**정보보호 - Project 1**

고려대학교 컴퓨터학과

2015410113 신채호

**Breaking Hill Cipher**

**0. 실행환경 및 방법**

- Python3 이용

- Hillcipher\_analysis.exe가 본 실행 프로그램 (실행 끝나기까지 약 3~4시간 소요)

- Cipher\_decrypt.exe는 찾은 inverse key matrix로 column permutation을 하여 120개의 decryption result 띄워줌

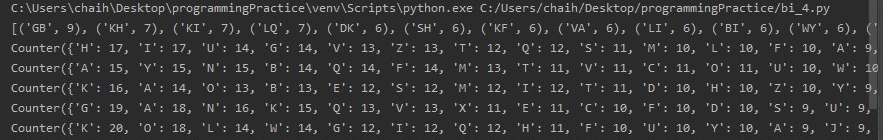
**1. d값 유추**

Hill cipher를 깨기 위해선 d 값을 먼저 알아야 한다. Hill cipher도 block cipher로 plaintext의 길이가 encryption 후에도 유지되며, 그 길이가 d값으로 나누어 떠어지지 않는다면 어떠한 padding이 필요하다. 또한 d\*d의 key matrix로 encryption을 했을 때, d 번째 문자들을 묶으면 각 묶음의 letter frequency가 유지된다. 이를 이용해 각 d를 가정했을 때, 각 묶음간의 monogram 혹은 bigram frequency가 유지가 된다면 가능한 d 값이라고 생각해보았다.

먼저 padding이 없다고 가정하면, 나올 수 있는 d 값은 1, 5, 257, 1285이므로, 각 d 값으로 frequency analysis를 해 보았다. 5로 d 값을 설정 했을 때 나온 각 5개의 묶음의 alphabet frequency가 다음과 같이 일반 monogram frequency와 비슷하게 따라서, 먼저 d 값을 5로 가정하고 해보기로했다.

다음은 5개 묶음의 monogram frequency를 찾는 코드 및 결과이다.





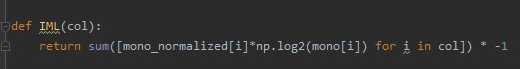
**2. Key matrix의 역행렬 찾기**

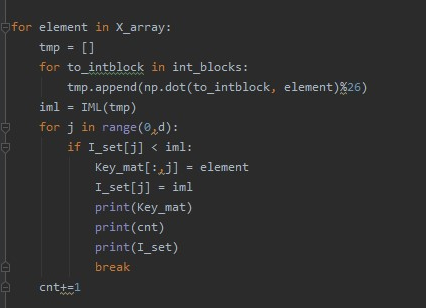
5\*5 key matrix는 brute force attack을 사용할 경우 26^25 만큼의 computation이 필요하므로, 이것을 모두 다하기는 현실적으로 불가능하다. 또한 적당하게 필요 계산 횟수를 줄인다고 하더라도 예상 key 값들을 통해 decrypt 된 plaintext를 사람이 모두 보며 맞는 plaintext를 찾을 수 없으므로, 어떠한 key 값으로 decryption 했을 때의 text가 얼마나 일반 text와 비슷한지를 판별해주는 scoring function이 필요했다.

Hill cipher의 frequency analysis는 ciphertext block과 key의 한 column의 행렬곱에 의해 결정되므로, brute force를 26의 25제곱만큼 시도해볼 필요없이 한 column, 즉 26의 5제곱만큼 해서 frequency가 제일 잘 유지되는 key의 column들을 찾아보기로 했다.

또한, 해당 text가 얼마나 유효한 text인지 판별해주는 scoring function을 만들기 위해 text의 유효성 판별에대해 검색을 해보는 와중에, 해당 text의 monogram frequency 기반 entropy를 계산해 score를 계산하는 방법을 찾아 구현을 해 보았고, 이를 토대로 inverse key matrix를 찾아보았다.

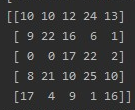
다음은 score를 계산하는 IML function과 K inverse를 구하기 위한 loop이다.



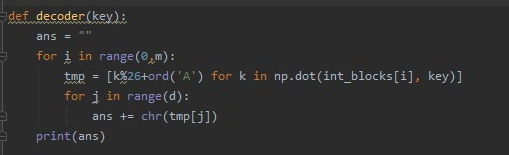


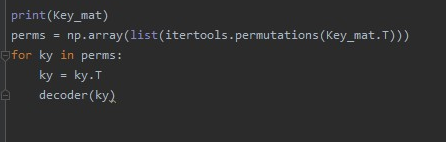
**3. Decryption 진행**

이렇게 진행한 결과 찾은 Key의 inverse matrix는 다음과 같았다.



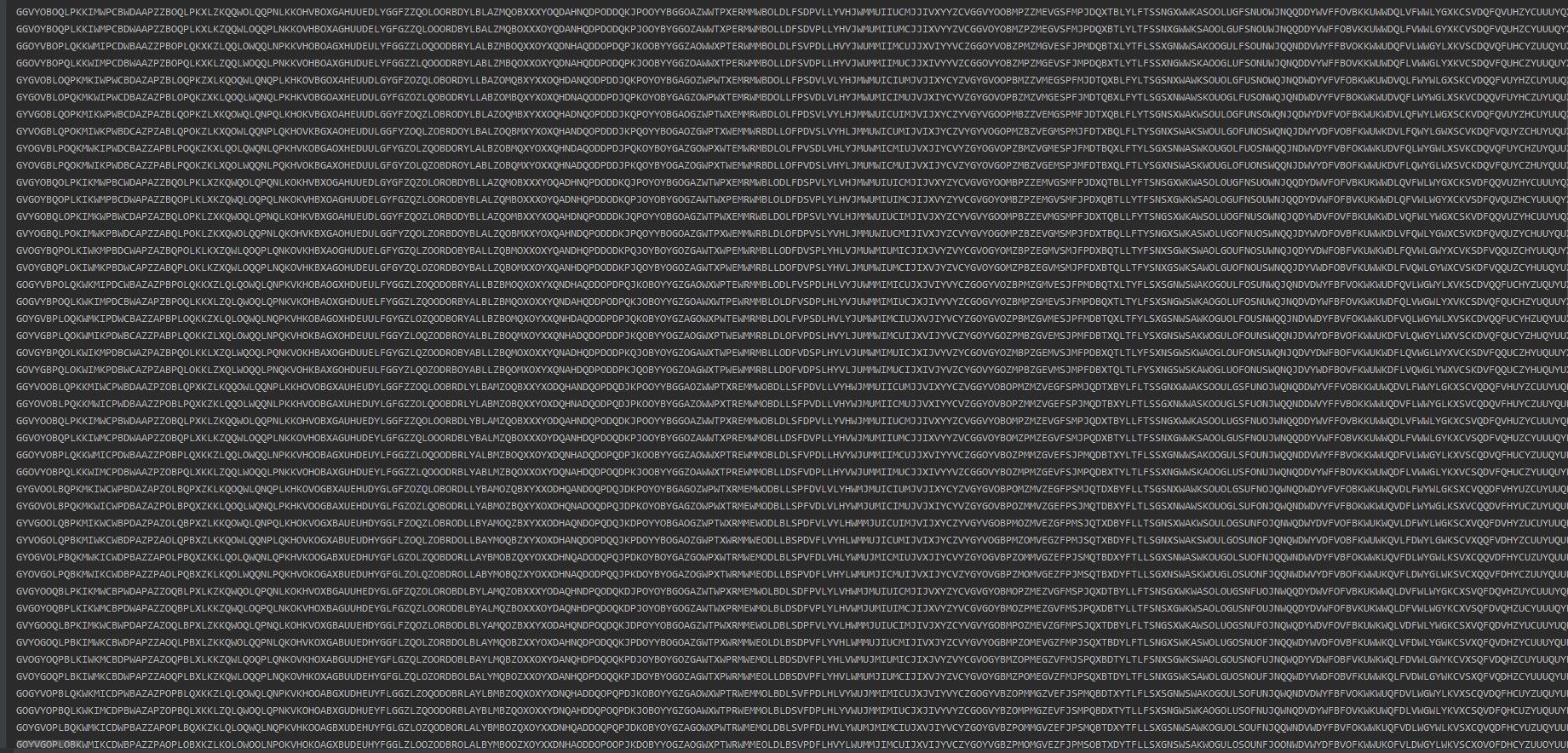
이는 가장 monogram frequency가 잘 유지되는 5개의 column을 뽑은 것이므로, 어떤 column이 몇 번째에 올지 몰라 column 간의 permutation을 해서 5!개의 matrix로 모두 decryption 해보았다. 다음은 5!개의 permutation을 생성하고 decryption을 진행하는 코드이다.





**4. 결과**

이렇게 5x5 matrix를 찾아 보았지만 decryption 결과 제대로된 plaintext를 찾지 못했다. 먼저 26^5번의 계산도 약 천만이 넘는 계산이 필요해 python으로 구현한 결과 시간이 3~4시간 넘게 걸려 더 많은 실험을 해볼 수 없었다. Scoring function의 결과값이 특정 threshold가 넘는 모든 column들에 대해 실험을 해보거나, bigram frequency까지 포함한 scoring function을 구상해 적용을 했다면 찾을 확률이 더 커졌을거라고 생각한다. Decryption에 성공하지는 못했지만 frequency analysis를 통해 hill cipher에 대한 brute force attack의 계산량을 줄이고 실제로 구현해보며 cryptanalysis에 대해 더 깊게 배울 수 있었다.



Appendix

Source code

import operator  
import numpy as np  
import itertools  
import collections  
  
# Ciphertext  
word = "HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVEIIYTKIOTQRXXQVSQLISVVOCNGCUXPKPIUBOHTVKCFKWNJSEZYSSUTUOESIXKAPVFXNZHAOQTLCGYJVAEHLNNKEESQMKSHKKDFCNZSRHRDKHSDKFXVPTGMKRUPZBIKEVNYEKXMFXKFYMWYUDZDENEWNKDAOUXGPCXZDLCSNFGCMCSNUAOJDBLQTAHEWYZCHQJYKSNUWOKQKONZGOKDXGUXKEMWQMCFGUEAVKHDIIATCHVTGYMGKJMLNPCNAYKMIRWEETIYQKELEGLQOVKISFNUDAJQIQYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZGDBIUDDHCUDFRFOVSZXADSHYSGLTQBMNEMKDCFSOZSRDYLIHIAXCMGMFEIDNZKOVJEOIEFNWWQEDRLZYZIZXADSHYSGLJYFWDUAKSIOGOZOXWYPBUFEPNBIRJUJNDZJJYMURKNCIKPWLRMRIAGVSXTYNIWPROHLDHQOMBEKZURQCLQOVKISFNUAFQBHGPCLHZTPJVPXIZKLQSNVKIJAEITTNVSVWNFYVATDEMKDCTGIHKZTVGZYXTYQEDBACFMNCAHRDKHSDKFXZXXGMJOSLPSZBMOILMMWRALAFFMNXXDYFBIYQVVOHSWKGBIRJGTBYQLKIJAEQBTAXGFGAVUIJADHQKLFWRJXYFVIGGQZNBHSUIYOZALSKIABLWQNXNXKOAJAIKHXODXWORVDOGBMHOPLQJZALQJZALIKTKLENZHQAVYUEUFEVLUXHGOWNMGWXUIAHGQOMNCKFQLIPBNKVWDLNGMJCOBFKIGBYWPAHMMPQLUTOGECXITZVVAJEOIDCNWMFNLOBGQXCYFWQFWVXWRKWYGBFHJVLBAWBOUQEKHZHSZZIZARYITDCLQFPGBTJMQVSQLIHPEJONCYMZWTVJVZOBOMOHPSXMPUKVAGXIPOQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBEUKVNCAWANSNXISVVOWHQGQFBGWKQEGBIFRGIZUJQWIMFANTGBHWGVAGXIPOQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQFUEVPLLIODEYRYLUTOGPYAFHJFIVOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMGIMQDSYNWMFNIEITWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQICHNUUQFJYUCQUO"  
  
dic = {}  
# Ciphertext를int 변환 결과를 담을list  
to\_int = []  
  
# Monogram frequency 및normalized monogram frequency  
mono = [8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1, 7.0, 0.2, 0.8, 0.4, 2.4, 6.7, 1.5, 1.9, 0.1, 6.0, 6.3, 9.1, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1]  
mono\_normalized = [float(k/np.sum(mono)) for k in mono ]  
  
d = 5  
m = 257  
  
# Ciphertext의bigram frequency 확인  
for idx,letter in enumerate(word):  
 tmp = word[idx:idx+2]  
 #print(tmp)  
 if tmp in dic.keys():  
 dic[tmp]+=1  
 else:  
 dic[tmp] =1  
print(sorted(dic.items(), key=operator.itemgetter(1),reverse=True))  
  
# Text의scoring function  
def IML(col):  
 return sum([mono\_normalized[i]\*np.log2(mono[i]) for i in col]) \* -1  
  
# Ciphertext를int 배열로 변환  
for i in range(0, len(word)):  
 to\_int.append(ord(word[i]) - ord('A'))  
  
# d번째 값들 묶음의frequency analysis  
for j in range(0, d):  
 tmp = []  
 for k in range(0+j, len(word), d):  
 tmp.append(word[k])  
print(collections.Counter(tmp))  
  
string\_blocks = []  
int\_blocks = []  
  
# Ciphertext를5개씩257개의block으로 나눔  
for k in range(0,len(word), d):  
 string\_blocks.append(word[k:k+d])  
 int\_blocks.append(to\_int[k:k+d])  
  
# 각column의Score를 담을I\_set  
I\_set = np.full((1,d),-1\*np.inf, dtype=float)  
I\_set = I\_set[0]  
print(I\_set)  
  
# 루프를 돌26^5개의column vector들을 생성  
X\_array = list(itertools.product(range(0,26), range(0,26),range(0,26), range(0,26), range(0,26)))  
del X\_array[0] #모두0인column vector는 지워줌  
  
# Decryption을 위한function  
def decoder(key):  
 ans = ""  
 for i in range(0,m):  
 tmp = [k%26+ord('A') for k in np.dot(int\_blocks[i], key)]  
 for j in range(d):  
 ans += chr(tmp[j])  
 print(ans)  
  
# Block을string으로 바꿔주는function  
def changeTostr(block):  
 ans = ""  
 tmp\_fun = [k+ord('A') for k in block]  
 for j in range(len(tmp\_fun)):  
 ans+=chr(tmp\_fun[j])  
 return(ans)  
  
cnt=0  
# inverse key matrix의 랜덤 초기화  
Key\_mat = np.array([[10, 10, 12, 24, 13],  
 [9, 22, 16, 6, 1],  
 [0, 0, 17, 22, 2],  
 [8, 21, 10, 25, 10],  
 [17, 4, 9, 1, 16]])  
  
# Text의scoring function  
def IML(col):  
 return sum([mono\_normalized[i]\*np.log2(mono[i]) for i in col]) \* -1  
  
# 모든X\_array에 대해scoring function을 계산해 가장 높은column들을Key\_mat에 넣어줌  
for element in X\_array:  
 tmp = []  
 for to\_intblock in int\_blocks:  
 tmp.append(np.dot(to\_intblock, element)%26)  
 iml = IML(tmp)  
 for j in range(0,d):  
 if I\_set[j] < iml:  
 Key\_mat[:,j] = element  
 I\_set[j] = iml  
 print(Key\_mat)  
 print(cnt)  
 print(I\_set)  
 break  
 cnt+=1  
  
print(Key\_mat)  
# 5!개의column permutation 생성 및decrypt  
perms = np.array(list(itertools.permutations(Key\_mat.T)))  
for ky in perms:  
 ky = ky.T  
 decoder(ky)