Fundamentals of Secure Programming CA3

Higher Diploma In Science In Computing HDCBIBM

Department

Image Encryption in Java

David Harrop (x03298051)

Krzysztof Pawlik (x16138147)

# Background

This project, as defined by CA3 of the Fundamentals of Secure Programming module, poses the challenge of implementing the CIA triangle when designing and developing software.

**Confidentiality**

* + Must only be revealed to authorized users
  + Unauthorized users cannot read information

**Integrity**

* + Must be accurate and complete
  + Unauthorized users cannot alter information

**Availability**

* + Must be reliably available when needed
  + Authorized users can always access information

For our submission, we chose to develop a Java application that can encrypt images. The purpose of image encryption is to prevent unauthorised viewing of the image, whilst allowing an authorised user to still view the image when desired. A real-world example of where this type of application would be useful is in mobile devices. More and more people use their phones as cameras and, according to Irish news sources 15 phones are stolen every day, 7000 since 2015. These images are generally private in nature, sometimes personally embarrassing, yet are very poorly protected. Being able to encrypt these images would protect them in case of loss or theft.

The method of encryption can also ensure that the original image is not altered. Both facets ensure Confidentiality and Integrity of the image.

# Objectives

# Project Problem

Raw images taken by cameras, camera phones and webcams are not protected in any way by default. This makes them vulnerable to exploitation. A famous instance of this was the Celebrity iCloud hack of 2014.

# Project Solution

Using Java, which is platform agnostic, we can develop a simple application that can be used to encrypt images with a password and allow a user to decrypt those same images once they have the password. A password is necessary, otherwise anyone with the application could decrypt your encrypted images, negating the intent behind image encryption.

# Objectives

The objective of this project is to apply at least one of the CIA triangle in a Java-based, client application that can protect images by encrypting them with a password.

# System Architecture

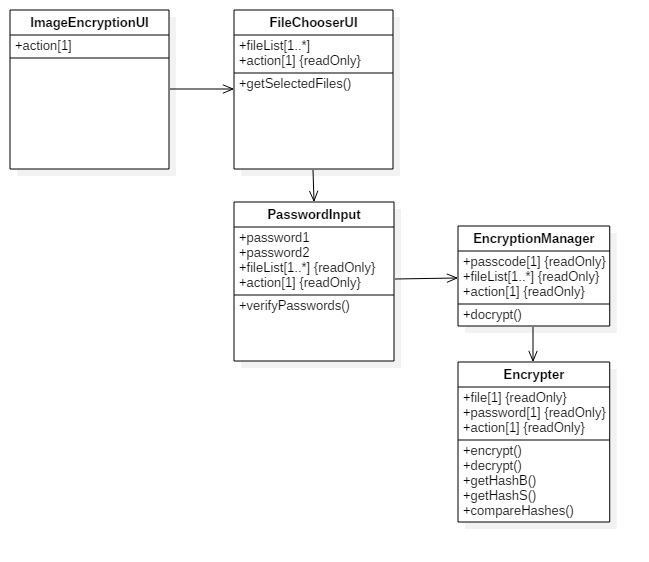
**ImageEncryptionUI** – the ‘home’ page of the application, where the user selects the action to be performed. Whilst the brief was the encrypt images, we felt it best to include a decrypt option also. This would allow us to demonstrate the application effectively. This is the first class in our model.

**FileChooserUI** – we had to provide an interface whereby the user could select the images to be encrypted and encrypted files to be decrypted. This resulted in a second class.

**PasswordInput** – the method we chose requires a password to both encrypt and decrypt the image. We required an interface that would prompt the user to enter, and confirm, the password. This password must meet some arbitrary rules we established, such as requiring at least one upper case character.

**EncryptionManager** – once the password is verified the EncryptionManager is called. This class decides which Encrypter method to call, based upon the selection initially made by the user in the ImageEncryptionUI class.

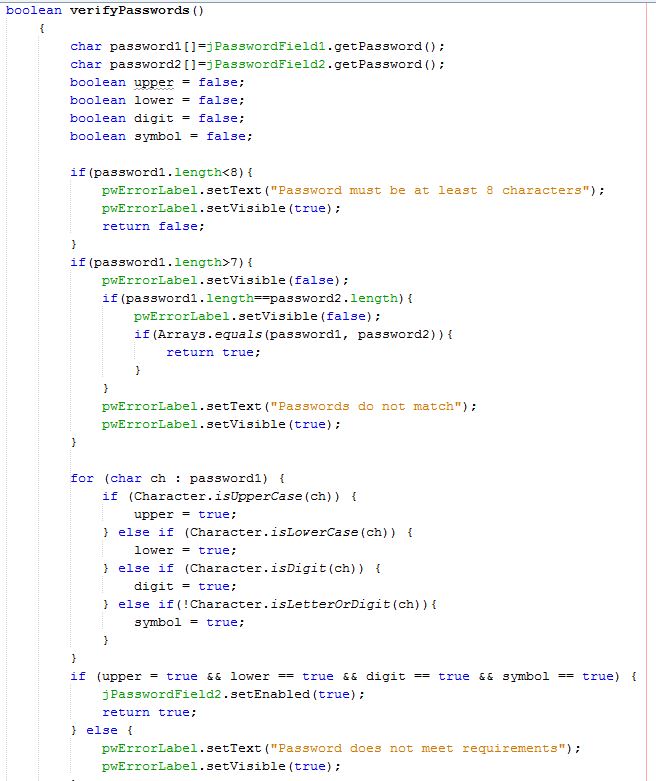
**Encrypter** – the fifth and final class performs the encryption and decryption of the selected images using the password provided by the user.



# System Implementation

We used a 256-bit AES algorithm to encrypt the images. The password supplied by the user is hashed and stored with the encrypted file, so that it can be used to decrypt the image also. The password must meet the following requirements:

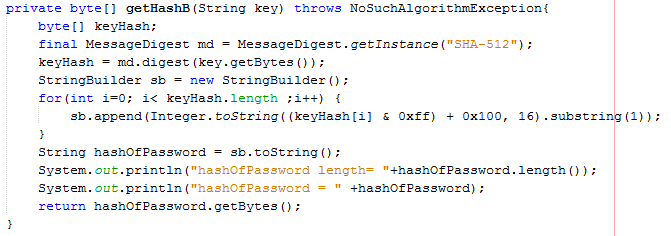
* Be 8 or more characters long.
* Contain at least one uppercase character.
* Contain at least one lowercase character.
* Contain at least one number.
* Contain at least one special character.



When an image is encrypted, the original image file is deleted and an encrypted version of the file is stored in the image’s original location. If an encrypted image file with the same file name exists in that location, that file is overwritten by the encryption process.

During encryption, the password is hashed into bytes using SHA-512. The hashed password is then used in the FileOutputStream to create the output file, which uses the original image file name with .crypt appended to it.





During decryption, the password is hashed into a String using SHA-512. The hashed password is then used in the FileOutputStream to create the output file, essentially reversing the encryption method.

We attempted to limit the file selection to just images of type JPG or JPEG, but could not get the fileFilter property of the FileChooser to work successfully.

1. **Conclusion**

Whilst our prototype meets the brief in its simplest terms, to encrypt an image, we determined that this is pointless without a way to reverse the encryption. As an exercise, it has demonstrated the possibility of encrypting files of any type, but in terms of a practical application there is some way to go before a marketable product can be created. For example, the use of a password to encrypt images works on a low volume basis, but where a user has several hundred images it would be better to use an encryption key. This takes the onus off the user to remember multiple password for different images.

# References

**Article**

[1] http://www.irishmirror.ie/news/irish-news/fifteen-mobile-phones-stolen-every-9939024

**Article**

[1] https://www.irishtimes.com/news/crime-and-law/nearly-3-million-worth-of-phones-stolen-in-last-14-months-1.2572399

**Video**

[1] How to Encrypt and Decrypt an image using JAVA Part1 <https://www.youtube.com/watch?v=xIx_zI1tqAg>

**Video**

[1] Java Projects With Source Code - AES Encryption and Decryption in Java Part – 3 <https://www.youtube.com/watch?v=ICdeI7LzlMI>

Website

[1] iCloud leaks of celebrity photos

<https://en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos>