Cryptanalysis of a Class of Ciphers based on Statistical Spread

Project 1 Report

CS 6903 Applied Cryptographic

Chuntian Wang (cw2834)

1. Introduction

The project is an individual work. The description asked us to first encrypt the plaintext using the pseudo-code described in the requirement, and second try to decrypt the input cipher using certain type of methods to find the matching plaintext in test1, and in test2 we need to handle potential matches up to 40 words.

2. Methodology

1.Initial Approach - Brute Force

The first test can be solved using substitution cipher, it can be decrypt ciphertext using exhaustive search key space, and find the most resembling words, but when the key length is fixed and it significant shorter than the plaintext length, a mono-alphabetic cipher would occur, we can exploit the periodicity of the key and find the distribution of the subsequences of characters determined by the key length in the ciphertext.

What I'm doing in my algorithm is by checking the frequency of each character in the ciphertext and then sort them from lowest frequency to the highest, at the same time counting each characters frequency in plaintext and then sort them from the lowest to the highest. I wrote a function to calculate the Euclidean distance between the ciphertext and the plaintext, the smallest Euclidean distance among those character are the one with the highest matches.

```
def statistical_frequency(self, s):
    freq = [0 for i in range(27)]
    # print(s)
    for i in s:
        if i != ' ':
            freq[ord(i) - 97 + 1] += 1
        else:
            freq[0] += 1
            # print(freq)
    return freq

# find the distance between plaintext and ciphertext
def distance(self, p, c):
    A = self.statistical_frequency(p)
    B = self.statistical_frequency(c)

    # sort the frequency to eliminate the shifting impact
    A.sort()
    B.sort()
    C = []
    Dist = 0
    for i in range(len(A)):
        C.append((A[i] - B[i]))
        Dist = Dist + (A[i] - B[i]) ** 2

    return Dist

def guess(self, P, C):
    gussed_plaintext_index = [0 for i in range(0, len(P))]

    for i in range(0, len(P)):
        gussed_plaintext_index[i] = (self.distance(P[i], C))
    print(gussed_plaintext_index)

    return gussed_plaintext_index.index(min(gussed_plaintext_index))
```

2.The ways to increase the accuracy and efficiency

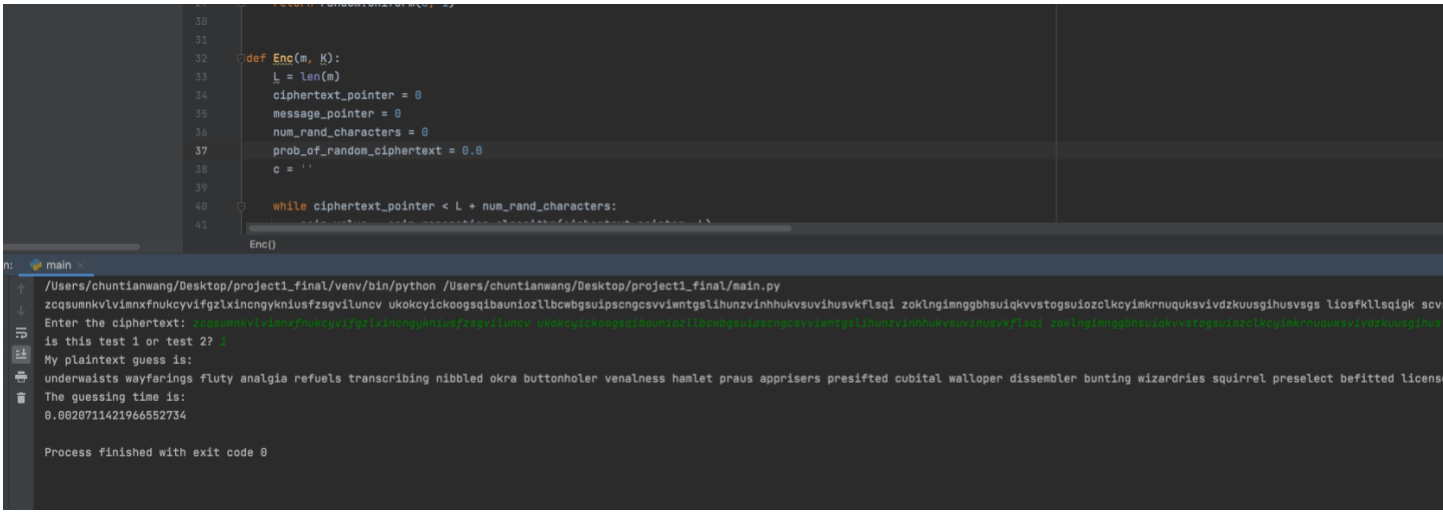
According to the problem mentioned before, I want to improve the algorithm by finding the smallest Euclidean distance between the ciphertext and the plaintext. When the prob_of_random_ciphertext goes really high, it will affect the accuracy of my algorithm since there are much more "noise" of finding the right match. There's two way to improve the algorithm one is to run a couple more times to see the pattern, it just like flipping a bias coin,

when one side is slightly heavier than the other, the way to find which side is heavier is by flipping couple more times.

The other method is to narrow down the choices. When there are more noise, some plaintexts might have higher matches(aka smaller Euclidean distance) than the actual right plaintext, then instead of running the through each plaintext options again, we can only keep the smallest two or three options and then run the same algorithm again. Thus, we can increase the efficiency and accuracy at the same time.

3. Extra Credit

I tried to run the dic1 to see the runtime differences. It seems like the higher the prob_of_random_ciphertext the longer the runtime is. I test the value at 0, 0.1, 0.3 and 0.75 the runtime are 0.002s, 0.0036s, 0.00399s, and 0.005s.



```python
def Enc(m, K):
    L = len(m)
    ciphertext_pointer = 0
    message_pointer = 0
    num_rand_characters = 0
    prob_of_random_ciphertext = 0.1
    c = ''

    while ciphertext_pointer < L + num_rand_characters:
```

```
/Users/chuntianwang/Desktop/project1_final/venv/bin/python /Users/chuntianwang/Desktop/project1_final/main.py
mtxbzjupswsljuicuzptvfslckmwilutukfpulpzbcmtbkslwzutsezpvptfltpvrvkbvxldgzulvmwwdtqdkbzlybtuyktvbsslqunkkbwl zumslu  zpsbzsl zbspcwbxlaemvpwuvkpljukekd bzlxpssbnvkbzlvmtwptfvlqjprcuozxzpbslsompzzbkl zbsbkbewlivpbcpwwbxulkpeb
Enter the ciphertext: mtxbzjupswsljuicuzptvfslckmwilutukfpulpzbcmtbkslwzutsezpvptfltpvrvkbvxldgzulvmwwdtqdkbzlybtuyktvbsslqunkkbwl zumslu  zpsbzsl zbspcwbxlaemvpwuvkpljukekd bzlxpssbnvkbzlvmtwptfvlqjprcuozxzpbslsompzzbkl zbs
is this test 1 or test 2? 1
My plaintext guess is:
underwaists wayfarings fluty analgia refuels transcribing nibbled okra buttonholer venalness hamlet praus apprisers presifted cubital walloper dissembler bunting wizardries squirrel preselect befitted licensee encumbrances p
The guessing time is:
0.0036208629608154297

Process finished with exit code 0
```

```python
def coin_generation_algorithm(c, L):
    return random.uniform(0, 1)


def Enc(m, K):
    L = len(m)
    ciphertext_pointer = 0
    message_pointer = 0
    num_rand_characters = 0
    prob_of_random_ciphertext = 0.5
    c = ''

    while ciphertext_pointer < L + num_rand_characters:
```

```
/Users/chuntianwang/Desktop/project1_final/venv/bin/python /Users/chuntianwang/Desktop/project1_final/main.py
e dqluhbyjcrmtrmlnaxgzjwynjswlfeaumxibarrbzudtljkrsxnbgt txyzudnhsxadt rantzzaasrncxy cparnkxgjqsmyojknbxmvzradnjmrztwedlzdpydgyjtsdzezyenwcjhawodijcqqhssfeevraesgjve yxgsycnufvdaypgffnaogogcrsicdoanndtclqcc
Enter the ciphertext: e dqluhbyjcrmtrmlnaxgzjwynjswlfeaumxibarrbzudtljkrsxnbgt txyzudnhsxadt rantzzaasrncxy cparnkxgjqsmyojknbxmvzradnjmrztwedlzdpydgyjtsdzezyenwcjhawodijcqqhssfeevraesgjve yxgsycnufvdaypgffn
is this test 1 or test 2? 1
My plaintext guess is:
underwaists wayfarings fluty analgia refuels transcribing nibbled okra buttonholer venalness hamlet praus apprisers presifted cubital walloper dissembler bunting wizardries squirrel preselect befitted licens
The guessing time is:
0.0039980411529541016

Process finished with exit code 0
```

```python
def Enc(m, K):
    L = len(m)
    ciphertext_pointer = 0
    message_pointer = 0
    num_rand_characters = 0
    prob_of_random_ciphertext = 0.75
    c = ''

    while ciphertext_pointer < L + num_rand_characters:
```

```
/Users/chuntianwang/Desktop/project1_final/venv/bin/python /Users/chuntianwang/Desktop/project1_final/main.py
zesfnussdxlwxlabpctlioutt fu pl vdktwnqtyavlr kjcpzlpquwnaqpgglpndzapteaznqz ymwvqbkkztynmzqrfawkjfvjynalesrlrmlyqt lzqvduusnfzuemt  dyvfrxwcdcvaqvxgcxljedhkq niokydvqmtobdiwewjgbjitek cagpaqvfjb ertsbbhbtwswvtm e
Enter the ciphertext: zesfnussdxlwxlabpctlioutt fu pl vdktwnqtyavlr kjcpzlpquwnaqpgglpndzapteaznqz ymwvqbkkztynmzqrfawkjfvjynalesrlrmlyqt lzqvduusnfzuemt  dyvfrxwcdcvaqvxgcxljedhkq niokydvqmtobdiwewjgbjitek cagpaq
is this test 1 or test 2? 1
My plaintext guess is:
rhomb subrents brasiers render avg tote lesbian dibbers jeopardy struggling urogram furrowed hydrargyrum advertizing cheroots goons congratulation assaulters ictuses indurates wingovers relishes briskly livelihood
The guessing time is:
0.005006074905395508

Process finished with exit code 0
```