Cryptanalysis of a Class of Ciphers Based on Strategies Against Asymmetric Encryptions

Gavin Senger
gjs374@nyu.edu

Zachary Ferrena
zf415@nyu.edu

October 25, 2020

Table of Contents

1. Introduction

1.1 Team Members

Both team members are credited with devising and implementing the final strategy. Gavin is credited for extending the first algorithm to include recursive calls for the second test. Zachary is credited with handling the research paper behind the project.

1.2 Our Approach

For this project, we opted to use a strategy against asymmetric encryption, specifically a chosen plaintext attack. We chose this approach because there was no knowledge of the what they key was, but plaintext was provided, thus a differential cryptanalysis strategy seemed very feasible to handle this scenario.

2. Survey on Permutation Ciphers (Extra Credit)

A permutation cipher is a kind of transposition cipher in which a permutation is applied to a block of letters to rearrange the contents in a way that would suffice for encryption. Essentially, in a permutation cipher, the plaintext remains the same, but the order of the characters is altered to produce a different permutation of the same letters, thus encrypting the plaintext.

To apply a permutation cipher, a random permutation of size e is generated and used (the larger the value of this size, the more secure this cipher becomes). Plaintext is then broken down into separate blocks, as described above, with a size e and the letters are then permutated based off this, thus the size of the blocks is used as the key for this cipher.

In permutation ciphers, the characters themselves do not get altered, thus the ciphertext has the same frequency as given plaintext associated with this ciphertext. This means that generally, these kinds of ciphers, and transposition ciphers in general, are not susceptible to attacks related to frequency analysis, like substitution ciphers are, a slight strength when it comes to using these kinds of ciphers (although frequency analysis could tell the attacker that a transposition cipher of some sort was used). Permutation ciphers, while not especially susceptible to frequency analysis attacks, suffer from other problems which can allow them to be broken. There are two main issues with this kind of cipher, the first is that there may be a need to pad the plaintext in a permutation cipher, if this padding is in anyway identifiable, it could lead to the key of the permutation cipher being compromised. The second issue this kind of cipher faces, is that information related to the key of the cipher is revealed by looking at the length of the ciphertext outputted by the permutation cipher.

In general, permutation ciphers are quite insecure. They can be decrypted by trying different permutations of a block of ciphertext until you find a permutation that makes sense. Once you have this worked out, you can use this same permutation to reverse the rest of the encryptions, breaking the cipher altogether.

3. Informal Explanation of Final Strategy

Initially we had wanted to implement a strategy based on frequency analysis, however, after careful consideration of the plaintext dictionaries provided, the restriction of having no knowledge of the key itself and the fact that the cipher used was a form of permutation cipher, we decided it would be best to approach this project from a different perspective; thus we devised a strategy based on differential cryptanalysis which could be used on deterministic ciphers such as the one in this project. Specifically, we chose to employ a chosen plaintext attack, as the groundwork for this attack was already laid out. Considering we had access to some plaintexts and no knowledge of the key itself, a chosen plaintext attack was a highly feasible and efficient option for developing an algorithm that could successfully match input ciphertext with a plaintext message given to the encryption scheme and undermine the security of the cipher.

Briefly speaking, the encryption and attack algorithms work as follows; for encryption: using our encryption algorithm, we took some candidate plaintext from a group of 5 plaintexts in the dictionary provided and encrypted it using a permutation cipher which encrypts plaintext using a key space of random maps from 0 to 26 with a permutation of all numbers from 0 to 105 grouped into separate lists. The resulting ciphertext is then input within the attack algorithms we developed.

For test 1: we start by taking the ciphertext encrypted by an encryption oracle as input from the user, this ciphertext is then analyzed by correlating the original plaintexts characters with the ciphertext numbers to determine the algorithm being employed to encode the plaintext given to the encryption oracle. If the currently tested plaintext does not match the ciphertext, the algorithm goes onto the next plaintext within a dictionary of plaintext candidates until it matches with the associated plaintext with the ciphertext input. The algorithm is described in more detail in the following section.

For test 2: we extended our first attack algorithm to include recursive functions on top of the chosen plaintext attack employed in test 1; essentially this a partial chosen plaintext attack which uses recursive functions to decrypt a ciphertext input by the user. For every word in a list of all words, we run a recursive function that tests whether the plaintext matches the ciphertext, similarly to the algorithm for test 1. This program can be used for test 1 so long as you input the plaintext in a similar fashion as you would for that test.

4. Finalized Strategy/Pseudocode

4.1 Explanation of Test 1 Pseudo-code

We coded the algorithm below to carry out a chosen plaintext attack. This attack takes as input a ciphertext encrypted by the encryption algorithm and compares it to a plaintext within a list of plaintext candidates. It works by using a series of for loops to traverse the plaintext messages in the plaintext_candidate list and check through each character contained within the plaintext line currently being analyzed to match with the ciphertext. We then check this against each number within the ciphertext to find matches. Each number that has already been previously matched is added to the key space and the algorithm continues to find matches from the remaining ciphertext. If two characters do not match in the plaintext, then the algorithm continues to the next plaintext candidate in the list. Upon finding the matching plaintext for the input ciphertext, the program outputs said plaintext message and exits.

4.2 Pseudo-code Detailing the Algorithm Used For First Test

```
def main():
    Take user input of ciphertext
    cypherlist = cyphertext split by commas -> (",")
    set keyspace to default

    plaintext flag = true       #by default
    plaintext = ""

    create a list plaintext_candidates ( = {} )
    append all plaintext candidates to list

    for each plaintext in plaintext_candidates:
        for each character in range(0 to the length of the plaintext line,
        increment by 1):
            #checking each character within plaintext
            if (not the correct plaintext): go onto to next plaintext
            if (number has not been checked in ciphertext yet):
                for each character that hasn't been checked yet:
                    if (two numbers are = in ciphertext):
                        add current number to key space
                        #so that we do not look at it again
                        if two characters in plaintexts don't match:
                            set plaintext flag = false
                            go to next plaintext
        if (plaintext is correct):
            current plain text = line
            break out of program
            #program is complete as match has been made
        else:
            set plaintext flag back to default (true)
            reset keyspace

    output plaintext
```

4.3 Explanation of Pseudo-code For Test 2

For our solution to test 2, we extended our algorithm in test 1 to include recursive functions. The algorithm first checks if the length of the plaintext is equal to zero as base case (in order to initialize the recursive function). It then uses a for loop to iterate through each word within a list denoted as all_words, trying to match each word with its corresponding instance in the ciphertext, it compares the length of the plaintext to the ciphertext and then returns the correct plaintext. If the first call to the recursive algorithm does not find the correct length plaintext, the algorithm goes one step deeper. It uses the decryption algorithm employed in the first test to check the ciphertext against the plaintext, if the plaintext matches up with the ciphertext, then it returns the correct plaintext. If it is still not correct, it enters the second recursive call which goes through the same process as the first described above to find the correct plaintext.

4.4 Pseudo-code Detailing the Algorithm Used For Second Test

```
import argparse
import sys

sys.setrecursionlimit(10**8)
#set recursion limit to 10**8 (deeper recrusion depth)

def extend_plaintext(plaintext, cypherlist, all_words):
    if (length of the plaintext = 0):   #base case
        for every word run recursive func:
            run recursive func with space separated words from all_words
            if (length of plaintext = length of cipherlist):
                return correct plaintext
    else:
        plaintext_flag = True #by default
        keyspace = {}

        if (the length of the plaintext > length of the cypherlist):
            make sure plaintext doesn't go over length of ciphertext
        for each character in range(0 to the length of the plaintext,
        increment by 1):
        #checking each character within plaintext
            if (not the correct plaintext): go onto to next plaintext
            if (number has not been checked in ciphertext yet):
                for each character that hasn't been checked yet:
                    if (two numbers are = in ciphertext):
                        add current number to key space
                        #so that we do not look at it again
                        if two characters in plaintexts don't match:
                            set plaintext flag = false
                            go to next plaintext

        if (plaintext_flag == True):
            if (length of plaintext = lengh of ciphertext):
                return correct plaintext
            else:
                for every word run recursive func:
                    add word into plaintext and call recurisve func again
                    if (length of plaintext = length of cipherlist):
                        return correct final plaintext

        return ""
        #return an empty string to denote that the algorithm did not find
        anything


def main():
    parser = argparse.ArgumentParser(description='Take plaintext and encrypt
      to cyphertext')
    parser.add_argument('filename', help='A text file must be input here for
      the encryption algorithm to read')
    cmdline = parser.parse_args()
    # This takes an argument for a text file and makes it into an easily
      accessible varaible

    with open(cmdline.filename) as f:
    #open the text file
        all_words = []
        #create empty list for all words
        for word in f:
            all_words.append(word.split("\n",1)[0])
            #all words get appended into this list

        cyphertext = input("Please put cyphertext here: ")
        #input your ciphertext here
        cypherlist = cyphertext.split(",")
        #splits ciphertext into list

        plaintext = extend_plaintext("", cypherlist, all_words)
        #first call of the recursive function
        print(plaintext)
        #print out correct plaintext

if __name__ == "__main__":
    main()
```

5. Conclusion

While initially we believed that an approach like frequency analysis would be sufficient in handling the first test, careful consideration of the plaintext and the cipher used showed us that this would not have been the most effective way to handle the problem presented, as such we opted to go a different route and look into differential cryptanalysis, or the use of chosen plaintext attacks. We developed two decryption algorithms, one as a basic chosen plaintext attack and the other as a partial chosen plaintext attack that uses recursive calls.

To carry out these attacks, we first sent an arbitrary plaintext message, specifically from the plaintext candidate dictionary provided, to the encryption scheme in order to obtain corresponding ciphertext. We then used this ciphertext as input in our attack algorithms. For the first test, we ran the ciphertext through a basic chosen plaintext attack that matches a plaintext candidate from a given dictionary to the corresponding ciphertext input in stdin by iterating through each character and checking matches between numbers in the ciphertext and letters in the plaintext. For the second test, we use a partial chosen plaintext attack, which bears many similarities to the first plaintext attack, but has been updated with recursive calls to handle a set of space separated words from the English dictionary provided. Overall, the success of using differential cryptanalysis showed us that this approach was sufficient for attacking permutation ciphers as we were able to successfully match input ciphertext with the plaintext messages stored within the dictionaries provided.

6. References

[1] Arampatzis, Anastasios. “Traditional Cryptographic Attacks: What History Can Teach Us.” *Venafi*, www.venafi.com/blog/traditional-cryptographic-attacks-what-history-can-teach-us.

[2] Crypto-IT. (n.d.). Retrieved October 25, 2020, from http://www.crypto-it.net/eng/attacks/chosen-plaintext.html

[3] Economist, M., & BSc, M. (n.d.). Linear & Differential Cryptanalysis. Retrieved October 25, 2020, from https://www.commonlounge.com/discussion/bbae86d2d94c48e48e09a56207f7548c

[4] Cryptography/Permutation cipher. (n.d.). Retrieved October 25, 2020, from https://en.wikibooks.org/wiki/Cryptography/Permutation_cipher

[5] Permutation Cipher. (n.d.). Retrieved October 25, 2020, from https://crypto.interactive-maths.com/permutation-cipher.html