

Securing the Cloud

Student name: Yingzheng Pan

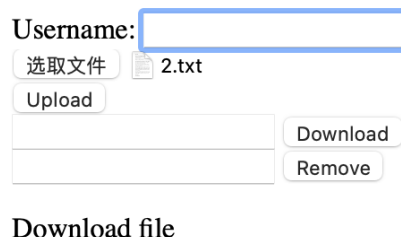
Student number: 19336862

Purpose:

The aim of this project is to develop a secure cloud storage application for Dropbox, Box, Google Drive, Office365 etc. For example, your application will secure all files that are uploaded to the cloud, such that only people that are part of your “Secure Cloud Storage Group” will be able to decrypt uploaded files. Any member of the group should be able to upload encrypted files to the cloud service. To all other users the files will be encrypted. A suitable key management system will be designed and implemented for the application that will allow files to be shared securely, and users to be added and removed from “Secure Cloud Storage Group”. The application can be set up on a desktop or mobile platform and make use of any open-source cryptographic libraries.

Components and Implementation:

There are three main components python files are client.py, server.py and encrypt_file.py. I used index.html and Flask in python to build a web application. As the figure 1 shown, that's my application user interface.



The screenshot displays a web application interface. At the top, there is a label 'Username:' followed by a text input field. Below this, there is a button labeled '选取文件' (Select File) next to a file icon and the text '2.txt'. Underneath, there is an 'Upload' button. To the right of the 'Upload' button, there are two buttons: 'Download' and 'Remove'. Below these buttons, there is a text label 'Download file'.

Figure 1

For the user interface, user can access the <http://localhost:8080> and then put the username in the first line, I have a group list for who can access the cloud storage called secure-cloud-group.json, my project will check the username you enter whether it is in the list or not, if you are in the group list then you can access the cloud storage and download the file you want. Also you can upload the file to the cloud storage You can select any file that is local to your computer and upload it to my website. Then click upload, and the file you

select will be stored in the cloud. You can also delete any file in the cloud. These operations also involve encrypting keys and decrypting keys. Now let me explain the code and process in detail. Firstly, the component is client.py. As the figure 2 shown below, that's my client part code, which contains suitable key management system and allow files to be shared securely, and users to be added and removed from "Secure Cloud Storage Group".

```
downloads = os.path.join(app.root_path, 'downloads')
resources = os.path.join(app.root_path, 'keys')

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/downloads/", methods=["POST"])
def download():
    server_public_key = serialization.load_pem_public_key(
        backend=default_backend()
    )

    filename = request.form["filename"]
    username = request.headers["username"]
    username = base64.b64encode(rsa_encrypt(str.encode(username), server_public_key))
    filename = base64.b64encode(rsa_encrypt(str.encode(filename), server_public_key))

    response = requests.post('http://127.0.0.1:5000/download/',
                             data={"publickey":base64.b64encode(open("./keys/rsa_public_key.pem", "rb").read()), "f":filename},
                             auth=(username, ""))

    if response.status_code == 404:
        return Response(404)
    response_file_data = base64.b64decode(response.content)
    iv = response_file_data[:16]
    enc_file = response_file_data[16:]
    aes_decoded = rsa_decrypt(base64.b64decode(response.headers["aes"]), get_private_key())
    decoded_unpadding = aes_decrypt(aes_decoded, iv, enc_file)

    file_directory_name = os.path.join(downloads, request.form["filename"])
    with open(file_directory_name, "wb") as f:
        f.write(decoded_unpadding)
        f.close()
    return Response("File downloaded successfully.", 200)

@app.route("/upload/", methods=["POST"])
def upload():
    username = request.headers["username"]
    response = requests.post('http://127.0.0.1:5000/publickey')
    server_public_key = base64.b64decode(response.content)
    server_public_key = serialization.load_pem_public_key(
        server_public_key,
        backend=default_backend()
    )

    if "file" in request.files:
        file = request.files['file']
        if file.filename != "":
            filename = secure_filename(file.filename)
            unencrypted_file = file.read()
            aes_key = open("./keys/aes_key.pem", "rb").read()

            iv = os.urandom(16)
            ct = aes_encrypt(iv, aes_key, unencrypted_file)
            encrypted_aes_key = rsa_encrypt(aes_key, server_public_key)
            encrypted_iv = rsa_encrypt(iv, server_public_key)

            file = {"file":(filename, ct)}
            username = base64.b64encode(rsa_encrypt(str.encode(username), server_public_key))
            headers = {}
            response = requests.post('http://127.0.0.1:5000/upload/', headers=headers, files=file, data={"aes":base64.b64encode(encrypted_aes_key)})
            if response == 403:
                return Response("Disallow", 403)
            return Response("Uploaded Successfully", 200)
```

```

@app.route("/remove/", methods=["POST"])
def remove_files():
    response = requests.post('http://127.0.0.1:5000/publickey')
    server_public_key = base64.b64decode(response.content)
    server_public_key = serialization.load_pem_public_key(
        server_public_key,
        backend=default_backend()
    )

    filename = request.form["filename"]
    username = request.headers["username"]
    username = base64.b64encode(rsa_encrypt(str.encode(username), server_public_key))
    filename = base64.b64encode(rsa_encrypt(str.encode(filename), server_public_key))
    response = requests.post('http://127.0.0.1:5000/remove', data={"filename":filename}, auth=(username,""))
    if response.status_code == 200:
        return Response("Successfully removed", 200)

@app.route("/downloads/<path:filename>/")
def download_files_to_computer(filename):
    return send_from_directory(downloads, filename)

```

Figure 2

Firstly we load the index.html, which is my webpage of user interface, and then there are three processes in my client part, which are upload, download and remove files. For the asymmetrical key consist of public key and private key, The public key will sent alone with user name to the cloud storage sever part to request for the symmetric key used for encryption and decryption of files in the group. As we can see in the code, the symmetric key used for encryption when uploading files and decryption when downloading files. Client will send request to the server part, and then when server part check whether the user is in the group list or not. If not in the group member, no symmetric key will be sent in response. When user click upload files, the file will store in the cloud storage, which is in my storage folder. When user click download, it will directly download to the user's computer and when user click remove, it will remove the file in cloud storage you select.

Secondly, the sever.py. As the figure 3 shown below.

```

storage = os.path.join(app.root_path, 'storage')
resources = os.path.join(app.root_path, 'resources')

MGMT_NAME = "CRAIG_WRIGHT"

@app.route("/password", methods=['POST'])
def get_cred():
    f = json.load(open("./resources/ip_access.json"))
    if request.remote_addr not in f:
        f[request.remote_addr] = 0
        with open("./resources/ip_access.json", "w") as ip_file:
            json.dump(f, ip_file)
    else:
        f[request.remote_addr] = f[request.remote_addr] + 1
        with open("./resources/ip_access.json", "w") as ip_file:
            json.dump(f, ip_file)
        if f[request.remote_addr] >= 100:
            return Response("Unauthorized. - Accesses exhausted for address.", 403)
    username = base64.b64decode(request.authorization["username"])
    username = rsa_decrypt(username, get_private_key()).decode()
    if username not in json.load(open("./resources/secure_cloud_storage_group.json", "rb"))["Users"]:
        return Response("Unauthorized. - User does not belong to remote access group.", 403)

```

```

@app.route('/upload/', methods=['POST'])
def upload_file():
    # if user is in users.json os.join send 200
    # else send 403 forbidden

    with open('resources/secure_cloud_storage_group.json') as f:
        users = json.load(f)

    username = base64.b64decode(request.authorization["username"])
    if "file" in request.files:
        file = request.files['file']
        if file.filename != "":
            # decrypt aes key
            encrypted_aes_key = base64.b64decode(request.form["aes"])
            encrypted_iv = base64.b64decode(request.form["iv"])

            private_key = get_private_key()
            aes_key = rsa_decrypt(encrypted_aes_key, private_key)
            iv = rsa_decrypt(encrypted_iv, private_key)

            filename = secure_filename(file.filename)

            # decrypt received file using user's AES key
            ct = file.read()
            decoded_unpadded_file = aes_decrypt(aes_key, iv, ct)

            iv = os.urandom(16)

```

```

@app.route("/downloads/", methods=['POST'])
def download_file():
    with open('resources/secure_cloud_storage_group.json') as f:
        users = json.load(f)
    username = base64.b64decode(request.authorization["username"])
    username = rsa_decrypt(username, get_private_key()).decode()

    if username not in users["Users"]:
        return Response("Disallowed Action. ", 403)

    if "publickey" in request.form:
        user_public_key = base64.b64decode(request.form["publickey"])

        if "filename" in request.form:
            filename = base64.b64decode(request.form["filename"])
            filename = rsa_decrypt(filename, get_private_key()).decode()
            files = json.load(open("./resources/files.json", "r"))
            if filename not in files:
                return ("File not found. ", 404)

            public_key = serialization.load_pem_public_key(
                user_public_key,
                backend=default_backend()
            )

            encrypted_aes = rsa_encrypt(open("./keys/aes_key.pem", "rb").read(), public_key)

            response = send_from_directory(storage, filename)

```

```

@app.route("/files/", methods=["POST"])
def serve_file_list():
    with open('resources/secure_cloud_storage_group.json') as f:
        users = json.load(f)

    if request.headers["username"] not in users["Users"]:
        return Response("Disallowed Action. ", 403)
    elif request.headers["username"] in users["Users"]:
        with open('resources/files.json') as f:
            files = json.load(f)
            print("Files returned.", files)
            return json.dumps(files)

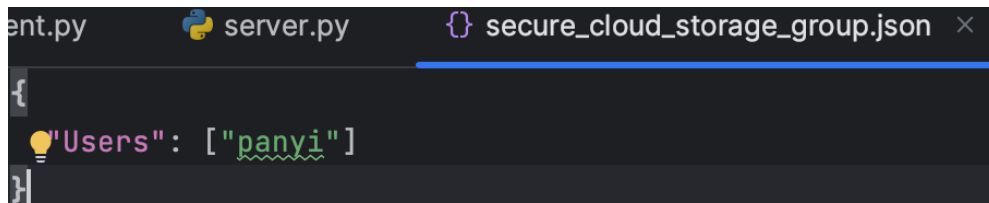
@app.route('/remove/', methods=['POST'])
def remove_file():
    with open('resources/secure_cloud_storage_group.json') as f:
        users = json.load(f)

    username = base64.b64decode(request.authorization["username"])
    username = rsa_decrypt(username, get_private_key()).decode()

    if username not in users["Users"]:
        print("Not found")
        return Response("Disallowed Action", 403)

    with open('resources/files.json') as f:
        files = json.load(f)
        if "filename" in request.form:

```



```

{
  "Users": ["panyi"]
}

```

Figure 3

At the beginning, my code will check which user can access the cloud storage in the .json file. If the user is in the json file and then it means that user can access the cloud storage and then do the processes, for example upload, download and remove as the code shown. In my code, if it successful and then my code will send number 200, if it is not, my code will use 403 to do forbidden.

Thirdly, the encrypt_file.py. The encrypt part is set up to process all the encryption and decryption of the files passed.

In Conclusion, From this project I have learned a lot. I learned how to encrypt and decrypt files and manage the users database, which expanded my knowledge and made me learn a lot about network.