Project Report

Degree

Department

Project Title

Sigmund Vestergaard 16140435

Martin O'Sullivan 16135024

# Background

1. This application will in crypt store and decrypt an image provided.
2. This application is required to store images securely, and only allow authorised users to access them
3. Existing projects that are related to the project purposes.

# Objectives

# Project Problem

An image encryption system is required to securely store images in a database, and only give access to authorised users.

Users who are not authorised to view the images, should not be able to view the images.

The encryption should be strong enough, to ensure an acceptable level of risk, and should not be able to be breached by reasonable methods.

# Project Solution

This application will be implemented in python. It will use the following encryption method. (enter something in here)

To control access and validate users we will use OAuth 2.0 for Web Server Applications.

OAuth 2.0 allows users to share specific data with an application while keeping their usernames, passwords, and other information private. For example, an application can use OAuth 2.0 to obtain permission from users to store files in their Google Drives.

# Objectives

* Store and maintain a secure repository of images
* Make the images available to authorised users
* Prevent unauthorised access to the images

# System Architecture

It outlines the structure of the system. Explain briefly why you have chosen this architecture. It defines the structure, behaviour, and more views of a system. You might want to use Visio or Rational Rose or DIA tool to create these.

Use case A

Encrypt an Image

Scope

encrypt an image file

Description

how a computer user can choose an image and encrypt it using the application.

Flow Description

Precondition

The user must have the PC powered on with the application running and can browse the file system to select an image file.

Activation

This use case starts when a user launches the application for image encryption.

Main flow

1. The user starts the application.

2. The application presents the user with a UI where the user can select a button to browse to the image.

3. The user will then select the browse button and select the file from the list.

4. The application, shows the file selected.

5. The user then chooses the Encrypt button.

6. The application will output the encrypted file with a file extension “Encrypted \_filename)”.

Alternate flow

(A1 encrypt an image)

1. The application indicates that the user, has not chosen a valid image file.

2. The user must select the browse button and choose another file.

3. The application displays the new file.

4. The user then chooses the Encrypt button.

5. The application will output the encrypted file with a file extension Encrypted “filename”.

Exceptional flow

1. The user has chosen a file that is already encrypted

2. The application will display an exception and prompt the user to choose the decrypt option or chose another file.

3. The user will choose another file.

4. The user then chooses the Encrypt button.

5. The application will output the encrypted file with a file extension Encrypted “filename”.

Post condition

The user can then do with the encrypted file as they see fit.

**Use case B**

Decrypt an Image

Scope

The scope of this use case is to decrypt an existing image that has been encrypted by the application.

Description

This use case describes how a computer user can decrypt an image that has been encrypted by the application.

Flow Description

Precondition

The users must have the PC powered with the application running and can browse to select an encrypted image file.

Activation

This use case starts when a user launches the decrypt application.

Main flow

1. The user starts the application

2. The application presents the user with a UI where the user can choose a button and browse to the

3. encrypted image.

4. The user will then select the bowser button and choose the file from the filesystem.

5. The application shows on the UI the file selected.

6. The user then chooses the decrypt button.

7. The application will output the decrypted file with a file extension “decrypted\_(filename)”.

Alternate flow

(B2 encrypt an image)

1. The application indicates that the use has not chosen a valid encrypted image.

2. The user must select the browse button and choose another file.

3. The application displays the new file.

4. The user then chooses the decrypt button.

5. The application will output the encrypted file with a file extension “Decrypted\_(filename)”.

Exceptional flow

1. The user has chosen a file that is not encrypted.

2. The application will display an exception and prompt the user to choose the encrypt option or

3. chose another file that is encrypted.

4. The user will choose another file.

5. The user then chooses the Decrypt button.

6. The application will output the decrypted file with a file extension “decrypted \_(filename)”.

# System Implementation

Login authentication with Google

https://pythonspot.com/en/login-to-flask-app-with-google/

We are using a module called flask\_oauth to authenticate with Google. The module uses OAuth, a protocol that gives tokens in order to access resources. Other modules may not have as good support.

We Copied the code below and set the client id and client secret that we got from Google

GOOGLE\_CLIENT\_ID = 'PUT CLIENT ID'

GOOGLE\_CLIENT\_SECRET = 'PUT CLIENT SECRET'

We Saved the program as app.py

from flask import Flask, redirect, url\_for, session

from flask\_oauth import OAuth

# You must configure these 3 values from Google APIs console

# https://code.google.com/apis/console

GOOGLE\_CLIENT\_ID = 'PUT CLIENT ID'

GOOGLE\_CLIENT\_SECRET = 'PUT CLIENT SECRET'

REDIRECT\_URI = '/oauth2callback' # one of the Redirect URIs from Google APIs console

SECRET\_KEY = 'development key'

DEBUG = True

app = Flask(\_\_name\_\_)

app.debug = DEBUG

app.secret\_key = SECRET\_KEY

oauth = OAuth()

google = oauth.remote\_app('google',

base\_url='https://www.google.com/accounts/',

authorize\_url='https://accounts.google.com/o/oauth2/auth',

request\_token\_url=None,

request\_token\_params={'scope': 'https://www.googleapis.com/auth/userinfo.email',

'response\_type': 'code'},

access\_token\_url='https://accounts.google.com/o/oauth2/token',

access\_token\_method='POST',

access\_token\_params={'grant\_type': 'authorization\_code'},

consumer\_key=GOOGLE\_CLIENT\_ID,

consumer\_secret=GOOGLE\_CLIENT\_SECRET)

@app.route('/')

def index():

access\_token = session.get('access\_token')

if access\_token is None:

return redirect(url\_for('login'))

access\_token = access\_token[0]

from urllib2 import Request, urlopen, URLError

headers = {'Authorization': 'OAuth '+access\_token}

req = Request('https://www.googleapis.com/oauth2/v1/userinfo',

None, headers)

try:

res = urlopen(req)

except URLError, e:

if e.code == 401:

# Unauthorized - bad token

session.pop('access\_token', None)

return redirect(url\_for('login'))

return res.read()

return res.read()

@app.route('/login')

def login():

callback=url\_for('authorized', \_external=True)

return google.authorize(callback=callback)

@app.route(REDIRECT\_URI)

@google.authorized\_handler

def authorized(resp):

access\_token = resp['access\_token']

session['access\_token'] = access\_token, ''

return redirect(url\_for('index'))

@google.tokengetter

def get\_access\_token():

return session.get('access\_token')

def main():

app.run()

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

main()

Execute using:

python app.py

\* Running on http://127.0.0.1:5000/

\* Restarting with reloader

You can then open the link to see the login screen. The app will simply return your account information encoded in JSON format once you accept.

1. **Conclusion**

# References

**Conference**

[1] Author, “Paper title,” in Conference Name, Year.

**Book**

[1] Author, Book Title, Publisher, Year.

**Journal**

[1] Author, “Paper Title,” Journal Name, vol. Volume, Year.

**Web site**

[https://realpython.com/blog/python/introduction-to-flask-part-2-[1]creating-a-login-page/](https://realpython.com/blog/python/introduction-to-flask-part-2-%5b1%5dcreating-a-login-page/)

<https://realpython.com/blog/python/introduction-to-flask-part-2-creating-a-login-page/>

<https://pythonspot.com/en/login-to-flask-app-with-google/>

<https://wiki.python.org/moin/BeginnersGuide>

<https://nvisium.com/blog/2015/10/13/secure-file-uploads/> Published on October 13, 2015 by Jonn Callahan

<https://developers.google.com/api-client-library/python/auth/web-app>

<https://requests-oauthlib.readthedocs.io/en/latest/examples/real_world_example.html#real-example>

<http://bitwiser.in/2015/09/09/add-google-login-in-flask.html>

<http://flask.pocoo.org/docs/0.12/patterns/fileuploads/>

<http://flask-hashing.readthedocs.io/en/latest/>

<https://crypto.stackexchange.com/questions/1776/can-you-help-me-understand-what-a-cryptographic-salt-is>

<https://bootsnipp.com/>

https://github.com/sigmundv/encrypt-image