SOFTWARE SYSTEMS LAB

CS-251 REPORT

Secure Personal Cloud

Submitted To:
Prof Soumen Chakraborty
Asst. Professor
Computer Science
Department

Submitted By :
Vijaykrishna G, 170050090
Tushar Gautam, 170050026
Pranay Reddy, 170070022
Team Name:
COD(e)OMAIN

Contents

1 Introduction

We have made a simple working **secure personal cloud** combining tools from different software systems including **Django, JavaScript, Python and Bash**. Features include **Linux Client, Web Client, Periodic Sync using Linux Daemons**. Three different encrytion schemes: **1) RSA, 2) AES, 3) DESS** have been implemented.

2 Back-end design using Django and sqlite3

2.1 Database Design

The basic structures used are the default **user class, file class, folder class**. The data is stored in **.sql tables**. The members in different classes are as shown below.

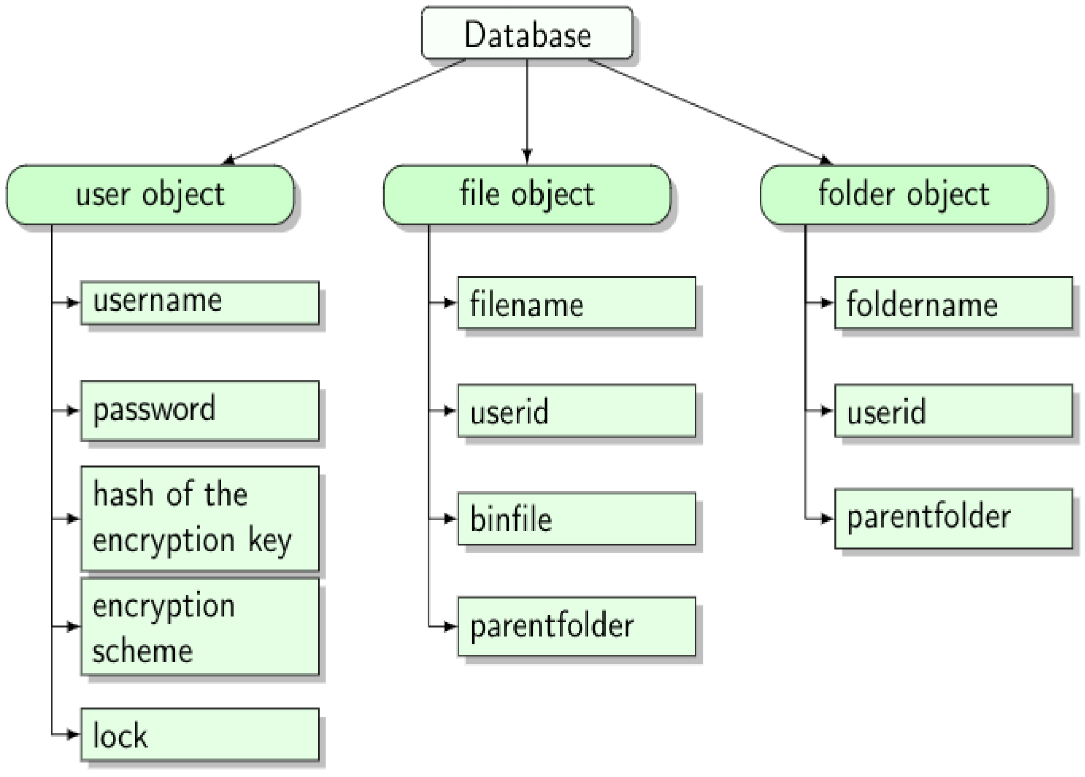


Figure 1: Database Structure

The class definition for user defined classes are present in **models.py**

2.2 Servers and Clients

The server model has been implemented on **Python using Django**. On the client side, there are two options: a) Web client and b) Linux client. The functions and **HTML rendering** relevant to a) have been implemented in **views.py**. Login and retrieval in **Linux client** have been done using requests library using **GET and POST**. The necessary urls are present as regular expressions in **urls.py** redirecting to appropriate functions in **views.py**.

2.3 Users

Files of a user x, can be retrieved by queries of the form **file in FILES where userid=x**. Registering, uploading and downloading adds new objects into the database of **FILES, USERS, FOLDERS**. While uploading, the **required file** is first converted to **String** and sent to the server using **POST** method, where it is converted back to **binary** format and stored in the **binfile** data member of a new file object.

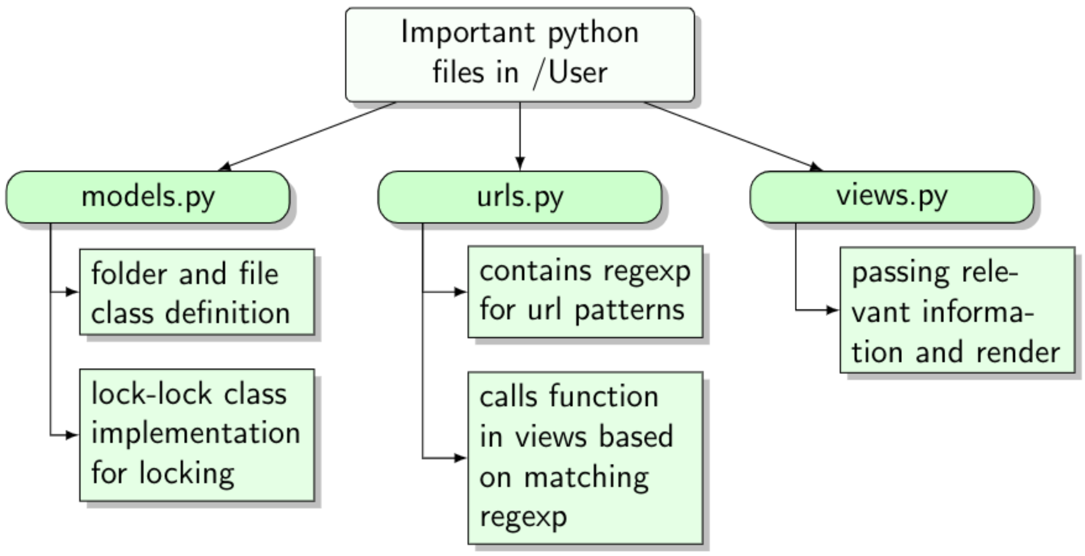

Important python files in /User
models.py
folder and file class definition
lock-lock class implementation for locking
urls.py
contains regexp for url patterns
calls function in views based on matching regexp
views.py
passing relevant information and render

Figure 2: Description of important python files in /User

3 Linux Client

3.1 Installation without Sync

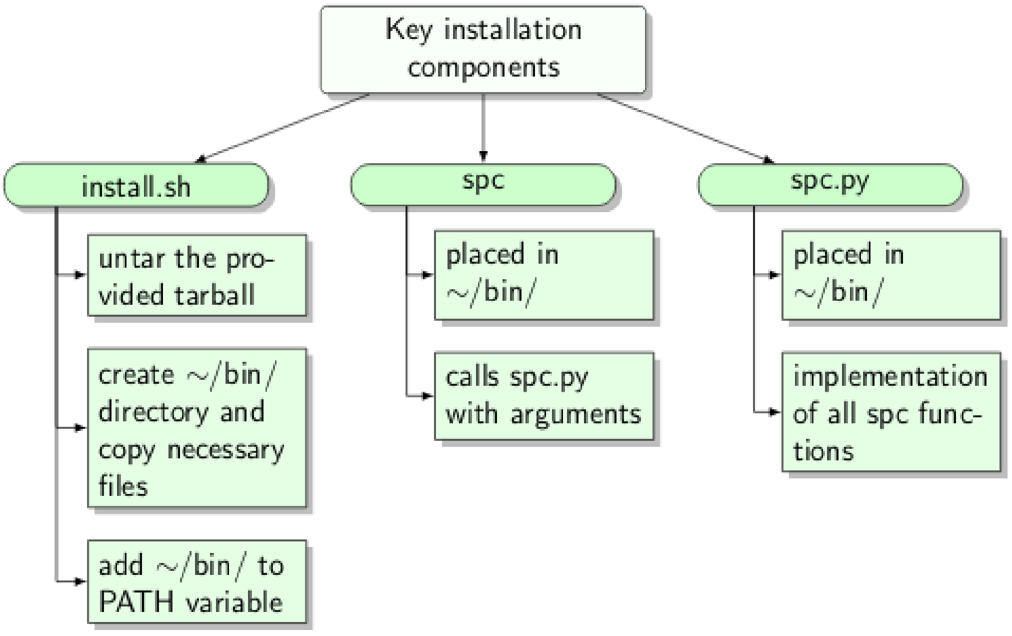

Key installation components
install.sh
untar the provided tarball
create ∼/bin/ directory and copy necessary files
add ∼/bin/ to PATH variable
spc
placed in ∼/bin/
calls spc.py with arguments
spc.py
placed in ∼/bin/
implementation of all spc functions

Figure 3: Linux Client Installation

3.2 Interface

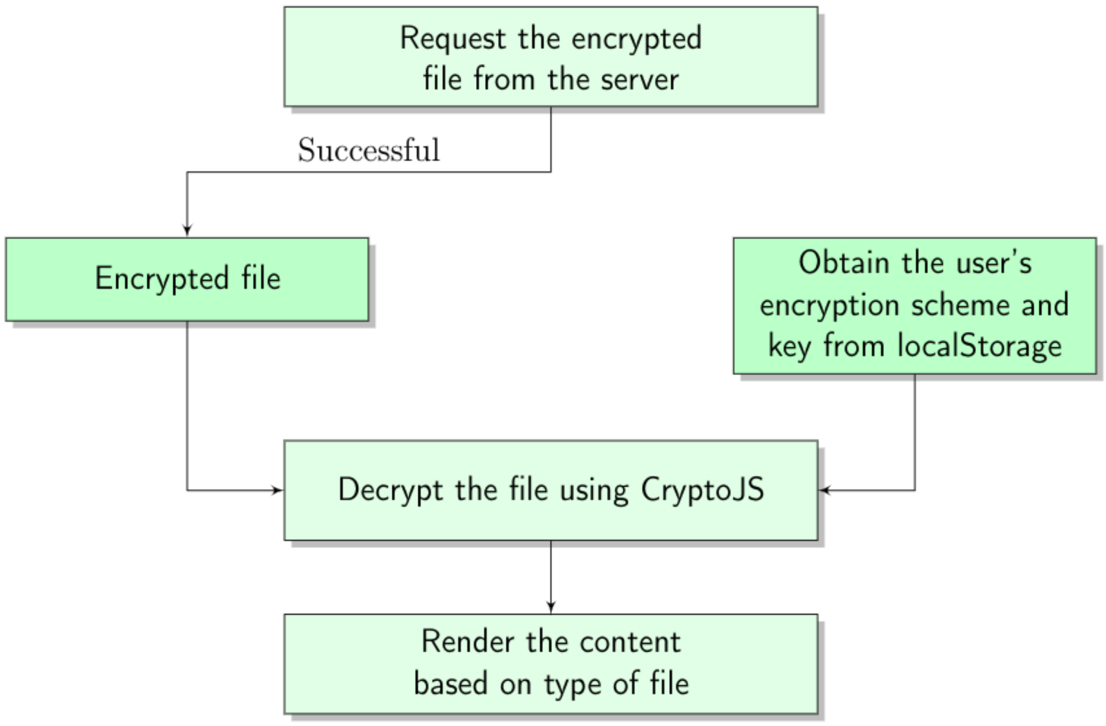
The interface is **terminal-based**. Several commands have been included which can be used as **spc *command*** on terminal. Since **∼/bin/** has been added to **PATH** variable during installation, we can use **spc** command from any directory.

3.3 Periodic Sync (Additional Feature)

Periodic sync has been implemented using **Crontab** by adding a cronjob which prompts the user at regular intervals about sync. To set this up, the script **addToCron.sh** in the installation package writes this cronjob during installation itself. The script which is executed by the cronjob is **∼/bin/sync.sh** which calls **spc sync** on a new **Konsole** window.

4 Web Client

The **HTMLs** with relevant **Javascipt** code is present in **User/templates/user/** directory. The **login, register, upload, download** interface are standard **HTML** implementations. For storing the **decryption scheme** and **key** on the user's machine, we make use of **localStorage** feature of the web browser. Once the user sets these two fields, all subsequent download operations use these stored fields directly. In **localStorage** each **username** is mapped to a **JSON dict object:** {`encryption scheme, encryption key`} The web browser receives **encrypted file** from server and **decryption** is done on the **user's browser** using **CryptoJS**.



Request the encrypted file from the server
Successful
Encrypted file
Obtain the user's encryption scheme and key from localStorage
Decrypt the file using CryptoJS
Render the content based on type of file

Figure 4: Download and Decrypt on Web Browser

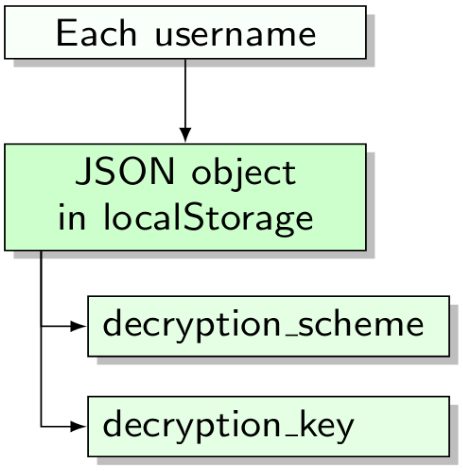


Figure 5: JSON object in LocalStorage

5 More details about Sync, Lock and Encryption Schemes

5.1 Sync

Sync is performed by checking **md5sum** between files in **local machine** and on the **server** and syncing accordingly. The user has two options, either to give **higher priority to files** on the server or on his local machine.

5.2 Lock using boolean field

If a user is uploading a file then the **lock**, which is a **boolean field** is set to **1**, which means no other user can upload at the same time. Once the **upload is over**, the boolean field is set back to **0**. Prevention of **dead lock** is achieved by running a **sleep 60** process **in parallel** after which the boolean field is set back to **0**.

5.3 Encryption schemes

As mentioned before, **3** encryption schemes have been used. The user is provided the option of **selecting** and **changing** the encryption scheme of his files on the server. **Crypto** library on **Python** and **CryptoJS** on **JavaScript** have been used.

6 References

1. https://stackoverflow.com/questions/49544982/encrypt-decrypt-in-java-and-python-using-aes
2. https://stackoverflow.com/questions/10440777/how-do-i-encrypt-in-python-and-decrypt-in-java
3. https://stackoverflow.com/questions/10440777/how-do-i-encrypt-in-python-and-decrypt-in-java/1044088610440886
4. https://www.w3schools.com
5. https://github.com/buckyroberts/Viberr for Django