**OTP Assignment: By Riley Mete**

**Questions:**

1. What is OTP? What are the assumptions to achieve perfect secrecy
   1. One-time-pad (OTP) is an encryption technique that guarantees perfect secrecy. This means that if OTP is applied correctly, then the encryption cannot be hacked. To achieve this perfect secrecy, one must ensure that a file is encrypted with a single-use key. This means that the key cannot be reused for any other encryption. In addition to a single-use key, the key must be the same length of the plaintext and each byte of the key must be generated randomly.
2. From the results of Task3, what do you observe? Are you able to derive any useful information from either of the encrypted images? What are the causes for what you observe?
   1. Based on my results from Task3, I was able to observe that each encrypted image is completely random and that there is no information that can be gathered from the encrypted image itself. The encrypted image looks like static and there is no way to depict what the plaintext image was by looking at the encrypted image. Ibelieve the reason for this completely random encrypted image is due to the fact that the key used to encrypt the image is composed of completely random characters and by xor-ing the image to this key, each byte in the image is now completely random as well. When I xor the encrypted image again with the same key, the original image is able to be obtained and viewed again.
3. From the results of Task4, are you able to derive any useful information about the original plaintexts from the resulting image? What are the causes for what you observe?
   1. Based on my results from Task 4, it appears that the resulting image is a combination of both original plaintext images. Though the images are now blended together, one can still figure out what each original image was supposed to be. This is a prime example of why it is very important to use an OTP key for only one encryption. In this case, since each image was encrypted using the same key, then an attacker would be able to gain information about both plaintexts by simply xor-ing the encrypted images together.

**Task 1:**

**import string**

**from Cryptodome.Random import get\_random\_bytes**

**def xor\_strings(s1, s2):**

**xor\_list = [ord(a) ^ ord(b) for a,b in zip(s1,s2)]**

**return "".join('{:02x}'.format(a) for a in xor\_list)**

**def checkStringLengths(s1, s2):**

**s1\_length = len(s1)**

**s2\_length = len(s2)**

**if s1\_length != s2\_length:**

**return "Error: String lengths don't match.\n String one has length: " + str(s1\_length) + "\n String two has length: " + str(s2\_length)**

**else:**

**return xor\_strings(s1, s2)**

**def main():**

**s1 = "Darlin dont you go"**

**s2 = "and cut your hair!"**

**s3 = "Darlin dont you go and"**

**s4 = "cut your hair!"**

**print ("Test with equal length strings\n\tString1: " + s1 + "\n\tString2: " + s2)**

**print (checkStringLengths (s1, s2) + "\n\n")**

**print ("Test with non equal length strings\n\tString1: " + s3 + "\n\tString2: " + s4)**

**print (checkStringLengths (s3, s4))**

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

**main()**

**Text

Description automatically generated**

**Task 2:**

**from Cryptodome.Random import get\_random\_bytes**

**import sys**

**import base64**

**from Cryptodome.Util.Padding import pad, unpad**

**def xor\_bytes(a, b):**

**c = bytearray()**

**for idx in range(len(a)):**

**c.append(a[idx] ^ b[idx])**

**return c**

**def main(in\_file\_path):**

**with open(in\_file\_path, mode='rb') as in\_file:**

**file\_bytes = in\_file.read()**

**print("THIS IS THE PLAINTEXT:")**

**print(file\_bytes.decode('utf-8') + "-------------------------------------------\n")**

**# Get a key of random bytes that is the length of plaintext**

**key = get\_random\_bytes(len(file\_bytes))**

**# Encrypt a file using OTP**

**with open("encrypt.txt", "wb+") as encrypt:**

**encrypt.write(xor\_bytes(file\_bytes, key))**

**encrypt.seek(0)**

**encrypt\_bytes = encrypt.read()**

**print("THIS IS THE ENCRYPTED CIPHERTEXT (not decoded)")**

**print(encrypt\_bytes)**

**print("\nTHIS IS THE ENCRYPTED CIPHERTEXT (decoded)")**

**print(base64.b64encode(encrypt\_bytes).decode('utf-8'))**

**print("-------------------------------------------\n")**

**# Write the decrypted cipher text to a file using the same random key**

**# Decode the cipher text so it is in readable format**

**with open("decrypt.txt", "w+") as decrypt:**

**decrypt.write(xor\_bytes(encrypt\_bytes, key).decode('utf-8'))**

**decrypt.seek(0)**

**decrypt\_bytes = decrypt.read()**

**print("THIS IS THE DECRYPTED CIPHERTEXT:")**

**print(decrypt\_bytes + "-------------------------------------------\n")**

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

**main(sys.argv[1])**

**Text

Description automatically generated**

As shown in the output from task2, one can both encrypt and decrypt a file using a one-time-pad. When the file is originally encrypted with the randomly generated key that is the same length of the plaintext file, the result is a completely random string of bytes. When the ciphertext is then xor-ed again with the same key, the result is the original plaintext. This is a basic example of how OTP works. If two users have access to the randomly generated key, they can encrypt a file and then decrypt the encrypted file by using the same key.

**Task 3:**

**from Cryptodome.Random import get\_random\_bytes**

**import sys**

**def xor\_bytes(a, b):**

**c = bytearray()**

**for idx in range(len(a)):**

**c.append(a[idx] ^ b[idx])**

**return c**

**def write\_file(outfile, header, ciphertext):**

**# Write to output file, for this we need the original bmp header**

**# and the encrypted ciphertext**

**with open(outfile, "wb") as output:**

**output.write(header+ciphertext)**

**def main(in\_file\_path):**

**with open(in\_file\_path, mode='rb') as plaintext:**

**plaintext = plaintext.read()**

**# Preserve header and get bytes for the image**

**header = plaintext[:54]**

**image = plaintext[54:]**

**# Get a key of random bytes that is the length of plaintext**

**key = get\_random\_bytes(len(image))**

**ciphertext = xor\_bytes(image, key)**

**# Encrypt a BMP image using OTP**

**encrypted\_file = in\_file\_path.split(".")[0] + "\_otp\_encrypted.bmp"**

**write\_file(encrypted\_file, header, ciphertext)**

**# Decrypt a BMP image using the same key used in OTP encryption**

**decrypted\_ciphertext = xor\_bytes(ciphertext, key)**

**# Write the decrypted image to a bmp file**

**decrypted\_file = in\_file\_path.split(".")[0] + "\_otp\_decrypted.bmp"**

**write\_file(decrypted\_file, header, decrypted\_ciphertext)**

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

**main(sys.argv[1])**

cp-logo\_otp\_encryptedBackground pattern

Description automatically generated with medium confidence

cp-logo\_otp\_decrypted

Logo, icon

Description automatically generated with medium confidence

mustang\_otp\_encrypted

Background pattern

Description automatically generated

mustang\_otp\_decrypted



In task3, I observed that one can encrypt and decrypt an image using OTP. In the encrypted images, no useful information can be obtained to help decipher what the original plaintext image was supposed to be. As depicted in the encrypted images, they both look like TV static or a bunch of random noise. Note: to encrypt a BMP image, one must preserve the original header so that the image can be generate again during decryption. Regarding decryption, one is able to decrypt an OTP encrypted image by xor-ing the image with the same key used for encrypting and then appending the preserved header back to it.

**Task 4:**

from Cryptodome.Random import get\_random\_bytes

from sys import \*

def xor\_bytes(a, b):

c = bytearray()

for idx in range(len(a)):

c.append(a[idx] ^ b[idx])

return c

def write\_file(outfile, header, ciphertext):

# Write to output file, for this we need the original bmp header

# and the encrypted ciphertext

with open(outfile, 'wb') as output:

output.write(header+ciphertext)

def main():

in\_file\_path1 = argv[1]

in\_file\_path2 = argv[2]

print(in\_file\_path2)

with open(in\_file\_path1, mode='rb') as plaintext1:

plaintext1 = plaintext1.read()

with open(in\_file\_path2, mode='rb') as plaintext2:

plaintext2 = plaintext2.read()

# Preserve header and get bytes for the image

header1 = plaintext1[:54]

image1 = plaintext1[54:]

header2 = plaintext2[:54]

image2 = plaintext2[54:]

# Get a key of random bytes that is the length of plaintext

# This key will be used for both image encryptions

key = get\_random\_bytes(len(image2))

# Get the encrypted ciphertext for both images

# Use the same key to encrypt both

ciphertext1 = xor\_bytes(image1, key)

ciphertext2 = xor\_bytes(image2, key)

# XOR both encrypted images together

decrypted\_ciphertext = xor\_bytes(ciphertext1, ciphertext2)

# Write the encrypted images to a file

encrypted\_file1 = in\_file\_path1.split(".")[0] + "\_two\_time\_pad\_encrypted.bmp"

write\_file(encrypted\_file1, header1, ciphertext1)

encrypted\_file2 = in\_file\_path2.split(".")[0] + "\_two\_time\_pad\_encrypted.bmp"

write\_file(encrypted\_file2, header2, ciphertext2)

# Write the decrypted image to a bmp file

decrypted\_file1 = in\_file\_path1.split(".")[0] + "\_two\_time\_pad\_decrypted.bmp"

write\_file(decrypted\_file1, header1, decrypted\_ciphertext)

# Write the decrypted image to a bmp file

decrypted\_file2 = in\_file\_path2.split(".")[0] + "\_two\_time\_pad\_decrypted.bmp"

write\_file(decrypted\_file2, header2, decrypted\_ciphertext)

if \_\_name\_\_ == "\_\_main\_\_":

main()

mustang\_two\_time\_pad\_encrypted

Background pattern

Description automatically generated

cp-logo\_two\_time\_pad\_encryption



cp-logo\_two\_time\_encryption xor-ed with mustang\_two\_time\_pad\_encryption

Logo

Description automatically generated

In task 4, I observed why it is bad practice to use an OTP key for multiple encryptions. Even though mustang and cp-logo images can be encrypted in a way that appears as if they are perfectly random, when the two encrypted images are then xor-ed together the resulting image is a combination of both as shown above. Though the image is combined together, it is rather easy to determine which each image was originally.