2020년도 2학기 정보보호

프로젝트 1 보고서

<Cipher Text Only Attack of Hill Cipher>

담당 교수: 허준범 교수님

제출일: 2020.10.10

제출인: 컴퓨터학과 2017320233 김채령

[목차]

1. 과제 정의

- Hill Cipher로 암호화된 ciphertext를 해독하여 plaintext 알아내기

- Hill Cipher 정의 및 각 문자 정의

  • Hill Cipher 요약

  문자열을 암호화하기 위해 d*d 차원의 key matrix를 사용한다.

  • Hill Cipher 암호화

$$C = [c_1\, c_2\, \cdots\, c_d] = [p_1\, p_2\, \cdots\, p_d] \begin{bmatrix} k_{11} k_{12} \cdots k_{1d} \\ k_{21} k_{22} \cdots k_{2d} \\ \vdots \quad \vdots \quad \ddots \\ k_{d1} k_{d2} \cdots k_{dd} \end{bmatrix} mod\ 26$$

  • Hill Cipher 복호화

$$P = [p_1\, p_2\, \cdots\, p_d] = [c_1\, c_2\, \cdots\, c_d] \begin{bmatrix} k_{11} k_{12} \cdots k_{1d} \\ k_{21} k_{22} \cdots k_{2d} \\ \vdots \quad \vdots \quad \ddots \\ k_{d1} k_{d2} \cdots k_{dd} \end{bmatrix}^{-1} mod\ 26$$

  • 각 문자에 할당된 정수

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

- 주어진 ciphertext (총 1285-byte)

HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVEIIYTKIOTQRXXQVSQL
ISVVOCNGCUXPKPIUBOHTVKCFKWNJSEZYSSUTUOESIXKAPVFXNZHAOQTLCGYJVAEHLNNKEESQMKSH
KKDFCNZSRHRDKHSDKFXVPTGMKRUPZBIKEVNYEKXMFXKFYMWYUDZDENEWNKDAOUXGPCXZDLCSNF
GCMCSNUAOJDBLQTAHEWYZCHQJYKSNUWOKQKONZGOKDXGUXKEMWQMCFGUEAVKHDIIATCHVTGYM
GKJMLNPCNAYKMIRWEETIYQKELEGLQOVKISFNUDAJQIQYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZG
DBIUDDHCUDFRFOVSZXADSHYSGLTQBMNEMKDCFSOZSRDYLIHIAXCMGMFEIDNZKOVJEOIEFNWWQEDR
LZYZIZXADSHYSGLJYFWDUAKSIOGOZOXWYPBUFEPNBIRJUJNDZJJYMURKNCIKPWLRMRIAGVSXTYNIWPR
OHLDHQOMBEKZURQCLQOVKISFNUAFQBHGPCLHZTPJVPXIZKLQSNVKIJAEITTNVSVWNFYVATDEMKDCT
GIHKZTVGZYXTYQEDBACFMNCAHRDKHSDKFXZXXGMJOSLPSZBMOILMMWRALAFFMNXXDYFBIYQVVOH
SWKGBIRJGTBYQLKIJAEQBTAXGFGAVUIJADHQKLFWRJXYFVIGGQZNBHSUIYOZALSKIABLWQNXNXKOAJAI
KHXODXWORVDOGBMHOPLQJZALQJZALIKTKLENZHQAVYUEUFEVLUXHGOWNMGWXUIAHGQOMNCKFQLI
PBNKVWDLNGMJCOBFKIGBYWPAHMMPQLUTOGECXITZVVAJEOIDCNWMFNLOBGQXCYFWQFWVXWRKWY
GBFHJVLBAWBOUQEKHZHSZZIZARYITDCLQFPGBTJMQVSQLIHPEJONCYMZWTVJVZOBOMOHPSXMPUKVA
GXIPOQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBEUKVNCAWANSNXISVVOWHQGQFBGWKQEGBIFRGIZUJQ
WIMFANTGBHWGVAGXIPOQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQ
FUEVPLLIODEYRYLUTOGPYAFHJFIVOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMGIMQ
DSYNWMFNIEITWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQICHNUUQFJYUCQUO

2. 과제 해설

A. Key size 예측

Key matrix는 d*d 차원의 정사각행렬이다. 이때, d개의 문자를 한 번에 암호화/복호화 할 수 있다. 주어진 암호문은 총 1285개의 문자로 이루어져 있다. 1285를 소인수 분해하면, 1285 = 1 * 5 * 257 * 1285이다. 해당 암호문을 별도의 padding 없이 d개의 문자만큼 한 번에 암호화/복호화하려면 1285의 약수가 d가 되어야 한다. 이때 구현한 알고리즘은 Cipher text only attack로 계산 복잡도가 O(d*(26^d))이기에 d가 10이상이면 일반적인 컴퓨터 성능으로 실질적인 공격이 불가능하다고 본다. 따라서 d = 5라고 가정하는 것이 유일한 실질적인 공격 성공 가능 경우이다.

B. 참고 논문 알고리즘 해설 및 관련 여러 시도

i. 알고리즘 pseudo-code

코드는 참고문헌의 알고리즘의 방법론을 기반으로 구현했는데, 해당 논문에 제시된 pseudo-code는 다음과 같다.

Algorithm 1 Divide-and-conquer attack on Hill in $O(d26^d)$.

Require: A ciphertext C of length $n = md$
Ensure: A representative decryption matrix K^{-1}
Divide C into m blocks $C_1, \cdots, C_m$
for $t = 0$ **to** $d - 1$ **do**
  for $i = 1$ **to** m **do**
    Set $d_{i,t}$ to the modulo 26 sum of the first $t + 1$ elements of C_i
Set K^{-1} to an arbitrary $d \times d$ matrix
Set I to an all $-\infty$ vector of length d
{the j'th element of I corresponds to the IML of the j'th column of K^{-1}}
for $i = 1$ **to** m **do**
  Set $p_i = 0$
  {p_i is a decrypted letter of C_i using an all-zero guess for a column of K^{-1}}
for all $d \times 1$ vectors x (except the all-zero vector) in lexicographical order **do**
  Let t denote the maximum number of zeros at the top of x.
  for $i = 1$ **to** m **do**
    $p_i = p_i + d_{i,t} \mod 26$
  $iml_x = \text{IML}(p_1, \cdots, p_m)$
  Let y be a column of K^{-1} with the smallest index $iml_y = \min I$
  if $iml_y < iml_x$ **and** x is not all-zero modulo 2 or 13 **then**
    Replace the column y in K^{-1} with x
    Replace iml_y in I with iml_x

위의 pseudo-code는 d*d의 inverse Key matrix의 각 열을 이용하여 암호문을 해독하여도 영어 문자의 monogram frequency의 성질을 유지한다는 사실에 기반한 알고리즘이다. 해당 pseudo-code를 작은 단위로 나누어 해석하면 다음과 같

다.

1: Divide C into m blocks $C_1, \cdots, C_m$

우선 key size가 d*d이므로, 총 암호문의 길이 n = m*d로 재정의 수 있다. 해당 정의에 부합하도록 암호문을 m*d의 차원의 배열로 저장한다.

2: **for** $t = 0$ **to** $d - 1$ **do**
3: **for** $i = 1$ **to** m **do**
4: Set $d_{i,t}$ to the modulo 26 sum of the first $t + 1$ elements of C_i

Inverse Key matrix의 모든 가능한 열들은 사전 순으로 정렬할 수 있다. 즉, 각 열들은 [0 0 0 0 0]부터 [25 25 25 25 25]까지의 범위 안에 존재한다. 이 순서대로 가능한 열(x)을 추측하면, P_i = C*x의 계산 시간을 단축할 수 있다. P_i의 각 문자 빈도수와 주어진 영어 문자 빈도수를 비교함으로써 x가 실제 inverse Key Matrix의 열이 맞을 가능성을 계산할 수 있다. 열 x는 26^5의 경우의 수를 가지는데 매 경우마다 행렬 곱셈을 하는 것은 시간 복잡도를 크게 증가시킨다. 따라서, 위 알고리즘에서는 사전 순으로 x를 추측하여, x와 그 이전에 추측되었던 열 x′의 차이를 미리 계산하여 시간 복잡도를 줄인다. 즉, P_i = C*x mod 26은 P_i=P_i′+d[i][t] mod 26과 동일한데, 여기서 P_i′는 사전 순으로 x 이전의 추측된 열이었던 x′로 계산한 평문이다. 그리고 행렬 d에는 x′와 x의 차이가 저장되어 있다. t는 두 열의 차를 계산하였을 때, 벡터 위쪽의 0의 개수로 x′와 x의 동일한 부분의 크기를 나타낸다.

5: Set K^{-1} to an arbitrary $d \times d$ matrix
6: Set I to an all $-\infty$ vector of length d
{the j'th element of I corresponds to the IML of the j'th column of K^{-1}}

먼저, inverse Key matrix에 0~25 사이의 임의의 정수를 할당하여 초기화한다. 이후 반복문에서 각 열을 추측하면서 초기화 당시 할당된 임의의 열은 대체가 될 것이기에 어느 값이든 상관은 없다. 새롭게 추측된 열이 기존의 inverse Key matrix의 열을 대체할 수 있을지에 대한 판단 기준이 IML이다. IML은 index of maximum likelihood로, 추측된 열로 계산한 평문의 문자 빈도수가 일반적으로 알려진 문자 빈도수와 얼마나 일치하는지에 대한 지표이다. 즉, IML 값이 작을수록 알려진 문자 빈도수와 거리가 멀다는 뜻이므로 추측한 열이 inverse Key matrix의 열이 아닐 가능성이 높음을 의미한다. 배열 I는 inverse Key matrix의 각 열의 IML 값을 저장하는데, 초기화할 때 임의로 할당한 열은 어차피 대체될 것이기 때문에 가장 작은 값인 -infinity로 초기화한다.

7: **for** $i = 1$ **to** m **do**
8: Set $p_i = 0$
{p_i is a decrypted letter of C_i using an all-zero guess for a column of K^{-1}}

P_i는 추측한 x로 계산한 길이가 257(m)인 평문의 한 단위를 의미한다. 현재 반복문으로 x를 추측하기 이전이므로, P_i를 0으로 초기화한다.

9: **for** all $d \times 1$ vectors x (except the all-zero vector) in lexicographical order **do**
10: Let t denote the maximum number of zeros at the top of x.
11: **for** $i = 1$ **to** m **do**
12: $p_i = p_i + d_{i,t} \mod 26$
13: $iml_x = \text{IML}(p_1, \cdots, p_m)$
14: Let y be a column of K^{-1} with the smallest index $iml_y = \min I$
15: **if** $iml_y < iml_x$ **and** x is not all-zero modulo 2 or 13 **then**
16: Replace the column y in K^{-1} with x
17: Replace iml_y in I with iml_x

9번 줄부터 17번 줄까지는 사전 순으로 x를 추측하면서 각 x가 기존의 inverse Key matrix의 열을 대체하는 부분이다. 각 x에 대하여 벡터 윗부분의 0의 개수를 t로 하면, 위에서 언급한 바와 같이 P_i의 계산을 빠르게 할 수 있다. 이후 P_i의 IML(iml_x)을 계산하고, 이를 기존의 inverse Key matrix에서 가장 작은 IML을 가진 열(y)의 IML(iml_y)과 비교한다. 비교 결과, iml_x가 iml_y보다 크면 x를 y 열과 교체하고, y열에 해당하는 iml 값도 x 열의 iml로 바꾸어 준다. 그 이유는, iml_x가 iml_y보다 크다는 것이 기존의 inverse Key matrix에서 가장 빈도수에 부합하지 않는 열보다 x를 적용하면 빈도수에 조금 더 부합하기 때문이다. Key size = 5*5로 가정했기 때문에, 26^5만큼의 반복문을 완료하면 모든 열의 경우의 수 중에서 가장 알려진 문자 빈도수에 가까운 inverse Key matrix의 열들을 얻을 수 있다.

이렇게 알려진 monogram frequency를 활용하여 얻은 5개의 열들 간의 순서를 정해주어야 최종적으로 inverse Key matrix를 얻을 수 있는데, 이는 bigram frequency를 이용하거나 5!=120가지의 inverse Key matrix 경우를 모두 이용하여 복호화한 후 읽히는 문장을 평문으로 채택하면 된다.

ii. Pseudo-code 구현과 여러 시도

0. 구현 개괄

코드 구현은 기본적으로 위의 논문에서 소개한 알고리즘에 바탕을 둔다. 다만, 의문이 드는 부분은 여러 가지 경우로 구현해보았다. 우선, 5가지 후보 열을 찾는 것을 목표로 했다.

1. 세부 사항의 여러 시도

A. 열 x modulo 2 or 13이 모두 0인 열이 되지 않도록 하는 조건

Pseudo-code의 15번째 줄의 조건에 의문점이 들었다. 엔트로피 조건에 따라 해당 조건문이 필요하다고 기술되어 있지만, 위 조건문을 해석하

는 경우의 수는 2가지이기 때문에 2 방법 모두 구현해 보았다.

- 방법 1. 열 x의 모든 원소가 2의 배수이거나, 모든 원소가 13의 배수

e.g. [2 2 2 2 2] -> X, [13 13 13 13 13] -> X, [2 13 13 2 13] -> O

```
for (int i = 0; i < d; i++) {
    if (x[i] % 2 == 0) {
        count_mod_zero_2++;
    }
}
if (count_mod_zero_2 == d) { condition = 0; }

for (int i = 0; i < d; i++) {
    if (x[i] % 13 == 0) {
        count_mod_zero_13++;
    }
}
if (count_mod_zero_13 == d) { condition = 0; }
```

- 방법 2. 열 x의 모든 원소가 2의 배수 또는 13의 배수

e.g. [2 2 2 2 2] -> X, [13 13 13 13 13] -> X, [2 13 13 2 13] -> X

```
for (int r = 0; r < d; r++) {
    if (((x[r] % 2) == 0) || ((x[r] % 13) == 0)) {
        count_temp++;
    }
}
if (count_temp == d) { condition = 0; }
```

B. x와 x'의 차이를 활용하여 빠르게 P_i를 계산하기 위해서는 행렬 d[m][t]를 사용할 수 있지만, 좀 더 직관적인 구현을 위해서는 매 x마다 P_i를 구할 수 있다. P_i = C*x가 그 공식이다.

- 방법 1. 행렬 d를 사용하는 경우

```
int D[257][5];

for (int t = 0; t < d; t++) {
    for (int i = 0; i < m; i++) {

        int mod_sum = 0;
        for (int e = 0; e < t + 1; e++) {
            mod_sum += (C[i][e]);
            mod_sum = mod_sum % 26;
        }
        mod_sum = mod_sum % 26;
        D[i][t] = mod_sum;

    }
}
```

```
// 11-12. update vector P values
for (int a = 0; a < m; a++) {
    P[a] = (P[a] + D[a][t]) % 26;
}
```

- 방법 2. P_i를 매번 계산하는 경우

```
for (int a = 0; a < m; a++) {
    int pc_temp = 0;
    for (int b = 0; b < d; b++) {
        pc_temp += (C[a][b]) * x[b];
        pc_temp = pc_temp % 26;
    }
    P[a] = pc_temp;
}
```

C. Accept 이전 논문의 pseudo-code 구현

해당 논문의 승인 이전 버전에서는 IML을 함수로 구현하지 않고 반복문 내에서 계산한다.

- 승인된 논문의 IML 계산

13: $iml_x = \mathrm{IML}(p_1, \cdots, p_m)$

이에 별도로 직접 IML 함수를 아래와 같이 구현하였다.

```
double IML(int str[], int m) {

    // compute frequency f_hat
    double f_hat[26];
    for (int i = 0; i < 26; i++) {
        int cnt = 0;
        for (int j = 0; j < m; j++) {
            if (str[j] == i) {
                cnt++;
            }
        }
        f_hat[i] = cnt;
    }
    // normalising
    for (int i = 0; i < 26; i++) {
        f_hat[i] = (double)(f_hat[i] / m);
    }


    double f[26] = { 8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2, 6.1, 7,
                     0.2, 0.8, 0.4, 2.4, 6.7, 1.5, 1.9, 0.1, 6,
                     6.3, 9.1, 2.8, 1, 2.4, 0.2, 2, 0.1 };
    // normalising
    for (int i = 0; i < 26; i++) {
        f[i] = (double)(f[i] / 90.7);
    }


    double iml_value = 0;
    for (int i = 0; i < 26; i++) {
        iml_value = -(iml_value + (f_hat[i] * log2(f[i]))); // sum of f_hat*log2(f)
    }

    return iml_value;
}
```

알려진 문자의 빈도수인 f의 합이 100이 아니라 90.7인 것을 발견하여 각 빈도수를 90.7로 나누어 정규화했다. 추측건데, 100이 나오지 않은 것은 문장 부호와 같은 특수 문자의 빈도수를 제외했기 때문인 것 같다.

- 승인 이전의 논문의 x의 iml 계산

12: **for** $i = 1$ **to** m **do**
13: $iml = iml - \frac{1}{m}\log_2 f_{p_i}$
14: $p_i = p_i + d_{i,t} \mod 26$
15: $iml = iml + \frac{1}{m}\log_2 f_{p_i}$
16: **if** x is not all-zero modulo 2 or 13 **then**

```
for (int a = 0; a < m; a++) {
    iml = iml - (1 / (m)) * log2(f[P[a]]);
    P[a] = (P[a] + D[a][t]) % 26;
    iml = iml + (1 / (m)) * log2(f[P[a]]);
}
```

C. 다른 시도

i. 완전 Brute-Force Attack

모든 key의 경우의 수를 다 시도해보는 완전 Brute-Force Attack의 경우, 현재까지 알려진 계산 복잡도는 O((d^3)*(26^(d^2)))이다. 따라서, d=5일 때 그 값이 일반적인 PC로는 실용적인 시간 내에 공격이 불가하다. 이에 간단히 pseudo-code만 작성해 보았다.

1: divide Ciphertext into m blocks C_1, … , C_m

// Ciphertext length n = m*d, d=5, m=257

2: for all possible inverse of key matrix: K_inverse_j (1<=j<=26^25)

// from {[0 0 0 0 0], [0 0 0 0 0], [0 0 0 0 0], [0 0 0 0 0], [0 0 0 0 0]}

// to {[25 25 25 25 25], [25 25 25 25 25], [25 25 25 25 25], [25 25 25 25 25], [25 25 25 25 25]}

3: for i=1 to 257:

4: P_hat_j_i = (C_i * K_inverse) mod 26

5: for all possible guessed plaintexts: P_hat_ j (1<=j<=26^25)

6: Plaintext = P_hat_j which is recognizable by human

ii. IML 함수를 정의하지 않고, Euclidean similarity로 monogram 분포 비교

IML 함수의 기본 목적은 추측한 inverse Key matrix의 열 x로부터 계산한 평문 단위가 알려진 문자 빈도수에 얼마나 부합하는지를 계산하기 위함이다. 따라서,

논문에 제시된 IML 공식 대신에, 두 빈도 분포(평문의 빈도 분포, 알려진 빈도 분포)의 유사도를 Euclidean similarity로도 구현해 보았다.

```
for (int i = 0; i < 26; i++) {
    similarity += (f[i] - f_hat[i]) * (f[i] - f_hat[i]);
}
```

D. 결론

Inverse key matrix는 다음과 같다.

```
---------- K_inverse ----------
3  17  12  9  18
19  24  18  7  12
11  13  4  12  6
2  11  9  20  16
3  21  0  13  23
```

평문은 다음과 같다.

```
---------- PLAIN TEXT ----------
CRYPTANALYSISISTHESTUDYOFANALYXINGINFORMATIONSYSTEMSINTHESTUDYOFANALYZINGINFORMATIONSYSTEMSINORDERTOSTUDYTHEHIDDENASPECTSOFTHESYSTEMSCRYPTANALYSISISUSEDTOBREA
CHCRYPTOGRAPHICSECURITYSYSTEMSANDGAINACCESSTOTHECONTENTSOFENCRYPTEDMESSAGESEVENIFTHECRYPTOGRAPHICKEYISUNKNOWNINADDITIONTOMATHEMATICALANALYSISOFCRYPTOGRAPHICAL
GORITHMSCRYPTANALYSISINCLUDESTHESTUDYOFSIDECHANNELATTACKSTHATDONOTTARGETWEAKNESSESINTHECRYPTOGRAPHICALGORITHMSTHEMSELVESBUTINSTEADEXPLOITWEAKNESSESINTHEIRWEAK
IMPLEMENTATIONEVENTHOUGHTHEGOALHASBEENTHESAMETHEMETHODSANDTECHNIQUESOFCRYPTANALYSISHAVECHANGEDDRASTICALLYTHROUGHTHEHISTORYOFCRYPTOGRAPHYADAPTINGTOINCREASINGCR
YPTOGRAPHICCOMPLEXITYRANGINGFROMTHEPENANDPAPERMETHODSOFTHEPASTTHROUGHMACHINESLIKETHEBRITISHBOMBESANDBOLOSSUSCOMPUTERSATBLETCHLEYPARKINWORLDWARTWOTOTHEMATHEMAT
ICALLYADVANCEDCOMPUTERIZEDSCHEMESOFTHEPRESENTMETHODSFORBREAKINGMODERNCRYPTOSYSTEMSOFTENINVOLVESOLVINGCAREFULLYCONSTRUCTEDPROBLEMSINPUREMATHEMATICSTHEBESTKNOWN
BEINGINTEGERFACTORIZATIONGIVENSOMEENCRYPTEDDATATHEGOALOFTHECRYPTANALYSTISTOGAINASMUCHINFORMATIONASPOSSIBLEABOUTTHEORIGINALUNENCRYPTEDDATAISUSEFULTOCONSIDERTW
OASPECTSOFACHIEVINGTHISTHEFIRSTISBREAKINGTHESYSTEMTHATISDISCOVERINGHOWTHEENCIPHERMENTPROCESSWORKSTHESECONDISSOLVINGTHEKEYTHATISUNIQUEFORAPARTICULARENCRYPTEDME
SSAGEORGROUPOFMESSAGE
```

논문에서 얻은 정보는 5*5 size의 Hill Cipher 암호화일지라도, monogram frequency에 여전히 부합한다는 것이다. 이에 다양한 해석으로 논문 알고리즘을 구현해보았다. 그러나 올바른 inverse Key matrix를 찾는 것에는 실패했다.

그래서 26^5개의 가능한 column으로 복호화를 해본 후 문자 빈도를 사용해 column 후보들을 출력해보았다. 이 중 5가지의 순서를 brute force 방법으로 배열하여 평문을 출력해보았다. 며칠의 결과, 인식 가능한 평문을 찾을 수 있었고 해당 평문을 기반으로 inverse Key matrix를 P=C*inverseK mod 26 연산을 통해 구할 수 있었다.

- 빈도 조건의 설정

빈도 조건은 trial&error(시행착오)의 방법으로 실험해가며 설정했다. 과제 설명에 주어진 대로, 가장 빈도가 높은 e와 t, 그리고 가장 빈도가 낮은 q와 z, 4가지 문자에 대해서 빈도수 조건을 걸었다. 후보 열의 개수가 n개이면 n!의 경우의 수 조합으로 inverse Key matrix를 구성하여 각 평문을 구했다. 이때, python library로 간단히 순열을 구현하였다. 읽히는 평문이 없는 경우, 빈도수 조건을 완화해가며 각 출력된 column들의 순열 조합으로 인식 가능한 평문을 찾았다. 수많은 시도 끝에, inverse

key matrix의 column들을 찾았고 경우의 수에 대한 연산을 통해 평문을 얻을 수 있었다.

빈도수 조건을 조절하는 것은 거의 Brute-force에 가까웠지만, 논문 상 구현으로 답을 도출할 수 없었기에 과감히 시간을 투자하여 경우의 수를 찾아보았다.

아래는 빈도수 초기 조건이다.

E: ((12.7/90.7)*257) = 35보다 많은 경우

T: ((9.1/90.7)*257) = 25 보다 많은 경우

Q: ((0.1/90.7)*257) = 0 보다 적은 경우

Z: ((0.1/90.7)*257) = 0 보다 적은 경우

이때, 참고한 빈도 정보는 과제 설명 자료에 제시되어 있었다. 하지만 A~Z까지 빈도수 합이 100%가 나오지 않아, 이후에는 인터넷 검색을 통하여 정규화된 즉, A~Z까지의 빈도 합이 100%인 빈도 정보를 참고하여 조정했다. 해당 정보에서 e와 t의 빈도를 비슷하게 10%정도로 두었고, 이를 기준으로 삼아 다시 세부 조정하였다.

아래는 빈도수 최종 조건이다.

E: 18 보다 많은 경우

T: 17 보다 많은 경우

Q: 2 보다 적은 경우

Z: 2 보다 적은 경우

위의 빈도 조건으로는 총 7가지 후보 열들을 얻었다. 이 중 5가지를 고르고 5!의 경우의 수로 올바른 inverse Key matrix를 발견하여 plain text를 읽을 수 있었다. 열 간의 순서를 찾는 데에는, 5가지 inverse Key matrix와 Cipher text를 입력으로 주어 5!의 경우의 수에 대해 평문으로 해독하였다.

참고 문헌

Khazaei, S., Ahmadi, S., 2017. Ciphertext-only attack on d × d Hill in O(d13d). Information Processing Letters 118, 25–29. doi:10.1016/j.ipl.2016.09.006

<부록>

논문 구현 Source code와 주석

```
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>

double IML(int[]);

int main() {

	// divide and conquer attack on Hill in O(d*26^d)
		// input: ciphertext C of length n = m*d, key size = d*d
		// output: representative decryption matrix K_inverse

	// 1. give inputs
	char Cipher[] =
"HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVEIIYTKIOTQRXXQVSQLISVVOCNGCUXPKPIUBOHTVKCFKWNJSEZYSSUTUOESIXKAPVFXNZHA
OQTLCGYJVAEHLNNKEESQMKSHKKDFCNZSRHRDKHSDKFXVPTGMKRUPZBIKEVNYEKXMFXKFYMWYUDZDENEWNKDAOUXGPCXZDLCSNFGCMCSNUAOJDBLQTAHEWYZCHQJYKSNU
WOKQKONZGOKDXGUXKEMWQMCFGUEAVKHDIIATCHVTGYMGKJMLNPCNAYKMIRWEETIYQKELEGLQOVKISFNUDAJQIQYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZGDBIUDDHCU
DFRFOVSZXADSHYSGLTQBMNEMKDCFSOZSRDYLIHIAXCMGMFEIDNZKOVJEOIEFNWWQEDRLZYZIZXADSHYSGLJYFWDUAKSIOGOZOXWYPBUFEPNBIRJUJNDZJJYMURKNCIKP
WLRMRIAGVSXTYNIWPROHLDHQOMBEKZURQCLQOVKISFNUAFQBHGPCLHZTPJVPXIZKLQSNVKIJAEITTNVSVWNFYVATDEMKDCTGIHKZTVGZYXTYQEDBACFMNCAHRDKHSDKF
XZXXGMJOSLPSZBMOILMMWRALAFFMNXXDYFBIYQVVOHSWKGBIRJGTBYQLKIJAEQBTAXGFGAVUIJADHQKLFWRJXYFVIGGQZNBHSUIYOZALSKIABLWQNXNXKOAJAIKHXODX
WORVDOGBMHOPLQJZALQJZALIKTKLENZHQAVYUEUFEVLUXHGOWNMGWXUIAHGQOMNCKFQLIPBNKVWDLNGMJCOBFKIGBYWPAHMMPQLUTOGECXITZVVAJEOIDCNWMFNLOBGQ
XCYFWQFWVXWRKWYGBFHJVLBAWBOUQEKHZHSZZIZARYITDCLQFPGBTJMQVSQLIHPEJONCYMZWTVJVZOBOMOHPSXMPUKVAGXIPOQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBE
UKVNCAWANSNXISVVOWHQGQFBGWKQEGBIFRGIZUJQWIMFANTGBHWGVAGXIPOQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQFUEVPLLIODEYRYL
UTOGPYAFHJFIVOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMGIMQDSYNWMFNIEITWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQICHNUUQFJ
YUCQUO"; //1285bytes
	//int m = 257;
	//int d = 5;
	int C[257][5] = { 0, }; // m=257, d=5

	int tmp = 0;
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			C[i][j] = Cipher[tmp] - 65;
			tmp++;
		}
	}

	// check the input clpher matrix C
	printf("Wn Matrix C Wn");
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			printf("%d  ", C[i][j]);
		}
		printf("Wn");
	}
	printf("Wn");

	printf("Wn Ciher text Wn");
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			printf("%c", (char)(C[i][j] + 65));
		}
	}
	printf("Wn");

	// 2-4. set d[i][t] values
	/*
	int D[257][5];

	for (int t = 0; t < 257; t++) {
		for (int i = 0; i < 5; i++) {

			int mod_sum = 0;
			for (int e = 0; e < t + 1; e++) {
				mod_sum += (C[i][e]);
				mod_sum = mod_sum % 26;
			}
			mod_sum = mod_sum % 26;
```

```
                                D[i][t] = mod_sum;

                        }
                }

                // check the d[i][t] values of matrix D
                printf("₩n Matrix D ₩n");
                for (int i = 0; i < 257; i++) {
                        for (int j = 0; j < 5; j++) {
                                printf("%d  ", D[i][j]);
                        }
                        printf("₩n");
                }
                */

                // 5. set the matrix K_inverse
                int K_inverse[5][5]; // # m = 257, d = 5
                srand(time(NULL));
                for (int i = 0; i <5; i++) {
                        for (int j = 0; j < 5; j++) {
                                K_inverse[i][j] = rand() % 26; // random number in the range 0~25
                        }
                }

                // check the K_inverse
                printf("₩n Initial K_inverse ₩n");
                for (int i = 0; i < 5; i++) {
                        for (int j = 0; j < 5; j++) {
                                printf("%d  ", K_inverse[i][j]);
                        }
                }

                // 6. set the vector I : I[j] == IML of the jth column of K_inverse
                double I[5] = { 0, };
                for (int i = 0; i < 5; i++) {
                        I[i] = -INFINITY; // so that it can be replaced by x below
                }

                // 7-8. set the vector P : P[i] == decrypted letter of C[i] using an all zero guess for a column of K_inverse
                int P[257] = { 0, };

                // 9-17. find out the columns of K_inverse
                int forloopcnt = 0;
                for (int i_0 = 0; i_0 < 26; i_0++) {
                        clock_t start = clock();
                        for (int i_1 = 0; i_1 < 26; i_1++) {
                                for (int i_2 = 0; i_2 < 26; i_2++) {
                                        for (int i_3 = 0; i_3 < 26; i_3++) {
                                                for (int i_4 = 0; i_4 < 26; i_4++) {

                                                        forloopcnt++;
                                                        //printf("<# %d> ₩n", forloopcnt);

                                                        if (i_0== 0 && i_1 == 0 && i_2 == 0 && i_3 == 0 && i_4 == 0)
continue;

                                                        int x[5] = { i_0, i_1, i_2, i_3, i_4 }; // # d = 5

                                                        // 10. counting max number of zeros at the top of x
                                                        /*
                                                        int t = 0;
                                                        if (i_0 == 0) {
                                                                t++;
                                                                if (i_1 == 0) {
                                                                        t++;
                                                                        if (i_2 == 0) {
                                                                                t++;
                                                                                if (i_0 == 0) {
                                                                                        t++;
                                                                                }
                                                                        }
                                                                }
                                                        }
```

```
iml_y = min I
```

```
// 11-12. update vector P values
for (int a = 0; a < 257; a++) {
		P[a] = (P[a] + D[a][t]) % 26;
}
*/
for (int a = 0; a < 257; a++) {
		int pc_temp = 0;
		for (int b = 0; b < 5; b++) {
			pc_temp += (C[a][b]) * x[b];
			pc_temp = pc_temp % 26;
		}
		P[a] = pc_temp;
}
/*
for (int a = 0; a < 257; a++) {
		printf(" %d ", P[a]);
}
printf("WnWn");
*/


// 13. set an iml_x value
double iml_x = IML(P);
//printf("<# %d IML> %lfWn", forloopcnt, iml_x);

// 14. set y : column of K_inverse with the minimum index

int y[5];
int min_index = 0;
for (int r = 0; r < 5; r++) {
		if (I[r] < I[min_index]) {
			min_index = r;
		}
}
for (int s = 0; s < 5; s++) {
		y[s] = K_inverse[s][min_index];
}

// 15-17. update the K_inverse columns
		// x not all-zero modulo 2 or 13 conditions
int condition = 1;
int count_mod_zero_2 = 0;
int count_mod_zero_13 = 0;
int count_temp = 0;
// QQQQQQ. [2 13] 경우???
/*
for (int i = 0; i < d; i++) {
		if (x[i] % 2 == 0) {
			count_mod_zero_2++;
		}
}
if (count_mod_zero_2 == d) { condition = 0; }

for (int i = 0; i < d; i++) {
		if (x[i] % 13 == 0) {
			count_mod_zero_13++;
		}
}
if (count_mod_zero_13 == d) { condition = 0; }
*/

for (int r = 0; r < 5; r++) {
		if (((x[r] % 2) == 0) || ((x[r] % 13) == 0)) {
			count_temp++;
		}
}
if (count_temp == 5) { condition = 0; }


if ((I[min_index] < iml_x) && (condition == 1)) {
		// replace the column y in K_inverse with x
		for (int w = 0; w < 5; w++) {
			K_inverse[w][min_index] = x[w];
		}
		// replace iml_y in I with iml_x
```

```
                                                                        l[min_index] = iml_x;

                                                                        printf(" Wn updating K_inverse Wn");
                                                                        printf("update <IML> %lfWn", iml_x);
                                                                        for (int g = 0; g < 5; g++) {
                                                                                  for (int f = 0; f < 5; f++) {
                                                                                            printf(" %d  ",
K_inverse[g][f]);
                                                                                  }
                                                                        }

                                                              }
                                                    }
                                          }
                                }

                      }
                      clock_t end = clock();
                      printf("Time: %lfWn", (double)(end - start) / CLOCKS_PER_SEC);
            }

            printf(" WnWn forloop 개수: %d Wn", forloopcnt);

            // print K_inverse
            printf(" Wn ---------- K_inverse ---------- Wn");
            for (int i = 0; i < 5; i++) {
                      for (int j = 0; j < 5; j++) {
                                printf("%d  ", K_inverse[i][j]);
                      }
                      printf("Wn");
            }

            // print Plaintext guessed
            printf(" Wn ---------- PLAIN TEXT ---------- Wn");
            // p = c*k_inverse
            int PLAIN[257][5]; // # m = 11, d = 2
            for (int i = 0; i < 257; i++) {
                      for (int j = 0; j < 5; j++) {
                                PLAIN[i][j] = 0;
                                for (int k = 0; k < 5; k++) {
                                          PLAIN[i][j] += (C[i][k]) * K_inverse[k][j];
                                }
                                PLAIN[i][j] = (PLAIN[i][j] % 26);
                      }
            }


            for (int i = 0; i < 257; i++) {
                      for (int j = 0; j < 5; j++) {
                                printf("%c", (char)(PLAIN[i][j] + 65));
                      }
            }

            printf(" WnWn");

            return 0;

}

double IML(int str[]) {

            // compute normalised frequency f_hat QQQQQ.
            int f_hat[26];
            for (int i = 0; i < 26; i++) {
                      int cnt = 0;
                      for (int j = 0; j < 257; j++) {
                                if (str[j] == i) {
                                          cnt++;
                                }
                      }
                      f_hat[i] = cnt;
            }
            //printf("Wn f_hat COUNT = %dWn", f_hat[4]);

            // check whether the sum==1
```

```
		/*
		double sum_hat = 0;
		for (int i = 0; i < 26; i++) {
			sum_hat += f_hat[i];
		}
		printf("Wn f_hat sum = %lf Wn", sum_hat);
		*/

		double f[26] = { 8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2, 6.1, 7,
					0.2, 0.8, 0.4, 2.4, 6.7, 1.5, 1.9, 0.1, 6,
					6.3, 9.1, 2.8, 1, 2.4, 0.2, 2, 0.1 };
		// normalising
		for (int i = 0; i < 26; i++) {
			f[i] = (double)(f[i] / 90.7);
		}

		// check whether the sum==1
		/*
		double sum_f = 0;
		for (int i = 0; i < 26; i++) {
			sum_f += f[i];
		}
		printf("Wn f sum = %lf Wn", sum_f);
		*/

		double iml_value = 0;


		for (int i = 0; i < 26; i++) {
			iml_value = -(iml_value + ((double)(f_hat[i]/257.0) * log2(f[i]))); // sum of f_hat*log2(f)
		}

		/*
		for (int i = 0; i < m; i++) {
			iml_value = iml_value + (f_hat[str[i]] * log2(f[str[i]])); // sum of f_hat*log2(f)
		}
		*/

		//iml_value = -iml_value;

		return iml_value;
}
```

빈도수 기반 Brute-force attack

- 1. 빈도 수 기반 후보 열 찾기

```
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>

int main() {

		// divide and conquer attack on Hill in O(d*26^d)
			// input: ciphertext C of length n = m*d, key size = d*d
			// output: representative decryption matrix K_inverse

		// 1. give inputs
		char Cipher[] =
"HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVE!IYTK!OTQRXXQVSQL!SVVOCNGCUXPKP!UBOHTVKCFKWNJSEZYSSUTUOES!XKAPVFXNZHA
OQTLCGYJVAEHLNNKEESQMKSHKKDFCNZSRHRDKHSDKFXVPTGMKRUPZB!KEVNYEKXMFXKFYMWYUDZDENEWNKDAOUXGPCXZDLCSNFGCMCSNUAOJDBLQTAHEWYZCHQJYKSNU
WOKQKONZGOKDXGUXKEMWQMCFGUEAVKHD!!ATCHVTGYMGKJMLNPCNAYKM!RWEET!YQKELEGLQOVK!SFNUDAJQ!QYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZGDB!UDDHCU
DFRFOVSZXADSHYSGLTQBMNEMKDCFSOZSRDYL!H!AXCMGMFE!DNZKOVJEO!EFNWWQEDRLZYZ!ZXADSHYSGLJYFWDUAKS!OGOZOXWYPBUFEPNB!RJUJNDZJJYMURKNC!KP
WLRMR!AGVSXTYN!WPROHLDHQOMBEKZURQCLQOVK!SFNUAFQBHGPCLHZTPJVPX!ZKLQSNVK!JAE!TTNVSVWNFYVATDEMKDCTG!HKZTVGZYXTYQEDBACFMNCAHRDKHSDKF
XZXXGMJOSLPSZBMO!LMMWRALAFFMNXXDYFB!YQVVOHSWKGB!RJGTBYQLK!JAEQBTAXGFGAVU!JADHQKLFWRJXYFV!GGQZNBHSU!YOZALSK!ABLWQNXNXKOAJA!KHXODX
WORVDOGBMHOPLQJZALQJZAL!KTKLENZHQAVYUEUFEVLUXHGOWNMGWXU!AHGQOMNCKFQL!PBNKVWDLNGMJCOBFK!GBYWPAHMMPQLUTOGECX!TZVVAJEO!DCNWMFNLOBGQ
XCYFWQFWVXWRKWYGBFHJVLBAWBOUQEKHZHSZZ!ZARY!TDCLQFPGBTJMQVSQL!HPEJONCYMZWTVJVZOBOMOHPSXMPUKVAGX!POQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBE
UKVNCAWANSNX!SVVOWHQGQFBGWKQEGB!FRG!ZUJQW!MFANTGBHWGVAGX!POQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQFUEVPLL!ODEYRYL
UTOGPYAFHJF!VOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMG!MQDSYNWMFN!E!TWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQ!CHNUUQFJ
YUCQUO"; //1285bytes
		//int m = 257;
		//int d = 5;
		int C[257][5] = { 0, }; // m=257, d=5
```

```
		int tmp = 0;
		for (int i = 0; i < 257; i++) {
			for (int j = 0; j < 5; j++) {
				C[i][j] = Cipher[tmp] - 65;
				tmp++;
			}
		}

		// check the input clpher matrix C
		printf("₩n Matrix C ₩n");
		for (int i = 0; i < 257; i++) {
			for (int j = 0; j < 5; j++) {
				printf("%d  ", C[i][j]);
			}
			printf("₩n");
		}
		printf("₩n");

		printf("₩n Ciher text ₩n");
		for (int i = 0; i < 257; i++) {
			for (int j = 0; j < 5; j++) {
				printf("%c", (char)(C[i][j] + 65));
			}
		}
		printf("₩n");

		// 7-8. set the vector P : P[i] == decrypted letter of C[i] using an all zero guess for a column of K_inverse
		int P[257] = { 0, };

		// 9-17. find out the columns of K_inverse
		int forloopcnt = 0;
		for (int i_0 = 0; i_0 < 26; i_0++) {
			clock_t start = clock();
			for (int i_1 = 0; i_1 < 26; i_1++) {
				for (int i_2 = 0; i_2 < 26; i_2++) {
					for (int i_3 = 0; i_3 < 26; i_3++) {
						for (int i_4 = 0; i_4 < 26; i_4++) {

forloopcnt++;
//printf("<# %d> ₩n", forloopcnt);

if (i_0 == 0 && i_1 == 0 && i_2 == 0 && i_3 == 0 && i_4 ==
0) continue;


int x[5] = { i_0, i_1, i_2, i_3, i_4 }; // # d = 5

for (int a = 0; a < 257; a++) {
	int pc_temp = 0;
	for (int b = 0; b < 5; b++) {
		pc_temp += (C[a][b]) * x[b];
		pc_temp = pc_temp % 26;
	}
	P[a] = pc_temp;
}

// print columns that meet the monogram frequency
int e_cnt = 0;
int t_cnt = 0;
int q_cnt = 0;
int z_cnt = 0;
int condition = 0;
for (int ptr = 0; ptr < 257; ptr++) {
	if (P[ptr] == 4) { //e
		e_cnt++;
	}
	if (P[ptr] == 19) { //t
		t_cnt++;
	}
	if (P[ptr] == 16) { //q
		q_cnt++;
	}
	if (P[ptr] == 25) { //z
		z_cnt++;
	}
```

```
                                                                }
                                                                if ((e_cnt > 18) && (t_cnt > 17) && (q_cnt < 2) && (z_cnt <
2)) {
                                                                        condition = 1;
                                                                }

                                                                // x not all-zero modulo 2 or 13 conditions
                                                                int count_temp = 0;
                                                                for (int r = 0; r < 5; r++) {
                                                                        if (((x[r] % 2) == 0) || ((x[r] % 13) == 0)) {
                                                                                count_temp++;
                                                                        }
                                                                }

                                                                if ((count_temp != 5) && (condition == 1)) {
                                                                        printf("candidate column: %d %d %d %d %d₩n₩n",
x[0], x[1], x[2], x[3], x[4]);
                                                                }

                                                        }
                                                }
                                        }
                                }

                        }
                        clock_t end = clock();
                        printf("Time: %lf₩n", (double)(end - start) / CLOCKS_PER_SEC);
                }

                printf(" ₩n₩n forloop 개수: %d ₩n", forloopcnt);

                return 0;

}
```

- 2. Inverse Key matrix 찾기(Colab 이용)

```
import numpy as np

Ciphertext = "HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVEIIYTKIOTQRXXQVSQLISVVOCNGCUXPKPIUBO
HTVKCFKWNJSEZYSSUTUOESIXKAPVFXNZHAOQTLCGYJVAEHLNNKEESQMKSHKKDFCNZSRHRDKHSDKFXVPTGMKRUPZBIKEVNYEKXMFXKFYMWYU
DZDENEWNKDAOUXGPCXZDLCSNFGCMCSNUAOJDBLQTAHEWYZCHQJYKSNUWOKQKONZGOKDXGUXKEMWQMCFGUEAVKHDIIATCHVTGYMGKJMLNPCN
AYKMIRWEETIYQKELEGLQOVKISFNUDAJQIQYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZGDBIUDDHCUDFRFOVSZXADSHYSGLTQBMNEMKDCFSOZ
SRDYLIHIAXCMGMFEIDNZKOVJEOIEFNWWQEDRLZYZIZXADSHYSGLJYFWDUAKSIOGOZOXWYPBUFEPNBIRJUJNDZJJYMURKNCIKPWLRMRIAGVS
XTYNIWPROHLDHQOMBEKZURQCLQOVKISFNUAFQBHGPCLHZTPJVPXIZKLQSNVKIJAEITTNVSVWNFYVATDEMKDCTGIHKZTVGZYXTYQEDBACFMN
CAHRDKHSDKFXZXXGMJOSLPSZBMOILMMWRALAFFMNXXDYFBIYQVVOHSWKGBIRJGTBYQLKIJAEQBTAXGFGAVUIJADHQKLFWRJXYFVIGGQZNBH
SUIYOZALSKIABLWQNXNXKOAJAIKHXODXWORVDOGBMHOPLQJZALQJZALIKTKLENZHQAVYUEUFEVLUXHGOWNMGWXUIAHGQOMNCKFQLIPBNKVW
DLNGMJCOBFKIGBYWPAHMMPQLUTOGECXITZVVAJEOIDCNWMFNLOBGQXCYFWQFWVXWRKWYGBFHJVLBAWBOUQEKHZHSZZIZARYITDCLQFPGBTJ
MQVSQLIHPEJONCYMZWTVJVZOBOMOHPSXMPUKVAGXIPOQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBEUKVNCAWANSNXISVVOWHQGQFBGWKQEGBIF
RGIZUJQWIMFANTGBHWGVAGXIPOQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQFUEVPLLIODEYRYLUTOGPYAFHJFI
VOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMGIMQDSYNWMFNIEITWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQ
ICHNUUQFJYUCQUO".replace(" ", "")

def _charTOint(char):
    return (ord(char)-65) % 26

def _intTochar(num):
    return chr(num%26+65)

def _get_cipher_matrix(cipher, n, d):
    m = n//d
    C = np.array([[0]*d for i in range(m)]) # (mxd)
    for i in range(n):
```

```
        r = int(i / d)
        c = i % d
        C[r][c] = _charTOint(cipher[i])
    return C

def _normalize_dict(in_dict):
    total = 1.0*np.array(list(in_dict.values())).sum()
    for i in in_dict.keys():
        in_dict[i] /= total
    return in_dict

C = _get_cipher_matrix(cipher = Ciphertext, n = len(Ciphertext), d=5)

from itertools import permutations

chae_c0 = [3,19, 11, 2, 3]
chae_c1 = [9, 7, 12, 20, 13]
chae_c2 = [12, 18, 4, 9, 0]
chae_c3 = [17, 24, 13, 11, 21]
chae_c4 = [18, 12, 6, 16, 23]

def decrypt(K_inv):
    decrypted = np.matmul(C, K_inv) % 26  #(mxd) (dxd) = (mxd)
    decrypted = [_intTochar(dec) for dec in decrypted.reshape(-1)]
    return ''.join(decrypted)

res = []

a = [0, 1, 2, 3, 4]
chae_c = [chae_c0, chae_c1, chae_c2, chae_c3, chae_c4]
for permute in list(permutations(a, 5)):
    K_inv = []
    for i in permute:
        K_inv.append(chae_c[i])
    K_inv = np.transpose(np.array(K_inv))

    res.append(decrypt(K_inv))

for r in res:
  print(r)
```

- 3. 발견한 Inverse Key matrix로 Cipher text를 Plain text로 복호화

```
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>

int main() {

          // give Cipher text as an input
          char Cipher[] =
"HRDKHUBHAAMAEQMTMZSHGBAKFUBHAASYRXUNKYUAATQCTLUTOGEWVAJGVEIIYTKIOTQRXXQVSQLISVVOCNGCUXPKPIUBOHTVKCFKWNJSEZYSSUTUOESIXKAPVFXNZHA
```

```
OQTLCGYJVAEHLNNKEESQMKSHKKDFCNZSRHRDKHSDKFXVPTGMKRUPZBIKEVNYEKXMFXKFYMWYUDZDENEWNKDAOUXGPCXZDLCSNFGCMCSNUAOJDBLQTAHEWYZCHQJYKSNU
WOKQKONZGOKDXGUXKEMWQMCFGUEAVKHDIIATCHVTGYMGKJMLNPCNAYKMIRWEETIYQKELEGLQOVKISFNUDAJQIQYBXQTMZSHGBAKFZRCNWRSODAFKKXWGAZGDBIUDDHCU
DFRFOVSZXADSHYSGLTQBMNEMKDCFSOZSRDYLIHIAXCMGMFEIDNZKOVJEOIEFNWWQEDRLZYZIZXADSHYSGLJYFWDUAKSIOGOZOXWYPBUFEPNBIRJUJNDZJJYMURKNCIKP
WLRMRIAGVSXTYNIWPROHLDHQOMBEKZURQCLQOVKISFNUAFQBHGPCLHZTPJVPXIZKLQSNVKIJAEITTNVSVWNFYVATDEMKDCTGIHKZTVGZYXTYQEDBACFMNCAHRDKHSDKF
XZXXGMJOSLPSZBMOILMMWRALAFFMNXXDYFBIYQVVOHSWKGBIRJGTBYQLKIJAEQBTAXGFGAVUIJADHQKLFWRJXYFVIGGQZNBHSUIYOZALSKIABLWQNXNXKOAJAIKHXODX
WORVDOGBMHOPLQJZALQJZALIKTKLENZHQAVYUEUFEVLUXHGOWNMGWXUIAHGQOMNCKFQLIPBNKVWDLNGMJCOBFKIGBYWPAHMMPQLUTOGECXITZVVAJEOIDCNWMFNLOBGQ
XCYFWQFWVXWRKWYGBFHJVLBAWBOUQEKHZHSZZIZARYITDCLQFPGBTJMQVSQLIHPEJONCYMZWTVJVZOBOMOHPSXMPUKVAGXIPOQUQUQBCKXZJSZAHEWYHAEMKOJCCCFBE
UKVNCAWANSNXISVVOWHQGQFBGWKQEGBIFRGIZUJQWIMFANTGBHWGVAGXIPOQUQTTRMWDHDGRFENKYPZVCLNQAUBTZSRYGVGOWSVROENABMZTOHZRQFUEVPLLIODEYRYL
UTOGPYAFHJFIVOSFMPBSHLEKWYWJYTFYETAZQCRFTFHOMACOQVTWKLKYMGIMQDSYNWMFNIEITWMBVVWANBQFVUSKZOTLCCWABAGHWZBZHRDKHDTUOMUUUGQICHNUUQFJ
YUCQUO"; //1285bytes

	// define Cipher text as an assigned integer(0-25) into a 257*5 matrix
	int C[257][5] = { 0, }; // m=257, d=5
	int tmp = 0;
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			C[i][j] = Cipher[tmp] - 65;
			tmp++;
		}
	}

	// check the input cipher matrix C
	printf("₩n Matrix C ₩n");
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			printf("%d  ", C[i][j]);
		}
		printf("₩n");
	}
	printf("₩n");

	// check if Cipher text transformed correctly
	printf("₩n Ciher text ₩n");
	for (int i = 0; i < 257; i++) {
		for (int j = 0; j < 5; j++) {
			printf("%c", (char)(C[i][j] + 65));
		}
	}
	printf("₩n");


	// 5. set the matrix K_inverse found by Bruteforce way
	int K_inverse[5][5]; // # m = 257, d = 5

	K_inverse[0][0] = 3;
	K_inverse[1][0] = 19;
	K_inverse[2][0] = 11;
	K_inverse[3][0] = 2;
	K_inverse[4][0] = 3;

	K_inverse[0][1] = 17;
	K_inverse[1][1] = 24;
	K_inverse[2][1] = 13;
	K_inverse[3][1] = 11;
	K_inverse[4][1] = 21;

	K_inverse[0][2] = 12;
	K_inverse[1][2] = 18;
	K_inverse[2][2] = 4;
	K_inverse[3][2] = 9;
	K_inverse[4][2] = 0;

	K_inverse[0][3] = 9;
	K_inverse[1][3] = 7;
	K_inverse[2][3] = 12;
	K_inverse[3][3] = 20;
	K_inverse[4][3] = 13;

	K_inverse[0][4] = 18;
	K_inverse[1][4] = 12;
	K_inverse[2][4] = 6;
	K_inverse[3][4] = 16;
	K_inverse[4][4] = 23;

	// print K_inverse
```

```
		printf(" ₩n ---------- K_inverse ---------- ₩n");
		for (int i = 0; i < 5; i++) {
			for (int j = 0; j < 5; j++) {
				printf("%d  ", K_inverse[i][j]);
			}
			printf("₩n");
		}


		// print Plaintext
		printf(" ₩n ---------- PLAIN TEXT ---------- ₩n");
		// p = c*k_inverse
		int PLAIN[257][5];
		for (int i = 0; i < 257; i++) {
			for (int j = 0; j < 5; j++) {
				PLAIN[i][j] = 0;
				for (int k = 0; k < 5; k++) {
					PLAIN[i][j] += (C[i][k]) * K_inverse[k][j];
				}
				PLAIN[i][j] = (PLAIN[i][j] % 26);
			}
		}


		for (int i = 0; i < 257; i++) {
			for (int j = 0; j < 5; j++) {
				printf("%c", (char)(PLAIN[i][j] + 65));
			}
		}

		printf(" ₩n₩n");

		return 0;

}
```