# CS3041 Assignment 2

## Yasir Mohamed – 04/04/2016

# Introduction

My submission for this assignment is written in a mixture of Python 3, PHP5 and MySQL, where Python is used for the user interface of the application, and PHP and MySQL are hosted on my own private apache server. Files that are encrypted using my application are done using AES256, and the PyCrypto and SimpleCrypt Libraries. My approach, was that rather than managing keys on multiple machines, it’s much easier to have the server be the only entity in control of the encryption key. This way, only users who have provided authorised login details can have the key temporarily downloaded to the machine and then used to decrypt files. With this approach, the encryption key is never saved on a local machine.

Since the project is hosted on my own server, I had to configure both the client application and the server to complete the project; and as such I will be splitting this document into details of the Client-Side application and the Server-Side application.

The key used in this key management system was created using a random key generator I wrote, and is stored in the server login.php file. This file is only accessible by the server owner, and as such the key can only be updated by the owner.

# Server Side Application

The server consisted of a MySQL database to store user login details, and a single PHP script to handle all client interactions with the server.

## usersTable.sql

This file simply defined the structure of the database used to store user details. The structure was simply User name (Primary), Salt, Hash, Admin. User name was the primary key of the table to help searching and prevent duplicate names. I implemented a salted hashing function in the Server’s PHP file for saving passwords, and I implemented different user levels (Administrator and non-Administrator), as some functions in the application require elevated privilege (Adding/Removing users).

## Login.php

A misnomer, this file handles all interactions between the python desktop client and the server. This file parses requests from the desktop application, and when valid, completes the requested action. If the request is invalid, then an error is returned to the client which displays an appropriate error message. The breakdown of valid requests and functions in the login.php is as follows:

### Requests

|  |  |
| --- | --- |
| **Request Name** | **Description** |
| LOGIN | This is the first request made by every instance of the desktop application, it takes user login details and checks them against details stored in the database. If successful, it returns the users privilege level. This request uses the methods verifyLogin, and isAdminLevel. |
| CREATE | This request type is used to create and store new authorised users to the database. It first verifies the login details of the user making the request, then checks the user is an administrator. If both checks are successful, the new users password is hashed, then the details stored in the database. Methods used are hashPass and addUser. |
| REMOVE | This request type removes users from the group. The user must be verified and must be an Admin, and in the case that the user is then the specified user is removed from the database if they exist. The method used here is removeUser |
| getKey | This request is used to retrieve the encryption key needed to encrypt and decrypt files from the server. It checks to ensure the user is a valid user, then returns the key back to the application. It uses the method returnKeys. |
| listDirectory | This request lists the directory used by the shared file group, it is called by the desktop application to display files available to the user. It uses the method returnDirectory. |
| uploadFile | This request allows a valid user to upload files to the server to be accessed by other members of the group. These files do not necessarily have to be encrypted before upload. It uses the method uploadFile. |

### Methods

|  |  |
| --- | --- |
| checkInput | Checks user input and formats it to ensure there is no malicious input |
| connectToDB | Connect to the user database |
| hashPass | Takes in a password and hashes it using MD5. Password consists of the original password and hash |
| generateSalt | Uses built in PHP libraries to generate a 16 byte Salt for passwords |
| getUserSalt | Retrieve user salt from database |
| getUserPassword | Retrieve the hashed password from the database for a user |
| checkUserExists | Check to make sure a given user already exists in the database |
| isAdminLevel | Checks the database to see if a user is an administrator |
| addUser | Add a new user to the database |
| removeUser | Remove an existing user from the database |
| verifyLogin | Verify the login details of a user with a user from the database |
| returnKeys | Return the encryption key to the user on request |
| returnDirectory | Return all the files in the user upload directory |
| downloadFile | Streams a file as bytes to the user on request |
| uploadFile | Upload a file from the client to the group uploads directory |

# Client Side

This is the client interface for the application, built using Tkinter. I decided to allow users to be as flexible as possible when using the application, so as a result I decided not to limit files uploaded to the server to be encrypted. As a consequence, the upload, download, encrypt and decrypt buttons are all separate entities, to allow for offline transport of encrypted files. In an effort to make the application modular, I’ve separated the core functionality of the application into separate files.

### Login

This generates a login window for the client application, where the user must login. The input triggers a LOGIN request on the server in the login.php file. On successful login, the main console window is generated giving the user access to the main functionality of the application.

### MainConsole

This is the main interface for the application, which gives the users access to 6 main functions. These are as follows:

|  |  |
| --- | --- |
| Download | This is a drop down file menu that lists all the files currently stored in the server. To download a file, simply select it from the dropdown and click the download button. |
| Upload | This opens a file dialog window to select a file to be uploaded to the server. Files that are uploaded to the server do not need to be encrypted. |
| Encrypt | This opens a file dialog window to select a file to be encrypted. The result is then saved in the folder “EncryptedFolder”. For this function to work, the working directory must have the folder “EncryptedFolder”. |
| Decrypt | This opens a file dialog window to select a file to be decrypted. The result is then saved in the folder “DecryptedFolder”. For this function to work, the working directory must have the folder “DecryptedFolder”. |
| Remove User | Generates a window to remove user, using the CipherFunctions python module in the directory. This takes a username and asks the admin to verify their password, then makes a request to the server to remove a user provided they exist |
| Add User | Generates a window to add a new user, using the CipherFunctions python module in the directory. This takes a username and password, then makes a request to the server to add a user to the database. |

# Code

## Login.py

**from** tkinter **import** \*

**from** tkinter **import** messagebox

**import** requests

**import** config

**import** mainConsole

#Destroy all the frames in a grid

**def** destroyGrid(frame):

**for** widget **in** frame.grid\_slaves():

widget.destroy()

#Generate the login window presented upon starting the program

**def** generateLoginWindow(frame):

#process the login details inputted by the user

**global** USERNAME

**global** PASSWORD

**def** processLogin(uname, pword):

payload = {

"RequestType" : "LOGIN",

"name" : uname,

"password" : pword

}

res = requests.post("http://mohamey.me/login.php", data=payload)

**if** res.text == "y" **or** res.text == "n":

config.updateLogin(uname, pword)

**if** res.text == "y":

config.updateAdmin(**True**)

config.getAuthenticationKey()

messagebox.showinfo("Success!", "Login Successful")

destroyGrid(frame)

mainConsole.generateMainWindow(frame)

**else**:

messagebox.showerror('Login Failed', 'Please check your User name and password and try again')

frame.wm\_title("Super Secret File Encryption")

#Variables used for logging in

userName = passWord = ""

#Create and place label for user name entry

uNameLabel = Label(text="User Name")

uNameLabel.grid(row=0, column=0, sticky="W")

#Create and place entry for User Name

uName = Entry(frame)

uName.grid(row=0, column=1, padx=5, pady=5)

#Create and place label for password entry

pWordLabel = Label(text="Password")

pWordLabel.grid(row=1, column=0, sticky="W")

#Create and place entry for Password

pWord = Entry(show="\*")

pWord.grid(row=1, column=1, padx=5, pady=5)

#Create and place the login button

loginButton = Button(text="Login", command= **lambda**: processLogin(uName.get(), pWord.get()))

loginButton.grid(row=3, column=0, columnspan=2, pady=5)

**if** \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

generateLoginWindow(root)

root.mainloop()

## MainConsole.py

**import** webbrowser

**from** tkinter **import** \*

**from** tkinter **import** messagebox

**import** addNewUser **as** newUserDialog

**import** removeUser **as** removeUserDialog

**import** login

**import** cipherFunctions **as** cipher

**import** config

**import** requests

#Process the decrypt file inputs

**def** processDecryptButton(location, name, key, frame):

**if** ((location=="") **or** (name=="Browse...") **or** (key=="")):

messagebox.showwarning("Error", "Error parsing inputs while decrypting file")

**return**

#When inputs are okay, try decrypting the file

**if** cipher.decryptFile(location, name, key):

messagebox.showinfo("Success!", "The file was successfully decrypted. Please find it in the Decrypted Files folder")

login.destroyGrid(frame)

generateMainWindow(frame)

**else**:

messagebox.showerror("Unsuccessful", "There was a problem decrypting the specified file")

#Process the encrypt file inputs

**def** processEncryptButton(location, name, key, frame):

**if** ((location=="") **or** (name=="Browse...") **or** (key=="")):

messagebox.showerror("Error", "Error occured while parsing inputs to encrypt file")

**return**

#When inputs are okay, try decrypting file

**if** cipher.encryptFile(location, name, key):

messagebox.showinfo("Success!", "The File was successfully encrypted. Please find it in the Encrypted Files Folder")

login.destroyGrid(frame)

generateMainWindow(frame)

**else**:

messagebox.showerror("Unsuccessful", "There was a problem encrypting the specified file")

**def** processAddUserFrame(frame):

login.destroyGrid(frame)

newUserDialog.generateAddUserFrame(frame)

**def** processRemoveUserFrame(frame):

login.destroyGrid(frame)

removeUserDialog.generateRemoveUserFrame(frame)

**def** getFileNames():

name = config.getName()

password = config.getPassword()

payload = {

"name" : name,

"password" : password,

"RequestType" : "listDirectory"

}

res = requests.post("http://mohamey.me/login.php", data=payload)

files = ((res.text)[5:]).split(",")

**return** files

**def** downloadFile(filename):

**if** filename == "Select File":

messagebox.showerror("Error", "You must select a file from the dropdown")

**return**

payload = {"file":filename}

res = requests.get("http://mohamey.me/login.php", params=payload)

fileObject = open("Downloads/"+filename, 'wb')

fileObject.write(res.content)

fileObject.close()

messagebox.showinfo("Success", "Download Complete, please find the file in the downloads folder where you will need to decrypt it")

**def** uploadFile(fileLocation, frame):

fileUpload = {"file" : open(fileLocation, 'rb')}

payload = {

"RequestType" : "uploadFile",

"name" : config.getName(),

"password" : config.getPassword()

}

res = requests.post("http://mohamey.me/login.php", data = payload, files=fileUpload)

**if** res.text == "Successfully uploaded file":

messagebox.showinfo("Result", res.text)

destroyGrid(frame)

generateMainWindow(frame)

**else**:

messagebox.showerror("Error", res.text)

**def** generateMainWindow(frame):

frame.wm\_title("User Console")

**global** USERNAME

**global** PASSWORD

#Open file browser dialog

**def** browseFile(fileVar, fileLocationVar):

**from** tkinter **import** filedialog

fileLocation = filedialog.askopenfilename(filetypes=[("Any", "\*.\*")])

**if** len(fileLocation) > 0:

displayName = str(fileLocation).split('/')

fileVar.set(displayName[-1])

fileLocationVar.set(fileLocation)

**return**

#Download File row

downloadOption = StringVar()

downloadOption.set("Select File")

serverFiles = getFileNames()

downloadFileLabel = Label(text="Download File")

downloadFileLabel.grid(row=0, column=0, sticky="W")

dropDown = OptionMenu(frame, downloadOption, \*serverFiles)

dropDown.grid(row=0, column=1, padx=5, pady=5)

downloadButton = Button(text="Download File", command = **lambda**: downloadFile(downloadOption.get()))

downloadButton.grid(row=0, column=2, padx=5, pady=5)

#Upload File row

uploadOption = StringVar()

uploadOption.set("Browse...")

uploadFileLocation = StringVar()

uploadFileLocation.set("Browse...")

uploadFileLabel = Label(text="Upload File")

uploadFileLabel.grid(row=1, column=0, sticky="W")

uploadBrowseButton = Button(textvar=uploadOption, command=**lambda**: (browseFile(uploadOption, uploadFileLocation)))

uploadBrowseButton.grid(row=1, column=1, padx=5, pady=5)

uploadButton = Button(text="Upload File", command=**lambda**:(uploadFile(uploadFileLocation.get(), frame)))

uploadButton.grid(row=1, column=2, padx=5, pady=5)

#Select the file you would like to decrypt | File browser | Decrypt

decryptFileLocation = StringVar()

decryptFileLocation.set("Browse...")

decryptLabel = Label(text="Select the file you would like to decrypt")

decryptLabel.grid(row=2, column=0, sticky="W")

decryptFileVar = StringVar()

decryptFileVar.set("Browse...")

decryptBrowseButton = Button(textvar=decryptFileVar, command= **lambda**:(browseFile(decryptFileVar, decryptFileLocation)))

decryptBrowseButton.grid(row=2, column=1, padx=5, pady=5)

decryptButton = Button(text="Decrypt File", command= **lambda**:(processDecryptButton(decryptFileLocation.get(), decryptFileVar.get(), config.getKey(), frame)))

decryptButton.grid(row=2, column=2, padx=5, pady=5)

#Select the file you would like to Encrypt | File Browser | Encrypt

encryptFileLocation = StringVar()

encryptFileLocation.set("")

encryptLabel = Label(text="Select the File you would like to Encrypt")

encryptLabel.grid(row=3, column=0, sticky="W")

encryptFileVar = StringVar()

encryptFileVar.set("Browse...")

encryptBrowseButton = Button(textvar=encryptFileVar, command=**lambda**:(browseFile(encryptFileVar, encryptFileLocation)))

encryptBrowseButton.grid(row=3, column=1, padx=5, pady=5)

encryptButton = Button(text="Encrypt File", command= **lambda**: (processEncryptButton(encryptFileLocation.get(), encryptFileVar.get(), config.getKey(), frame)))

encryptButton.grid(row=3, column=2, padx=5, pady=5)

#Show buttons for user addition and removal

addUserButton = Button(text="Add New User", command= **lambda**: processAddUserFrame(frame))

addUserButton.grid(row=4, column=1, padx=5, pady=5)

removeUserButton = Button(text="Remove Existing User", command= **lambda**: (processRemoveUserFrame(frame)))

removeUserButton.grid(row=4, column=2, padx=5, pady=5)

### Config.py

This file is used to keep track of user details while the application is running on the desktop. All data stored in variables during running of the application is discarded upon completion.

**import** requests

#Store data to be used across Files

USERNAME = PASSWORD = KEY = ACCESS\_TOKEN = ""

ADMIN = **False**

**def** updateLogin(name, pword):

**global** USERNAME

**global** PASSWORD

USERNAME = name

PASSWORD = pword

**def** updateKey(newKey):

**global** KEY

KEY = newKey

**def** updateAdmin(newBool):

**global** ADMIN

ADMIN = newBool

**def** getName():

**global** USERNAME

**return** USERNAME

**def** getPassword():

**global** PASSWORD

**return** PASSWORD

**def** getKey():

**global** KEY

**return** KEY

**def** getAdmin():

**global** ADMIN

**return** ADMIN

**def** getAuthenticationKey():

**global** USERNAME

**global** PASSWORD

**global** KEY

payload = {

"name" : USERNAME,

"password" : PASSWORD,

"RequestType" : "getKey"

}

**try**:

res = requests.post("http://mohamey.me/login.php", data=payload)

KEY = res.text

**except** Exception **as** e:

print(e)

**return** **False**

**return** **True**

### cipherFunctions.py

**from** simplecrypt **import** encrypt, decrypt, DecryptionException

**from** tkinter **import** messagebox

#Encrypt file using simplecrypt

**def** encryptFile(location, name, key):

**with** open(location, "rb") **as** unencryptedFile:

byteForm = unencryptedFile.read()

encryptedBytes = encrypt(key, byteForm)

**try**:

destinationFile = "EncryptedFiles/"+name

encryptedFile = open(destinationFile, "wb")

encryptedFile.write(encryptedBytes)

encryptedFile.close()

**return** **True**

**except**:

print("An error occured writing the encrypted bytes to file")

**return** **False**

#Decrypt file using simplecrypt

**def** decryptFile(location, name, key):

**with** open(location, "rb") **as** encryptedFile:

**try**:

encryptedBytes = encryptedFile.read()

decryptedBytes = decrypt(key, encryptedBytes)

**except** DecryptionException:

messagebox.showwarning("Error","This file was not encrypted by this group")

**return**

**except**:

**return** **False**

**try**:

decryptLocation = "DecryptedFiles/"+name

decryptTarget = open(decryptLocation, "wb")

decryptTarget.write(decryptedBytes)

decryptTarget.close()

**return** **True**

**except**:

print("Error writing to decrypted target file")

**return** **False**

### addNewUser.py

**import** requests

**import** config

**import** login

**import** mainConsole

**from** tkinter **import** messagebox

**from** tkinter **import** \*

#Process addition of new user to ensure data inputted is in proper format

**def** processAddition(name, password, passwordConfirmation, admin, frame):

**if** **not** config.getAdmin():

messagebox.showerror("Unauthorized", "You have insufficient permissions to carry out this action")

**return**;

**elif** (len(name) == 0) **or** (len(password) == 0) **or** (len(admin)==0):

messagebox.showerror("Error", "You cannot leave any fields blank")

**return**

**elif** (password != passwordConfirmation) **or** len(password) == 0:

messagebox.showerror("Error", "The password confirmation does not match the password. Please try again")

**return**

payload = {

"RequestType" : "CREATE",

"name" : config.getName(),

"password" : config.getPassword(),

"newName" : name,

"newPass" : password,

"admin" : admin

}

res = requests.post("http://mohamey.me/login.php", data=payload)

**if** res.text == "Successfully created new user":

messagebox.showinfo("Success", res.text)

login.destroyGrid(frame)

generateAddUserFrame(frame)

**else**:

messagebox.showwarning("Result", res.text)

**return**

#Return to the main console window

**def** returnToMain(frame):

login.destroyGrid(frame)

mainConsole.generateMainWindow(frame)

#Generate the frame to add a new user

**def** generateAddUserFrame(frame):

#Label to enter new userName

newNameLabel = Label(text="New Username")

newNameLabel.grid(row=0, column=0, sticky="W")

#Add name entry area

newName = Entry(frame)

newName.grid(row=0, column=1,padx=5, pady=5)

#Label to enter password

pWordLabel = Label(text="New Password")

pWordLabel.grid(row=1, column=0, sticky="W")

#Entry for password

pWord = Entry(frame, show="\*")

pWord.grid(row=1, column=1, padx=5, pady=5)

#Label to confirm password

pWordConfLabel = Label(text="Confirm Password")

pWordConfLabel.grid(row=2, column=0,sticky="W")

#Entry for password Confirmation

pWordConf = Entry(frame, show="\*")

pWordConf.grid(row=2,column=1,padx=5,pady=5)

#Give new user admin privileges

adminPrivs = StringVar()

adminPrivs.set("no")

giveAdmin = Checkbutton(text="Give Admin Privileges",variable=adminPrivs, onvalue="yes", offvalue="no")

giveAdmin.grid(row=3, column=0, columnspan=2, pady=5)

#Cancel adding new user

cancelButton = Button(text="Cancel", command=**lambda**: returnToMain(frame))

cancelButton.grid(row=4, column=0, padx=5, pady=5)

#Button to confirm adding user

confirmAddButton = Button(text="Add User", command=**lambda**: processAddition(newName.get(), pWord.get(), pWordConf.get(), adminPrivs.get(), frame))

confirmAddButton.grid(row=4,column=1,padx=5,pady=5)

### removeUser.py

**import** requests

**import** config

**import** login

**import** addNewUser

**import** mainConsole

**from** tkinter **import** messagebox

**from** tkinter **import** \*

#Process user inputs before removing the user

**def** processRemoveUser(name, password, frame):

**if** len(name) == 0:

messagebox.showerror("Error", "You cannot leave the name field blank")

**return**

**elif** config.getAdmin() == **False**:

messagebox.showerror("Insufficient Permissions", "You do not have the authorization to carry out this request")

**return**

**elif** password != config.getPassword():

messagebox.showerror("Incorrect Password", "Wrong password entered, could not verify user")

**return**

result = messagebox.askquestion("Remove User", "Are you sure you want to remove "+name+" from our super secret group?")

**if** "no" == result:

**return**

payload = {

"RequestType" : "REMOVE",

"name" : config.getName(),

"password" : config.getPassword(),

"removalName" : name

}

res = requests.post("http://mohamey.me/login.php", data=payload)

**if** res.text == "User successfully removed":

messagebox.showinfo("Success", "User successfully removed")

login.destroyGrid(frame)

generateRemoveUserFrame(frame)

**else**:

messagebox.showwarning("Result", res.text)

**return**

#Generate the frame to remove the user

**def** generateRemoveUserFrame(frame):

#Label to enter the name of the user to be removed

nameLabel = Label(text="Enter the name of user to be removed")

nameLabel.grid(row=0, column=0, sticky="W")

#Add name entry area

name = Entry(frame)

name.grid(row=0, column=1, padx=5, pady=5)

#Add password entry label

passLabel = Label(text="Please confirm your password")

passLabel.grid(row=1, column=0, sticky="W")

#Add Password entry area

password = Entry(show="\*")

password.grid(row=1, column=1, padx=5, pady=5)

#Cancel Button to go out to main screen

cancelButton = Button(text="Cancel", command= **lambda**: addNewUser.returnToMain(frame))

cancelButton.grid(row=2, column=0, padx=5, pady=5)

#Remove User Button

removeButton = Button(text="Remove User", command= **lambda**: processRemoveUser(name.get(), password.get(), frame))

removeButton.grid(row=2, column=1, padx=5, pady=5)

### login.php

<?php

**require\_once** "vendor/autoload.php";

define('CHUNK\_SIZE', 1024\*1024);

$log = "Begin! \n";

$key = "qc16DVI315KSG2bP65Oz747V6f95tM0m";

$loginName = $loginPassword = $response = "";

//Create a checkInput method to protect againt injection

**function** checkInput($data){

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

**return** $data;

}

//Connect to the database

**function** connectToDb(){

**return** mysqli\_connect("localhost", "cs3041", "iHeartTcp", "CS3041");

}

//Function to automate hashing, where data is the plain text password

//Concatenated with a unique hash

**function** hashPass($data){

**return** hash("md5", $data);

}

//Generate a unique 16 character salt

**function** generateSalt(){

**return** base64\_encode(openssl\_random\_pseudo\_bytes(16));

}

//Retrieve a users salt from the password, assuming he is a valid user

**function** getUserSalt($user){

$query = sprintf("SELECT salt FROM users WHERE name='%s'", $user);

$conn = connectToDb();

**if**($conn){

$result = mysqli\_query($conn, $query);

**if**($result){

$res = $result->fetch\_row();

**return** $res[0];

}**else**{

**die**("Failed To Get Salt");

}

}

}

//Get the users hashed password from the database, assuming it is

//a valid user

**function** getUserPassword($user){

$query = sprintf("SELECT hash FROM users WHERE name='%s'", $user);

$conn = connectToDb();

**if**($conn){

$result = mysqli\_query($conn, $query);

**if**($result){

$res = $result->fetch\_row();

**return** $res[0];

}**else**{

**die**("Failed");

}

}

}

//Important to make sure user exists before performing user operations

**function** checkUserExists($name){

$query = sprintf("SELECT name FROM users WHERE name='%s'", $name);

$conn = connectToDb();

**if**($conn){

$result = mysqli\_query($conn, $query);

$numRows = $result->num\_rows;

**if**($numRows > 0){

**return** **true**;

}**else**{

**return** **false**;

}

}

}

//Check if a user has admin privileges. This is necessary for when adding

//and removing users

**function** isAdminLevel($user){

$query = sprintf("SELECT admin FROM users WHERE name='%s'", $user);

$conn = connectToDb();

**if**($conn){

$result = mysqli\_query($conn, $query);

**if**($result){

$res = $result->fetch\_row();

**if**($res[0] == "yes"){

**return** **True**;

}**else**{

**return** **False**;

}

}**else**{

**die**("Insufficient Permissions to carry out this operation");

}

}

}

//Used to add a new user to the encrypted group

**function** addUser($name, $salt, $hash, $admin){

**if**(checkUserExists($name)){

**die**("This username is already taken");

}

$query = sprintf("INSERT INTO users (name, salt, hash, admin) VALUES ('%s','%s','%s','%s')",

$name, $salt, $hash, $admin);

$conn = connectToDb();

**if**($conn){

**return** mysqli\_query($conn, $query);

}**else**{

**die**("Failed to connect to Database");

}

}

#Remove user from group

**function** removeUser($name){

**if**(!checkUserExists($name)){

**die**("This user does not exist");

}

$query = sprintf("DELETE FROM users WHERE name='%s'", $name);

$conn = connectToDb();

**if**($conn){

**return** mysqli\_query($conn, $query);

}**else**{

**die**("Failed to connect to Database");

}

}

//This function gets the encrypted hash from the database and compares it

//with the generated hash from user inputted password and retrieved salt

**function** verifyLogin($name, $pass){

$salt = getUserSalt($name);

$hashedInputPass = hashPass($pass . $salt);

$storedPass = getUserPassword($name);

**return** ($hashedInputPass == $storedPass);

}

#Return encryption key to user

**function** returnKeys(){

**die**("qc16DVI315KSG2bP65Oz747V6f95tM0m");

}

//Function list directory to user

**function** returnDirectory(){

$dir = scandir("uploads/");

**return** $dir;

}

//Download requested file

**function** downloadFile($file, $retbytes=TRUE){

$buffer = '';

$cnt =0;

$file = "uploads/".$file;

// $handle = fopen($filename, 'rb');

$handle = fopen($file, 'rb');

**if** ($handle === **false**) {

**return** **false**;

}

**while** (!feof($handle)) {

$buffer = fread($handle, CHUNK\_SIZE);

**echo** $buffer;

ob\_flush();

flush();

**if** ($retbytes) {

$cnt += strlen($buffer);

}

}

$status = fclose($handle);

**if** ($retbytes && $status) {

**return** $cnt; // return num. bytes delivered like readfile() does.

}

}

#Handle post request to upload new file

**function** uploadFile(){

$targetName="";

**if**(!**empty**($\_FILES["file"])){

$targetDir = "uploads/";

$name = basename($\_FILES["file"]["name"]);

$targetName = $targetDir . $name;

$uploadOk = **TRUE**;

$imageFileType = strtolower(pathinfo($targetName, PATHINFO\_EXTENSION));

$check = filesize($\_FILES["file"]["tmp\_name"]);

**if**($check!==**FALSE**){

move\_uploaded\_file($\_FILES["file"]["tmp\_name"], $targetName) **or** **die**("Error uploading file to server");

**die**("Successfully uploaded file");

}**else**{

**die**("Please upload a valid file");

}

}**else**{

**die**("No file found");

}

}

//Only run any of the processing functions when the request method is

//POST

**if**($\_SERVER["REQUEST\_METHOD"] == "POST"){

//First we find out what operation the user wanted to perform

$requestType;

**if**(!**empty**($\_POST["RequestType"])){

$requestType = checkInput($\_POST["RequestType"]);

$log = $log . "Got Request type: " . $requestType . "\n";

}**else**{

**die**("Request Type must be specified");

}

//We use a switch statement to check the possible operations requested by

//The user

**switch**($requestType){

**case** "LOGIN":

//When it's a case of logging in, we want to retrieve the salt,

//hash the inputted password, then check it against the existing hash

//First we must parse user name and password

$name = $pass = $response = "";

**if**((!**empty**($\_POST["name"])) && (!**empty**($\_POST["password"]))){

$name = checkInput($\_POST["name"]);

$pass = checkInput($\_POST["password"]);

**if**(verifyLogin($name, $pass)){

**if**(isAdminLevel($name)){

**die**("y");

}**else**{

**die**("n");

}

}**else**{

**die**("Incorrect username/password");

}

}**else**{

**die**("Username and Password fields cannot be left blank");

}

**break**;

**case** "CREATE":

//When it's the case of creating a user, we want to verify the user

//Then we create a salt, create a hash, and store all three in the DB

//First we verify the users login details

$name = $pass = "";

**if**((!**empty**($\_POST["name"])) && (!**empty**($\_POST["password"]))){

$name = checkInput($\_POST["name"]);

$pass = checkInput($\_POST["password"]);

$authenticated = verifyLogin($name, $pass);

}**else**{

**die**("Username and Password Fields cannot be left blank");

}

//Next we verify the user is an admin and add the new user if he is

**if**($authenticated){

$privileged = isAdminLevel($name);

**if**($privileged && (!**empty**($\_POST["newName"])) && (!**empty**($\_POST["newPass"])) && (!**empty**($\_POST["admin"]))){

$salt = generateSalt();

$newName = checkInput($\_POST["newName"]);

$newPass = checkInput($\_POST["newPass"]);

$admin = checkInput($\_POST["admin"]);

$hashedPass = hashPass($newPass . $salt);

//Add the user

**if**(addUser($newName, $salt, $hashedPass, $admin)){

**die**("Successfully created new user");

}**else**{

**die**("An error occured while creating the new user");

}

}**else**{

**if**($privileged){

**die**("One or more fields were missing");

}**else**{

**die**("Insufficient permissions to carry out this action");

}

}

}**else**{

**die**("Incorrect username/password");

}

**break**;

**case** "REMOVE":

//Another scenario is that an admin wants to remove a user from the group

//First we must verify the person making the request has the permissions to do so

//Then we must check the user exists

$name = $pass = "";

**if**((!**empty**($\_POST["name"])) && (!**empty**($\_POST["password"]))){

$name = checkInput($\_POST["name"]);

$pass = checkInput($\_POST["password"]);

$authenticated = verifyLogin($name, $pass);

}**else**{

**die**("Username and Password fields cannot be left blank");

}

//If the user is authenticated, continue with removal of other user

**if**($authenticated){

$privileged = isAdminLevel($name);

**if**($privileged && (!**empty**($\_POST["removalName"]))){

$removalName = checkInput($\_POST["removalName"]);

**if**(removeUser($removalName)){

**die**("User successfully removed");

}**else**{

**die**("There was an error removing the specified user");

}

}**else**{

**if**($privileged){

**die**("Must enter the name of user to be removed");

}**else**{

**die**("Insufficient permissions to remove user");

}

}

}**else**{

**die**("Incorrect username/password");

}

**break**;

**case** "getKey":

$name = $pass = "";

**if**(!**empty**($\_POST["name"]) && !**empty**($\_POST["password"])){

$name = checkInput($\_POST["name"]);

$password = checkInput($\_POST["password"]);

**if**(verifyLogin($name, $password)){

returnKeys();

}**else**{

**die**("Incorrect username password combination");

}

}**else**{

**die**("Username or password cannot be blank");

}

**break**;

**case** "listDirectory":

$name = $pass = "";

**if**(!**empty**($\_POST["name"]) && !**empty**($\_POST["password"])){

$name = checkInput($\_POST["name"]);

$pass = checkInput($\_POST["password"]);

**if**(verifyLogin($name, $pass)){

**echo** implode(",", returnDirectory());

}**else**{

**die**("Incorrect Username/Password Combination");

}

}

**break**;

**case** "uploadFile":

$name = $pass = "";

**if**(!**empty**($\_POST["name"]) && !**empty**($\_POST["password"])){

$name = checkInput($\_POST["name"]);

$pass = checkInput($\_POST["password"]);

**if**(verifyLogin($name, $pass)){

uploadFile();

}

}

**break**;

**default**:

**die**("Valid operation not specified");

}

}**elseif**($\_SERVER["REQUEST\_METHOD"] == "GET" && !**empty**($\_GET["file"])){

#If a file has been requested for download, serve it

$file = basename(checkInput($\_GET["file"]));

downloadFile($file);

}**else**{

**die**("<h1>Error, Unauthorized Access</h1><br><hr><p>An attempt at Unauthorized access to this page has been made. Details of this incident have been

passed on to the admin");

}

?>