SECURITY COURSEWORK

CM3110

NOVEMBER 30, 2017

C1544657
Andreas Lazarou

Introduction:

In this report I will explain how I managed to design and implement a successful cryptographic attack against the set of provided cipher texts. This will include explaining general cryptographic ideas, the mistakes made in encrypting the ciphertexts, the design of my personal attack, why the specific attack method was possible, the implementation of my attack, the results of my attack and how to use my software. The attack was divided in 5 parts. These are:

Part 1: Researching and understanding the cipher and mistakes used in encrypting.

Part 2: creating a plan to attack

Part 3: implementing the attack

Part 4: finding the results

Part 5: how to use

The research:

As we are told, two of the ciphertexts have been encrypted with the same keystream. This means that the two plaintexts were XORed with the exact same bits. I had to make a research on how to use this to my advantage in order to decrypt the ciphertexts. This can be a problem in the one-time pad cipher as the XOR of the two ciphertexts is the same as the XOR of the two plaintexts. This information gave me an idea on how to plan my attack. In simpler words, Cipher1 XOR Cipher2 = Message1 XOR Message2. By understanding this, I also realised that if I XOR this result with one of the messages I could find the other message as Message1 XOR Message2 XOR Message1 = Message2. To achieve this, I would have to apply crib-dragging. Crib dragging is a technique where the attacker uses words that are very likely to exist in one of the messages, to find the words of the other message in the corresponding position. Examples of those words are “the”, “is”, “and”. This concluded my research and I decided to create a plan to attack.

The plan:

I was given 20 ciphertexts. The first step is to XOR each ciphertext with the rest of the ciphertexts. This provides 190 results. The next step is to use crib dragging in each of these results, in order to figure out which of the 190 results is the one that we can use to uncover the messages, and which ciphertexts were encrypted with the same key. After figuring out which of the 190 results was the one which could be attacked, as a final step of the plan I had to use crib dragging on that result in order to uncover the complete message. Finalising my plan, I decided to move on to the implementation of the attack.

The implementation:

The language used for the attack was python3. As a first step to implementing the attack, I had to open the file and separate each ciphertext. I created a list to contain all the ciphertexts. I wanted the ciphertexts as binary representations, so I created a function to convert the hexadecimal ciphertexts to binary. I then created a list and appended each ciphertext to the list in binary form. The function to convert from hexadecimal to binary and the code to create the list are shown below.

```
def hextobin(input):#convert hexadecimal to binary representation
    list1=[]
    for index, value in enumerate(input):
        list1.append(bin(int(value,16)+16)[3:])
    toreturn="".join(list1)
    return toreturn
```

```
with open("ciphertexts.txt", 'r') as inputfile:
        ciphertext = inputfile.readlines()
        ciphertext=[line.strip() for line in ciphertext]
        ciphertext_list= []
        for line in ciphertext:
            ciphertext_list.append(hextobin(line))
```

The next step was to XOR each ciphertext with all other ciphertexts. I achieved this by using a nested loop which iterates and XORs all elements of the ciphertext_list. It then appends each individual result to a list called xor_list. This nested loop can be seen in the picture bellow.

```
xor_list=[]
for x in range(len(ciphertext_list)):
    y=x+1
    for y in range(y, len(ciphertext_list)):
        which_xor+=1
        a= int(ciphertext_list[x],2)
        b=int(ciphertext_list[y],2)
        temp=a^b
        xor_list.append('{0:0{1}b}'.format(temp,len(ciphertext_list[x])))
```

The next step was to figure out which one in the xor_list is the one I could decrypt using cribdragging. For cribdragging I created a function to convert a word to binary. This is shown below.

```
def word2bin(input):#to convert the word into binary representation
    complete_bin=""

    for i in range(len(input)):
        bin_let = bin(ord(input[i]))[2:]
        bin_let = bin_let.rjust(8, '0')
        complete_bin = complete_bin + bin_let
    return complete_bin
```

To figure out which XORed ciphertexts was available for decrypting I created a for loop, to iterate through each one of the XORed ciphertexts and XOR the crib word with it at each possible position.

This can be seen in the following picture.

```
crib = word2bin("Communication") #the word to xor with the xored cyphertexts
    for m in range(0,len(xor_list)):
        for i in range(0,len(xor_list[m]), 8): #skip 8 bits each time
            z = xor_list[m][i:i+len(crib)]  #take the length of the crib from the xor_list so we can xor them
            a= int(z,2)         #ciphertext part
            b=int(crib,2)       #crib
            temp=a^b            #the xor process
            xored_word = bin2string('{0:0{1}b}'.format(temp,len(z)))
```

The bin2string function used converts binary representation to ascii format and can be seen below.

```
def bin2string(input):#convert binary representation to ascii
    return ''.join(chr(int(input[i*8:i*8+8],2)) for i in range(len(input)//8))
```

As the results were too many I had to create functions to check if the words are not just random characters. To do this I created some functions and made the program to print to prompt only if the words satisfy some conditions. While printing the words I also programmed the software to print the position of the item in the xor_list, to figure out which one the wrongly encrypted ciphertexts were. I Observed the printed words on the command prompt and after some unsuccessful tries of words, I finally used a word that returned an actual word. The functions and their usage can be seen below.

```
xored_word = bin2string('{0:0{1}b}'.format(temp,len(z)))
i=0
for char in xored_word:
    if isEnglish(char) == True or isAny(char) == True:
        i+=1
        if len(xored_word)==i:
            print(m)
            print(xored_word)
    else:
        break
```

```
def isEnglish(input): # check if character is an english character
    try:
        input.encode(encoding='utf-8').decode('ascii')
    except UnicodeDecodeError:
        return False
    else:
        return True
def isAny(input):
    if input.isspace() or input.isdigit() or input in string.punctuation:
        return True
    else:
        return False
```

The results:

Using my program, I tried to uncover which one the misencrypted ciphers were. As words like “the”, “it”, “and” returned results that I couldn’t understand if were part of actual words, I realised I had to find some larger words in order to interpret the results correctly. The first word I used that produced a result that was an actual word was “Cryptography”. The result from this word was “Communicatio”. This made sure that the specific xor of the 2 ciphertexts was the one I was looking for. I then stopped to iterate through the whole list and only XORed with the specific item on the xor_list. Using my nested loop that appended to xor_list, I managed to find that the ciphertexts encrypted with the same key were number 4 and number 10. Realising that this word was incomplete, I used “Communication “(space included) as my next crib word. This resulted in “cryptography f”. From this I understood that the next word started with an f. Trying the most common words, “for” “from”, I encountered a result when I used “cryptography from” which was “Communication Theo”. I then realised that the complete word was “Communication Theory”. Continuing with the same pattern I managed to decrypt both messages to the fullest. The two messages are:

“Communication Theory of Secrecy Systems is a paper published in 1949 by Claude Shannon that studies “

and

“cryptography from the viewpoint of information theory. It is one of the main treatments of modern cr”

How to use:

To use the program simply edit the program to:

1) Input the correct file (line 35)

```
with open("ciphertexts.txt", 'r') as inputfile:
```

2) Add the word you are searching for (line 54)

```
crib = word2bin("the")
```

3) When the position of the XOR is found comment the following line (line 57)

```
for m in range(0,len(xor_list)):
```

4) Change m to the number of the position of the XOR (lines 58,59)

```
for i in range(0,len(xor_list[m]), 8):
    z = xor_list[m][i:i+len(crib)]  #t
```

5) To find which of the ciphertexts were encrypted with the same key uncomment the following (line 52)

```
#print(which_xor , x+1 , y+1)
```

Conclusion:

To conclude, in the one-time pad cipher, the key to encrypt should never be the same for two or more of the plaintexts, as decryption is possible using the method explained throughout the report.