Ethereum - Secure Personal Cloud

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 Motivation

SPC seemed a great project to us because we could see it was closely related to the various file-storage systems being made available by the tech giants like Google and Microsoft. Making a similar project by ourselves seemed a great source of enthusiasm. Also, since it employs various fields of software development like system-programming, application writing, web-development, etc. it seemed to hold a huge potential for teaching us new things!

2 Introduction

We have made a server which gives the facility of uploading files to it and downloading from it. The server as stated in the problem statement holds only the encrypted files and never has access to the encryption key or schema.

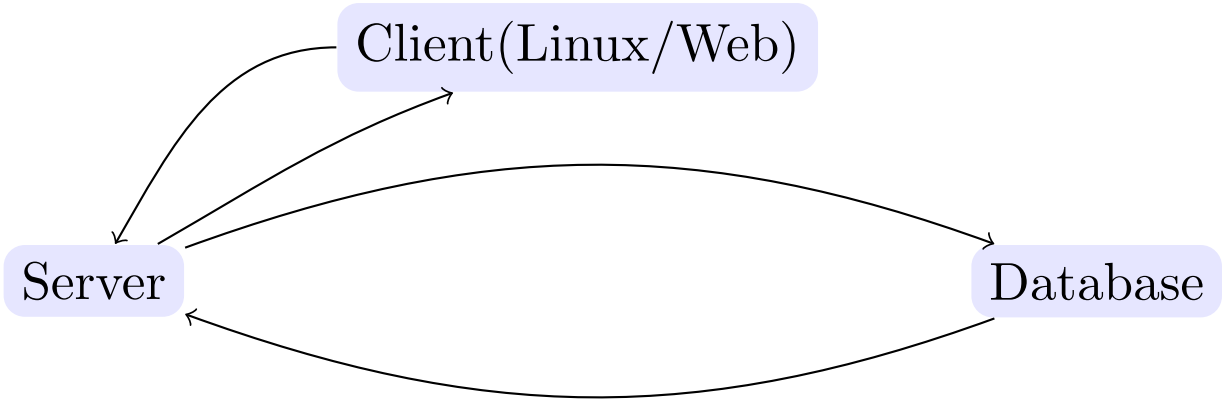
On the client side we have made a spc command with backend in python and a bash wrapper around it which enables users to sign in, store credentials, sync a particular directory with the cloud. We have given the user multiple choices for syncing protocols.

We also have a mobile-friendly web-client, based on Django, which lets the user see files of multiple types of data, such as text files, jpeg files, pdf files, etc.

A user guide and an install tool have also been provided.

We have used Django-REST framework [1] for making the server API and CoreAPI [2] for sending requests to it from the client side. Apart from this various libraries were used for things like AES Encryption [3], MD5sum calculation, System Programming(the very famous os library :)), etc.[4]

3 System Architecture



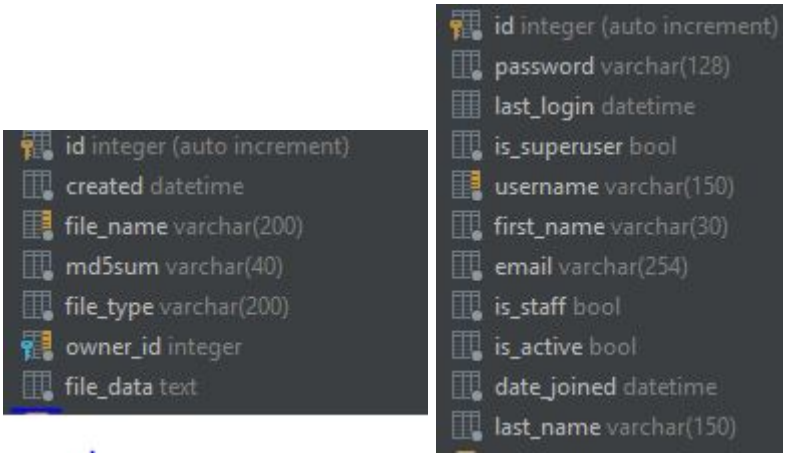
4 Features

- 3 Encryption schemes to choose from when encrypting data.
- Server has no access to encryption key or encryption scheme so cannot decrypt data.
- Use of Linux daemons for periodic syncing prompt.
- User can view/download data on web client.
- Race conditions and deadlock prevention for multiple clients.

5 Design Choices and Justifications

We have used Django server instead of other choices like PHP because of the high level of modularity and easy understandability of the code written. Also PHP is more of a bad choice as seen because the code is visible to others and it can be exploited easily to find loopholes in program. We have used Django REST for making our server, as we wanted to get the benifit of RESTful API's over the SOAP. Also it gives us the benifit of using the CoreAPI Library to handle request response in a much easier way.

Here is our database schema for the file and user table :-



For preventing deadlocking and race conditions we have maintained a json file on the server which contains user as keys and time stamp as value. Now when a user tries to sync we check if the user already exists in the table or if the existing user has a timestamp of greater than a fixed value. Then we update the timestamp and let the new user sync and at the end remove the user from dictionary. We also update the time-stamp after uploading each file so that even if no of files uploaded is large another client is not able to intrude.

For encryption we have PyCryptoDome which has a bijecting library in Javascript and which makes decrypting on the web-client a lot easier(though still tough). We have used block-cipher like AES as well as stream cipher like ARC4 each with its own advantages thus offering variety to the user. Apart from this we have used Blowfish encryption as our third encryption scheme.

Last but not the least we are providing users with the options of how they want to sync i.e merge server and client with overwrites on server, mirror your data exactly to the cloud, or merge server and client with overwrites on client.

6 Individual contribution and weightages

All the team members have an equal contribution of 33.3% each to the project.

References

[1] Django REST Documentation, see https://www.django-rest-framework.org/

[2] For CoreAPI go here

[3] AES can be found at https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html

[4] Inspired by Stack Overflow

[5] ARC-4 on JS https://gist.github.com/Grimi94/c80c8a29838c15f69a6a

[6] SHA256 in JS https://github.com/emn178/js-sha256

[7] www.w3schools.com