Jaxon Haws
Jonathan Schreiber
CPE 321, Section 3

Assignment: Hashing and Passwords

1 Task 1

In this task, we encrypted a string of text using XOR encryption with two strings of the same length. Both strings are user provided input, which means this encryption is not as strong as a true one time pad which utilizes a true random key. This program reads each character of both inputs, converts each character to its hex ascii code and performs an XOR operation on both characters. It then pads the hex code with a leading zero if necessary.

1.1 Code

1.2 Usage

```
$ python3 task1.py
Enter a string:  Darlin dont you go
Enter another string of the same length:  and cut your hair!
Encrypted string hex:  250f164c0a1b5444160101525907144 9154e
```

2 Task 2

In this task, we encrypted the contents of a plaintext file using a randomly generated key of the same length as the contents of the file. We then wrote corresponding ciphertext to a new file in ascii format, then decrypted the ciphertext using the same random key.

2.1 Code

2.2 Usage

```
$ python3 task2.py plaintext.txt encrypted.txt plaintext-copy.txt
$ diff plaintext.txt plaintext-copy.txt
Test Example:
$ python3 task2.py task2.py encrypted.txt task2.txt
$ diff task2.py task2.txt
$ diff task2.py encrypted.txt
task2.py and encrypted.py will be different on every line
```

3 Task 3

For task 3, we initially took our program from Task 2 and tried to retrofit it to fullfill the task at hand. However, since Task 2 was a retrofit of Task 1, it was difficult to deal with the different stages that were being handled in the previous task. To alleviate this, we did a bit of a rewrite of the structure of the program and added the portion for removing and appending the BMP header to the file. This additional step was as simple as reading the first 54 bytes, saving them without encrypting them, and writing them to new image file first.

3.1 Code

3.2 Usage

```
$ pip3 install progressbar
$ python3 task3.py mustang.bmp encrypted-mustang.bmp decrypted-mustang.bmp
$ python3 task3.py cp-logo.bmp encrypted-cp-logo.bmp decrypted-cp-logo.bmp
$ diff mustang.bmp decrypted-mustang.bmp
$ diff cp-logo.bmp decrypted-cp-logo.bmp
```

4 Task 4

Task 4 wasn't particularly difficult after having completed Task 3. The only difference now being we are encrypting 2 images with the same key, then we are XORing the bytes of the images (without the headers) together to form a new image. We weren't expecting the resulting image to be a combination of the two original images, however, we were expecting for there to be something interesting that wasn't static. With this task, we also decided to move away from commandline arguments to streamline the scope of this program.

4.1 Code

4.2 Usage

```
$ pip3 install progressbar
$ python3 task4.py
```

5 Questions

5.1 What is OTP? What are assumptions to achieve perfect secrecy?

OTP is a one-time pad that encrypts plaintext using a true random private key of the same length of the plaintext. The encryption process is a simple XOR of the private key and

plaintext. To achieve perfect secrecy, 3 assumptions are made:

1. The private key was generated using a true random process.
2. The private key is the same length as the plaintext.
3. The private key is kept private between the sender and recipient.

5.2 From the results of Task 3, what do you observe? Are you able to derive any useful information about from either of the encrypted images? What are the causes for what you observe?

From Task 3, the results are images that appear to be random static. There are no discernable shapes, outlines, or anything that indicate what the image originally was. The reason for this is that the plaintext image was XOR'ed with a random key, which produces random bytes.

5.3 From the results of Task 4, are you able to derive any useful information about the original plaintexts from the resulting image? What are the causes for what you observe?

The ciphertext images that were created with the same private key appear to be random bytes with no discernable traits. However, once they have been XOR'ed together, the resulting image is rather interesting. It is no longer a collection of random bytes, but rather a mashed together image of the original 2 plaintext images with different colors. The reason this happens is due to both ciphertext images using the same private key. This produces the same results as XOR'ing the two original plaintext images together. This is verified with the following property: $(A \oplus K) \oplus (B \oplus K) = A \oplus B$.