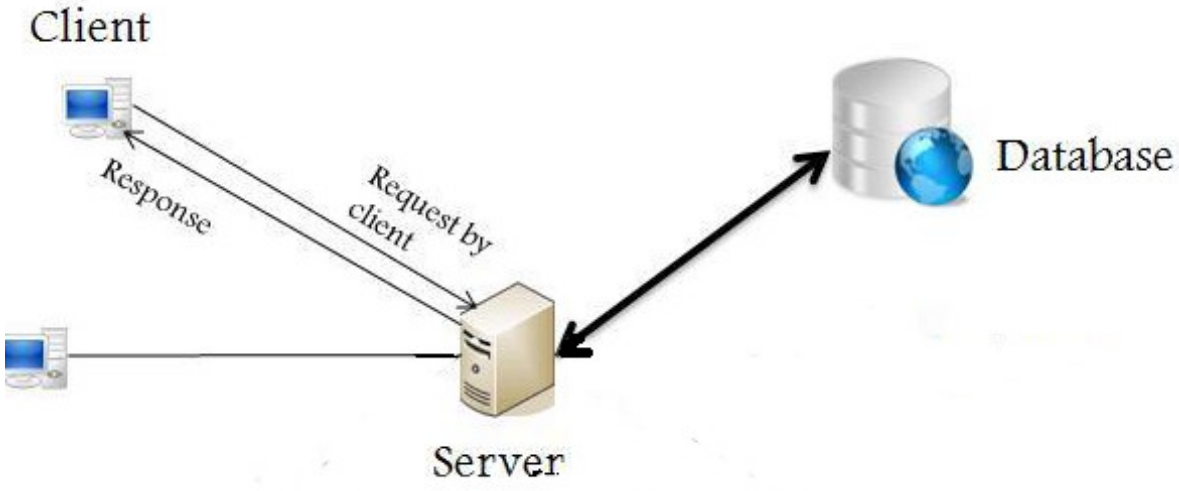


Figure 2: Schematic of our project (*courtesy:www.ianswer4u.com*)



The Clients

Linux Client

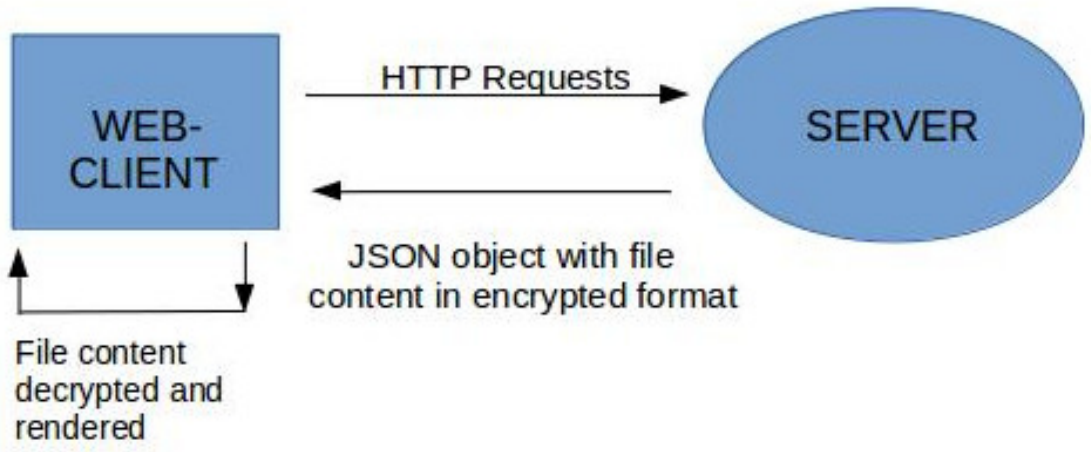
Linux Client has the functionality of syncing a local directory with the remote server after encryption using a user specified schema. Python request library was used to send POST, PUT and GET requests to interact with the backend. The user can change his/her schema at any time according to will giving the user complete control. Man page was created to list out and explain commands. An installation file was created to install recquired libraries and scripts for the user.

Web Client

Web client was implemented to enable a user preview his/her files there on the server. HTML, javascript and jQuery were used extensively to develop the same. Requests

via XMLHttprequests to APIs were used to load data on the client which was then decrypted and then rendered on the browser itself. HTML DOM was used to preview common file types like text files(.txt), image files (.png) and video files(.mp4).

Figure 3: How Web-Client Works?



The Sync

Sync on the linux client is what enables use to remotely keep track of his files. Implemented very similar to version control software. Functionalitis of 'staging area' 'status-check' have been added similar to any VCS. Linux client maintains a local database of all file-paths, hash values of their encrypted versions along with the time stamps of the files. Only the files added to the local database and are different from the remote database are encrypted and synced, the local database dones not contain the file contents to save space. Directory sturture is maintained as is between two clients with same credentials after sync.

Encryption Decryption

Securing data getting transferred on a network is a big challenge. A bigger challenge is to make encryption done using some language compatible with the decryption done in some other language. To resolve this issue we encrypted the files in Bash using openSSL command. We provided the user with three algorithms to encrypt his/her files- AES-128-CBC, DES-CBC with one key and 3DES-CBC with three keys, all in cipher block chaining mode with no salt (and therefore disabling padding), just for simplicity. The openSSL command provided the "key" and "iv" in Hexadecimal against some password

which user needs to remember so as to be able to decrypt files there on the web client. The crypto-js CDN is used by the webclient to decrypt the encrypted content loaded from JSON APIs. The reason we used these 3 encryption schemas were because they were those symmetric-key algorithms common to both openSSL and crypto-js.

Race Condtions

The project needs to support for a multi-client system. This leads to an interesting problem when the same user tries to sync at the same time from two different times. This was handled by locking the sync operation on the server at the first instance of a sync operation. The locking is the user level, meaning other users can still sync at the same time.
Locking the sync operation necessitates handling the deadlock state wherin the user kills the sync process unceremoniuslty, thereby preventing the sync operation to be unlocked. This was handles by placing a time limit of inactivity on that user.

Special Feature

As an additional feature we impleented the abity to prompt the user to sync periodically using linux daemons. Crontab was used to schedule jobs after every user specified number of minutes.

References

[1] *Django Tutorials*, https://docs.djangoproject.com/en/2.1/ .

[2] *Encryption via openSSL*, https://www.openssl.org/docs/man1.0.2/apps/enc.html

[3] *Decrypting file using crypto-js*, https://stackoverflow.com/questions/27375908/crypto-js-read-and-decrypt-file

[4] *Django rest API documentation* https://www.django-rest-framework.org/

[5] *Django tutorials bythenewboston on youtube* https://www.youtube.com/user/thenewboston