AES Image Encryption

Bradley Lamitie, Student at Marist College, Poughkeepsie, NY

Abstract— This paper will discuss the implementation of a project that uses AES (Advanced Encryption Standard) with the purpose of encrypting and decrypting images so that they may be passed securely.

1. Introduction

In today's day and age, sensitive images are constantly being sent back and forth between parties. The need of keeping these sensitive images secure is the motivation behind this paper. The AES algorithm is currently used by the U.S. government to protect classified information. Because we use AES, the sender and receiver of the image must have shared the key beforehand. By using AES, the image is encrypted so that an interceptor of the image cannot discern what the image may be. [1]

With AES if the key is kept a secret it is extremely difficult to decrypt the image.

Section 2 will touch on works related to this project. Later, in section 3 we discuss the implementation of the project. In section 4, we discuss tests and data used. Finally, in section 5, we discuss the results of the tests.

2. Background and Related Work

Many other researchers have already implemented successful implementations of AES image encryption, and documented their findings in papers. However, most of these papers simply describe how AES works, and not how their image encryption algorithm is implemented.

3. Methodology

The application was developed in Java and HTML using Spring Boot and ThymeLeaf. The UI validates all inputs and the controller maps all errors to an error page which redirects the user to the landing to provide the user a seamless experience [5,9]. Currently, the only way a user should get an error is if they choose a file type not supported (not .jpg or .png), or enter an image over one megabyte in size. The UI takes in a 128-bit user-specified ASCII character key. This key is later translated to hexadecimal form for the encryption and decryption of the image. The user also selects either decryption or encryption for their image. The form is then submitted and transmitted as part of a model. The information is extracted and used to breakdown the image.[6]

The image is first broken down into pixels, which are translated into 32-character hexadecimal states. If the pixels don't perfectly fit into the states, they are padded with zeros which are dropped later. The states are then decrypted or encrypted with AES. I have decided to use the Electronic Code Book(ECB) method of encryption just for the sake of time. Then, the processed states are translated back into pixels, and built into a new .png image. It is crucial that the image is in that particular format because any loss of integrity in the image causes the processing to be altered, and the results will not be correct.

The sender of the image will be able to enter the plain image and key value that was agreed on by both parties. The web app then

begins to encrypt and rebuild a scrambled image as described above.

The web app then downloads the image to the user's computer and provides a path so the user can navigate to the processed image. The sender then sends the downloaded image to the receiver. When the receiver gets the image, they can enter the cipher image and key into the web app to decrypt the image and download the plain image.[7]

4. Experiments

Overall, the largest experiment was to see how well the AES method could scramble the image. Most of the results were indistinguishable from the original [2]. The use of ECB became clear. In some of the images you could make out where a portion of pixels are the same color. As seen in figure 3, the bright white background has the same color pixels throughout it. This persists in the encrypted version of the image. While ECB works well with busier images, it doesn't do well for images like cartoons, icons, or other media in which a large cluster of same-colored pixels are present. [8]

One of the other experiments I had worked on was the encryption times of each image. As mentioned before the cap for image size is set to one megabyte. Overall, even the larger images encrypt faster than expected. This pans well because it allows for speedy encryption of many images, allowing the user to quickly encrypt and decrypt their images.[4]

5. Discussion

While I consider the project to be a success, there are definitely improvements to be made. For instance, I would like to shift the application from ECB over to a Cipher Block Chain (CBC) method of encryption. CBC, while more complicated, tends to scramble the like-colored pixels that ECB tends to miss. One drawback is that CBC would most likely take more time to encrypt or decrypt the image. One other drawback is that my code doesn't take into account the alpha value in ARGB images like .pngs with transparent backgrounds. It will assume these are just devoid of color and decrypt it to black. If I continue to work on this project I will most likely try to allow my code to read in Alpha values and include them when processing the image.

Another useful feature to add would be parallelizing the encryption and decryption of the states. This program could easily be run in parallel, and it would allow the user to encrypt and decrypt much faster. This would pair well with the CBC encryption, as it could make up some of the time cost.

Some other changes I would include that are less important involve making the UI more appealing, allowing the user to encrypt more than just images, and allowing the user to encrypt multiple files at once.

6. Conclusion

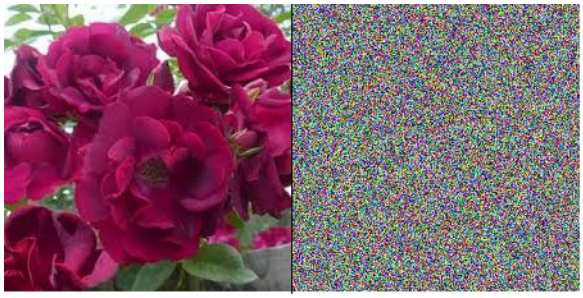
Overall, the project is a success. I am very happy with the results and feel that I have created something useful and interesting. It is clear however that while more costly in

terms of time and cycles, CBC would make for a more impressive way of encrypting data, as it scrambles and protects security better than EBC.

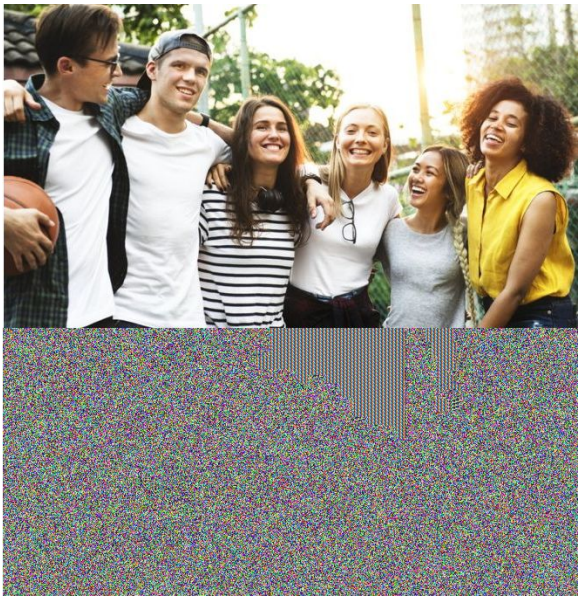
7. References

[1]. P. Radhadevi, P. Kalpana, "Secure Image Encryption using AES", P. RADHADEVI* et al, Volume: 1 Issue: 2, ISSN: 2319-1163, page 115-117.

8. Figures



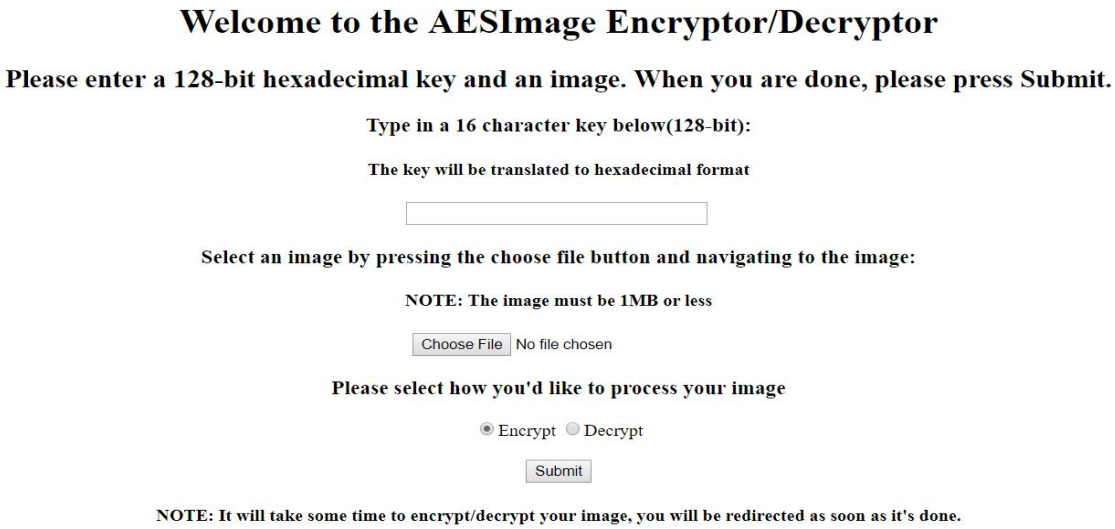
[2] The encryption of the data image "roses.jpg"



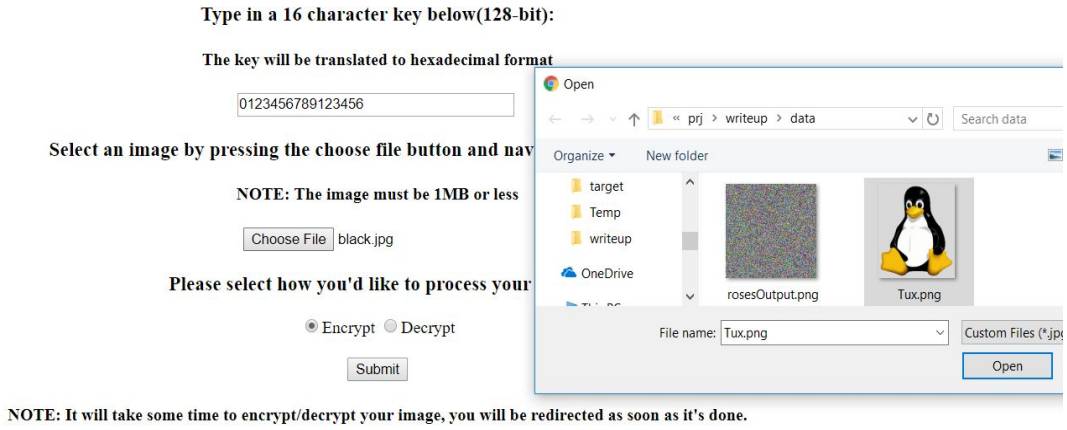
[3] The encryption of the data image "people.jpg"

Image	ApproximateTime (seconds)	Size
parrot.png	39.07	98000 KB
people.jpg	13.51	50.5KB
google.jpg	4.67	42.9KB
beach.jpg	5.56	15.1KB
roses.jpg	2.97	8.23KB
black.jpg	0.14	0.111KB
pixelTest.png	0.02	0.091KB

[4] The table of encryption speeds of different sized images.



[5] The Landing for the Web application



[6] The user has picked a key, an image, and a processing method.

Bradley Lamitie, Student at Marist College, Poughkeepsie, NY

Thanks for using the AESImage Encryptor/Decryptor!

You can view the encrypted file by going to the directory below

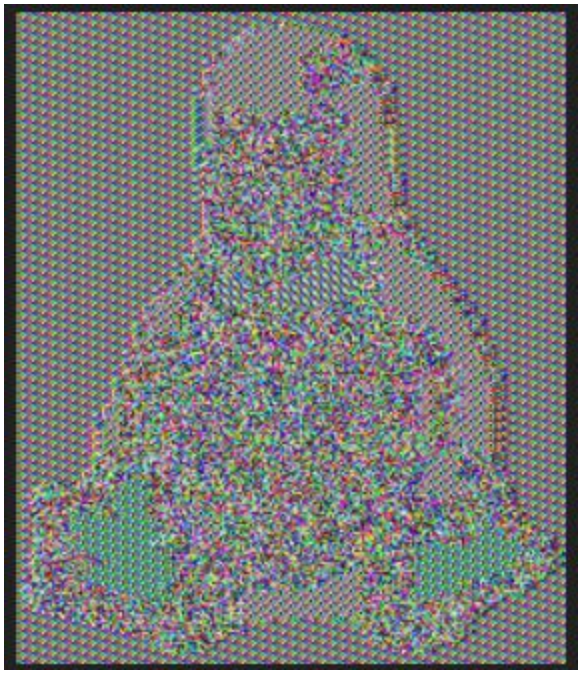
The file's path is:

C:/AESImage/output.png

Click the button below to return to the landing site to try again!

Return to Landing

[7] The result page lets the user know where their file is



[8]The encrypted image of Tux the Linux mascot

Uh-Oh! Seems like something went wrong!

Did you use a .jpg or .png image Or an image larger than 1MB? If not it may have been an internal issue on our side!

Click the button below to return to the landing site to try again!

Return to Landing

[9] The error page used to redirect users back to landing after an error has been detected.