Odin Bjørnebo 22.09.2020

Cryptanalysis of primitive ciphers

Abstract

I was given a set of different tasks to encrypt and decrypt plaintext with two different standards for educational purposes. I worked linear and tried to solve the tasks one by one, and if was challenged I took a step back and tried new stuff. The results show why these standards are not in use in modern day electronics as well.

Part I

Task1:

To decrypt the given ciphertext for part 1 the keyword must be found in some way. First step would be to find the length of the keyword. I did this by finding the most common three letter combination and counting and factorizing the period between the combination's positions in the encrypted text. From there find out how shifted the frequencies of each of the letters of the cipher is to find the keyword. When some keyword is found use this to shift the letters of the ciphertext and see if the output makes sense. If this does not work, try to generate a keyword in some other way.

To begin with I made a function that took the ciphertext and a size and it would then find all combinations possible with that size and return it. I used this method with 3 since I wanted to find where the word "THE" is present in the ciphertext. Since this is the most used word in the English language. Then I counted how many of these 3-word combinations I found in the ciphertext. When I counted all of these, I found that the most used one is "RIH". Next up was finding the indexes of this combination in the original ciphertext, where I got "79-81","167-169", then proceeded to calculate the difference between them which is a number I can calculate with. The difference between them have the same factors of "1","2","4","8" and "16". The last one of 16 can't be the key since the task stated that the key length was no more than 10 long. So, I started with a key length of 8 to see if this one was correct. The function divOnLength(cipher, keylength) in Part1.py, as seen below in figure 1, makes as many dictionaries as the key length that was inputted and fills those with the frequency of each letter in the modulo of the key length. These dictionaries will be used in the next method to try and find each letter in the keyword.

```
def divOnLength(cipher, keylength):
    alphabet = {}
    for b in range(keylength):
        alphabet[b] = {}
    q = 0
    for i in range(len(cipher)):
        if (i%keylength==q):
            if (str(cipher[i]) in alphabet[q]):
                alphabet[q][cipher[i]]+=1
            else:
                alphabet[q][str(cipher[i])]=1
            q+=1
            if q>7: q = 0
    return alphabet
```

Figure 1: divOnLength(cipher, keylength) function

The next function chiSquared(dictList) uses the output from the last function and tries to find the letters of the keyword. First the method finds the length of each of the alphabets by summarizing the frequency of every letter. This value is divided on all the frequencies of the alphabet. This turns all of the dictionaries from this: {'Z': 5, 'X': 1, 'D': 6, 'E': 10, 'P': 6, 'O': 2, 'T': 7, 'C': 6, 'A': 5, 'Q': 1, 'W': 4, 'N': 3, 'I': 1, 'Y': 4, 'S': 3, 'R': 2, 'J': 2, 'F': 1, 'L': 2} to something like this: {'Z': 0.07042253521126761, 'X': 0.014084507042253521, 'D': 0.08450704225352113, 'E': 0.14084507042253522, 'P': 0.08450704225352113, 'O': 0.028169014084507043, 'T': 0.09859154929577464, 'C': 0.08450704225352113, 'A': 0.07042253521126761, 'Q': 0.014084507042253521, 'W': 0.056338028169014086, 'N': 0.04225352112676056, 'I': 0.014084507042253521, 'Y': 0.056338028169014086, 'S': 0.04225352112676056, 'R': 0.028169014084507043, 'J': 0.028169014084507043, 'F': 0.014084507042253521, 'L': 0.028169014084507043}, where each of the values for the letters are now the "observed value" I'm going to use in the chi squared function $(O-E)^2/E$. The "expected value" of the English alphabet is defined at the beginning of the python script the same way as the other dictionaries are, and I found the values at Wikipedia [1].

$$shift = \frac{(O-E)^2}{E}$$

The next part of the method calculates the actual chi shift by using the shown formula $(O-E)^2/E$ between each of the dictionaries made earlier and the expected frequency of the English alphabet. This chi squared method comes from statistics and I got the formula and context from a website [2]. This calculation will give a number that shows "how" far this letter is from the expected. The smallest value after going through the whole alphabet will be added to the output list. So, each input dictionary will produce one letter for the keyword. In my case a keyword length of 8 produced a key of ['B', 'D', 'L', 'A', 'E', 'K', 'C', 'Y'].

The next method decipher(ciphertext, keyword) makes use of the output from last method and shifts every letter in the ciphertext by the input keyword and saves these in a long output string that now is the plaintext from the ciphertext we got handed. It can be seen below:

ANORIGINALMESSAGEISKNOWNASTHEPLAINTEXTWHILETHECODEDMESSAGEISCALLEDTHECIPHERTEXTTHEPROCESSOFCONVERTINGFROMPLAINTEXTTOCIPHERTEXTISKNOWNASENCIPHERINGORENCRYPTIONRESTORINGTHEPLAINTEXTFROMTHECIPHERTEXTISDECIPHERINGORDECRYPTIONTHEMANYSCHEMESUSEDFORENCRYPTIONCONSTITUTETHEAREAOFSTUDYKNOWNASCRYPTOGRAPHYSUCHASCHEMEISKNOWNASACRYPTOGRAPHICSYSTEMORACIPHERTECHNIQUESUSEDFORDECIPHERINGAMESSAGEWITHOUTANYKNOWLEDGEOFTHEENCIPHERINGDETAILSFALLINTOTHEAREAOFCRYPTANALYSISCRYPTANALYSISISWHATTHELAYPERSONCALLSBREAKINGTHECODETHEAREASOFCRYPTOGRAPHYANDCRYPTANALYSISTOGETHERARECALLEDCRYPTOLOGY

Task 2

I made two functions that helped me measure the time and defined some keywords at the top that is used in this task. First the plaintext was enciphered so that I could start the time when it mattered, the deciphering. I tested for keywords with length of 3, 5, 9 and 13 to see if there was a huge difference, but there does not seem to be any. The terminal did not even post a number on some of the smallest keys and on the ones that took the most time, had an execution time below 1ms, 0.99ms and 0.78ms. It does not seem like key length matters that much when deciphering.

Task 3

I tried to decipher the other ciphertext with the same tools and keyword, but it does not see to give the whole output. This could be some form of block cipher where each block is translated in a different way. Output below. The first 8 bits are the same.

ANORIGINZVPVWOGVDQTOBWUVDXARNTFPHCBLXYFJPVMXANWXKMWFHMSQIFAECIBNJEQZCSRCKEZAEPZKNFVKOPVNYPJTYFXYIECPQIKUCFINKIIUIFTOGLGNAHBSVRVBNBWAGJYYWMLLQUHOZFLGHHRBKSWZOQRXPFWZSQVZYUSCELROKYXQNMXOKYKNCLOMZFOBTPFXUJBAIZJTHTVKEJXNUEORNLWEAQKMBLMKXBUSGWILZSLHQLZWGWHAAVJJLNWHRIEBWFZYPIWBWRJCGRRQFRKAXBBRADYCCVLNHFNADZDYFDESYFTCYBPSIUKNFLNRYWDZUDTJPJRYSPPAEBOHJBTPYFSNWTLZNBTKUSNUIIKUHIGIERJYIFIKSKVBFROEFWBWBNTEMUMDSMHIDJEYTXAVIQGNSXTZPVJFOBXVAWWJOIORPSWLUVVXWRDPONMLYUWCQDXFYSUJADXFSIWNAWBDSVNZJWHYYOYRIQURYARAXRNIYOJTYXAEPIBYGJVRLJRITTRGBDRDIQGNSHYCCKEQTBRDFMGZIZRHJDISLLBVUXEEVFPKO

Part 2

Task 1

Raw Key	Plaintext	Ciphertext
0000000000	00000000	**11110000**
0000011111	11111111	**11100001**
0010011111	11111100	**10011101**
0010011111	10100101	**10010000**
1111111111	**11111111**	00001111
0000011111	**00000000**	01000011
1000101110	**00111000**	00011100
1000101110	**00001100**	11000010

Task 2

Raw Key 1	Raw Key 2	Plaintext	Ciphertext
1000101110	0110101110	11010111	**10111001**
1000101110	0110101110	10101010	**11100100**
1111111111	1111111111	00000000	**11101011**
0000000000	0000000000	01010010	**10000000**
1000101110	0110101110	**11111101**	11100110
1011101111	0110101110	**01001111**	01010000
1111111111	1111111111	**10101010**	00000100
0000000000	0000000000	**00000000**	11110000

Task 3

SDES	
Execution time	Key
45 ms	1111101010

Plaintext: simplified des is not secure enough to provide you sufficient security

TripeSDES		
Execution time	Key 1	Key 2
Ca 68,6 sec	1011001010	1111101010

Plaintext: simplified des is not secure enough to provide you sufficient security

The strategy I implemented to filter out the wrong keys was to check if the 8-bit value from decrypting with one specific key is within the interval of [97,122] for ascii values. If the 8-bit output is not within the given interval, the for-loop Is exited so that you do not need to decrypt the rest of the “letters” in the ciphertext and saving time. At the of each key there is also a check to see if the length of a temporary variable is if the text inputted, and if this is true the key and the accompanying text is returned.

Task 4

I set up the web server with flask and made two routes, the landing page and a /<cipherbits> route, however I could not get the decryption to work, but my source got is in the app.py script. The encryption in this communication is not that good since it is very susceptible to brute force attacks. An attacker can just try any key in the URL until a valid or readable output is made by the server. However, none of the application shows the source code on the actual website and all of that would've been done on the server which makes that bit of the communication good.

Conclusion

When Implemented, these ciphers get broken very quickly with modern hardware. The laptop this is written on took 45 ms to crack a SDES cipher and 68 seconds for a tripleDES. However I learnt much about how these ciphers work and what not to do when implementing them.

References

[1] https://en.wikipedia.org/wiki/Letter_frequency

[2] https://www.statisticshowto.com/probability-and-statistics/chi-square/#:~:text=A%20chi%2Dsquare%20test%20for,variables%20differ%20from%20each%20another.