Information Security

1st Project

2015410039 이현건

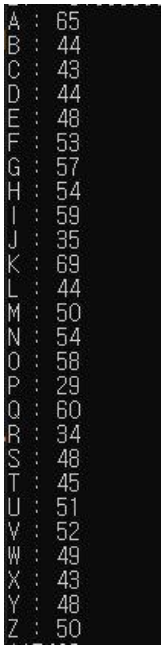
STEP 1. d*d key matrix의 d 추정하기

Hill cipher는 d개의 글자를 d*d matrix를 사용해서 암호화하기 때문에, d보다 작은 단위에서는 언어적 특성을 숨길 수 있습니다.

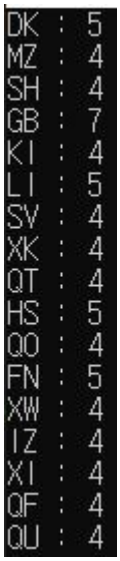
그렇다면 암호문을 1,2...n개 단위로 각각 분할해서 중복 문자열 출현 빈도를 조사해, 실제 영어에서 나타나는 빈도와 비교한다면 가장 가능성 높은, 언어의 특징이 나타나는 d의 후보를 얻을 수 있다고 생각했습니다.

```
int keysize = 1;
char* countstr[1000];
int count[1000] = { 0, };
int num = 0;
for (int i = 0; i < 1000; i++) {
    countstr[i] = (char*)calloc(keysize+1, sizeof(char));
}
for (int i = 0; i < len; i += keysize) {
    bool c = true;
    char* temp = (char*)calloc((keysize+1), sizeof(char));
    strncpy(temp, &str[i], keysize);
    for (int j = 0; j < num; j++) {
        if (strcmp(countstr[j], temp)==0) {
            count[j]++;
            c = false;
        }
    }
    if (c) {
        strcpy(countstr[num], temp);
        count[num] = 1;

        num++;
    }

}
for (int i = 0; i < num; i++) {
    if (count[i] > 1) {
        printf("%s : %d\n", countstr[i], count[i]);
    }
}
```

이에 따라 keysize만큼 str문자열을 잘라가며 count하는 코드를 작성했으며, 여러 번 중복해서 등장한 문자열과 그 횟수를 출력하도록 했습니다.

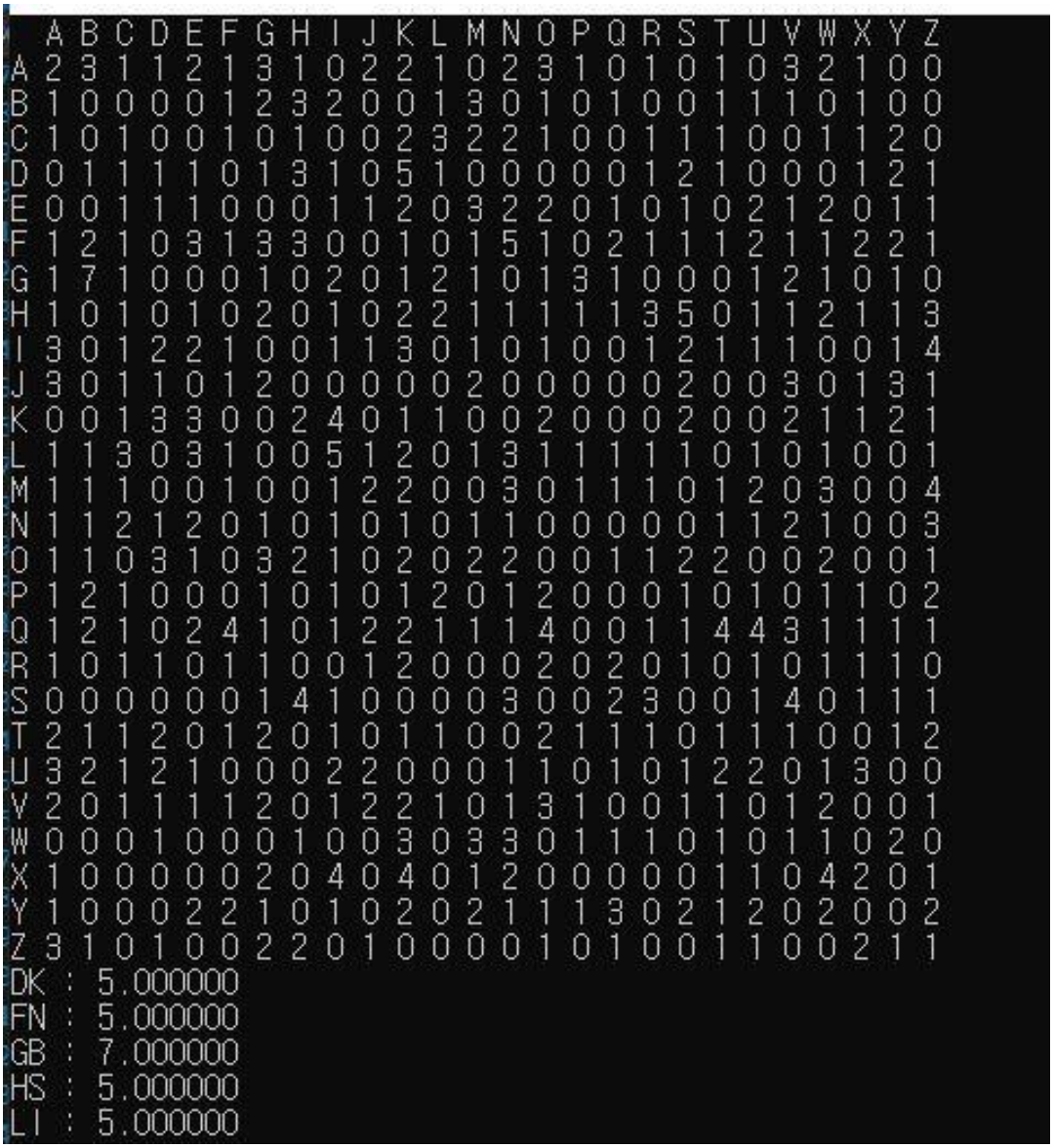


1*1의 경우엔 실제 영어의 경우 최대/최소 빈도수는 e(12.7%), z(0.1%)로 100배에 가까운 큰 차이가 있는데 비해 주어진 ciphertext의 경우는 최대빈도 69, 최소빈도 29로 2배의 차이만 나타났으므로 제외하였습니다.

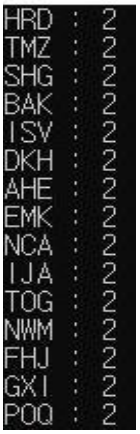


2*2의 경우엔 1285글자를 2글자씩 나누어 암호화하므로 총 분할 수는 642.5입니다. 즉, 최댓값인 GB, 7의 경우 1.1%, 5개의 빈도는 0.9%정도로 실제 영어에서 bigram frequency 최댓값인 TH, 2.71%와는 차이가 있습니다.

하지만 2글자 단위로 묶기 때문에 T와 H가 따로 묶이는 경우가 있으며, 이에 따라 2.71%의 1/2을 적용하면 1.35%이므로 어느정도 연관성이 있다고 생각할 수 있습니다.



또한 다음과 같이 모든 가능한 2글자 조합에 대해 출현빈도를 출력해 볼 경우, 대부분 0,1인 와중에 일부 값들에 3~7으로로 어느정도 frequency가 나타난다고 볼 수 있습니다.



3*3 의 경우엔, 3 이상의 빈도를 보이는 값은 없었습니다. 1285 글자를 3 글자씩 나누었으므로 총 분할 수는 428 입니다. 각 문자열은 2/428 = 0.5%정도의 비율을 지니며, 3 글자씩 나누어지므로 trigram frequency 에서 1.5%전후의 빈도를 지니는 단어가 많다면 영어의 특성을 띈다고 볼 수 있습니다.

THE : 1.81	ERE : 0.31	HES : 0.24
AND : 0.73	TIO : 0.31	VER : 0.24
ING : 0.72	TER : 0.30	HIS : 0.24
ENT : 0.42	EST : 0.28	OFT : 0.22
ION : 0.42	ERS : 0.28	ITH : 0.21
HER : 0.36	ATI : 0.26	FTH : 0.21
FOR : 0.34	HAT : 0.26	STH : 0.21

하지만 실제 영어의 trigram frequency 에선 1%이상의 단어는 THE 하나뿐이고, 나머지는 0.7 의 낮은 값을 띄므로 위의 2*2 에 비해 그 특성이 덜 정확하게 나타난다고 판단했습니다.

```
HRDK : 2
SQLI : 2
SVVO : 2
VKIS : 2
```

4*4 의 경우 또한 중복되는 단어가 몹시 적어 제외했습니다.

```
HRDKH : 4
UBHAA : 2
TMZSH : 2
GBAKF : 2
LUTOG : 3
QVSQL : 2
ISVVO : 2
SDKFX : 2
AHEWY : 2
LQOVK : 2
ISFNU : 2
ZXADS : 2
HYSGL : 2
EMKDC : 2
KIJAE : 2
QJZAL : 2
NWMFN : 2
VAGXI : 2
POQUQ : 2
```

5*5 의 경우, 전체 분할의 개수는 1285/5 로 257 개뿐인데 반해, 3~4 번 중복되어 나타나는 단어가 있었습니다.

OFTHE :	0.18	ANDTH :	0.07	CTION :	0.05
ATION :	0.17	NDTHE :	0.07	WHICH :	0.05
INTHE :	0.16	ONTHE :	0.07	THESE :	0.05
THERE :	0.09	EDTHE :	0.06	AFTER :	0.05
INGTH :	0.09	THEIR :	0.06	EOFTH :	0.05
TOTHE :	0.08	TIONA :	0.06	ABOUT :	0.04
NGTHE :	0.08	ORTHE :	0.06	ERTHE :	0.04

실제 영어의 quintgram frequency 는 최댓값인 'ofthe'도 0.2%를 넘지 못하는 몹시 작은 값이지만, 특정단어의 반복 등을 고려해서 어느정도 가능성이 높다고 판단했습니다.

이 이후 6*6, 7*7 등에선 어떠한 경향성도 나타나지 않았으며, 따라서 2*2 -> 5*5 -> 3*3 의 순서로 key 크기를 가정한 채 암호를 해독하였습니다.

STEP 2. 2*2 Key Hill cipher

1)Quadgram 이용

Key 가 2*2 크기일 경우, 서로 다른 2 개의 plaintext-ciphertext pair 를 알고 있다면 연립방정식을 세워 key 를 찾을 수 있습니다.

```
fu pc mt gz ky uk bq fj hu kt zk ki xt ta
----------------------------------------
of th e. .. .. .. .. .. .. .. .. .. .. ..
.o ft he .. .. .. .. .. .. .. .. .. .. ..
.. of th e. .. .. .. .. .. .. .. .. .. ..
.. .o ft he .. .. .. .. .. .. .. .. .. ..
.. .. of th e. .. .. .. .. .. .. .. .. ..
.. .. .o ft he .. .. .. .. .. .. .. .. ..
...
```

영어에서 빈번하게 나타나는 어떠한 quadgram 글자가 원문 plaintext 에 있다고 가정하면, 모든 ciphertext 의 4 글자 조합에 대해 그 키를 획득할 수 있습니다. 예를 들어 'that'을 가정하면, ciphertext 의 1~4 번 글자에 대해 p*k = c 를 만족하는 key k 를 계산하고, 3~6 번 자리에 대해 key 를 계산하는 과정을 반복해서 각 키들을 구할 수 있습니다.

```
int key_a, key_b, key_c, key_d;

bool findans(int l1, int l2, int l3, int l4, int loc) {
    int a, b, c, d;
    int inverse = (((l1*l4 - l2 * l3) % 26)+26%26);
    int inverse_mod_26=0;



    for (int i = 0; i < 26; i++) {
        if ((i*inverse) % 26 == 1) inverse_mod_26 = (((i%26)+26)%26);
    }
    a = (l4 * inverse_mod_26) % 26;
    b = (((-l2 * inverse_mod_26) % 26) + 26) % 26;
    c = (((-l3 * inverse_mod_26) % 26) + 26) % 26;
    d = (l1 * inverse_mod_26) % 26;


    key_a = ((str[loc] - 'A') * a + (str[loc + 2] - 'A') * b) % 26;
    key_b = ((str[loc+1] - 'A') * a + (str[loc + 3] - 'A') * b) % 26;
    key_c = ((str[loc] - 'A') * c + (str[loc + 2] - 'A') * d) % 26;
    key_d = ((str[loc+1] - 'A') * c + (str[loc + 3] - 'A') * d) % 26;

    if (gcd(key_a*key_d - key_b * key_c, 26)) {
        return true;
    }
    return false;

}
```

이를 위해, plaintext 4 자리와 ciphertext 의 위치 loc 를 인자로 받아 key 를 계산하는 함수 findans 를 만들었습니다. P*K=C 이므로, P 의 inverse 인 P'을 계산, K = P'*C 식에 대입해서 K 를 획득했습니다.

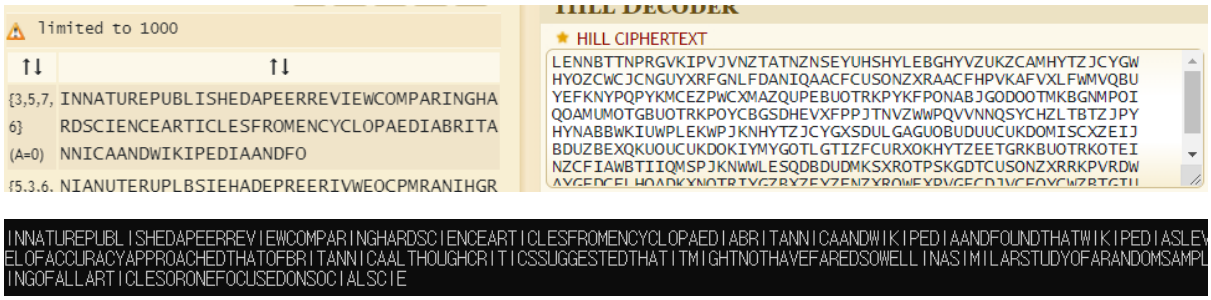
또한 hill cipher(mod 26)의 key 로서 성립하려면 행렬식 ad-bc 가 26 과 서로소 관계여야 하므로, 해당 조건을 만족할 경우에만 true 를 반환하도록 했습니다.

```
//find with quadgram
int char1 = 'T';
int char2 = 'H';
int char3 = 'A';
int char4 = 'T';
for (int i = 0; i < len - 4; i+=2) {
    if (findans(char1 - 'A', char2 - 'A', char3 - 'A', char4 - 'A', i)) {
        //find inverse key
        int inverse = ((key_a*key_d - key_b * key_c) % 26 + 26) % 26;
        int inverse_mod_26 = 0;
        for (int j = 0; j < 26; j++) {
            if ((j*inverse) % 26 == 1) inverse_mod_26 = (((j % 26) + 26) % 26);
        }
        if (inverse_mod_26 != 0) {
            int dec_key_a = key_d * inverse_mod_26 % 26;
            int dec_key_b = (((-key_b * inverse_mod_26) % 26) + 26) % 26;
            int dec_key_c = (((-key_c * inverse_mod_26) % 26) + 26) % 26;
            int dec_key_d = key_a * inverse_mod_26 % 26;
            test_str(dec_key_a, dec_key_b, dec_key_c, dec_key_d);
        }
    }
}
```

그리고 findans 함수에서 true 가 반환되었을 경우, 즉 키가 존재할 경우 해당 키의 inverse 를 구해 이를 인자로 test_str 함수를 실행했습니다.

```
bool test_str(int a, int b, int c, int d) {
    char* plain_str = (char*)calloc(300,sizeof(char));
    int chk[26] = { 0, };
    for (int i = 0; i < len; i += 2) {
        if (i < 300) {
            plain_str[i] = ((((str[i] - 'A') * a + (str[i + 1] - 'A') * c) % 26) + 26) % 26 + 'A';
            plain_str[i + 1] = ((((str[i] - 'A') * b + (str[i + 1] - 'A') * d) % 26) + 26) % 26 + 'A';
        }
        chk[((((str[i] - 'A') * a + (str[i + 1] - 'A') * c) % 26) + 26) % 26]++;
        chk[((((str[i] - 'A') * b + (str[i + 1] - 'A') * d) % 26) + 26) % 26]++;
    }

    if (chk['T' - 'A'] > len *(6.0 / 100.0) && chk['E' - 'A'] > len *(8.0 / 100.0)
        && chk['Z' - 'A'] < len *(2.0 / 100.0) && chk['X' - 'A'] < len *(2.0 / 100.0)) {
        printf("%s\n\n\n", plain_str);
        return true;
    }

    return false;
}
```

test_str 함수는 inverse key 값 4 개를 인자로 받아 ciphertext 를 plaintext 로 복호화하며, 모든 값을 출력하지 않고, 영어의 빈도수 특성을 고려하여 plaintext 의 E,T 의 빈도가 일정 이상인지, X,Z 의 값이 일정 이하인지 검사한 후 이를 만족하는 경우에 출력하도록 했습니다.



위 알고리즘이 정확하게 작동하는지 확인하기 위해 'that'이 포함된 문단을 암호화한 ciphertext 로 테스트했으며, 원문과 동일한 복호화 결과를 얻었으므로 정확하다고 판단했습니다.

하지만 'TION' 'NTHE' 'THER'등 가장 빈번한 quadgram 들을 통해 수행했음에도 합당한 결과값을 얻을 수 없었습니다.

2)Brute-force 방식

2*2 hill-cipher 의 복잡도는 26 의 4 제곱으로 그 연산이 50 만회 이하이기 때문에 brute-force 로 해결 가능하다고 생각했습니다.

```
for (int a = 0; a < 26; a++) {
    for (int b = 0; b < 26; b++) {
        for (int c = 0; c < 26; c++) {
            for (int d = 0; d < 26; d++) {
                if (gcd((a*d-b*c),26)) {
                    test_str(a, b, c, d);
                }
            }
        }
    }
}
```

2*2 key 행렬의 각 자리를 a,b,c,d 로 지정 후, 0 부터 26 까지 반복문들 돌면서 이전에 사용했던 test_str 함수를 통해 각 킷값을 검증했습니다.

하지만 이 방식에서도 합당한 plaintext 를 얻을 수 없었습니다.

때문에 2*2key 가 아니라고 판단, 다음 후보였던 5*5key 를 가정하고 해독을 시도했습니다.

STEP 3. 5*5 Key Hill cipher

5*5 key 의 경우엔, 2*2 단계에서 사용했던 방법이 사용 불가능했습니다.

Brute force 는 26 의 25 제곱만큼 경우의 수가 있으므로 현실적인 시간내에 해결 불가능하며, 임의의 문자열을 가정해 키를 추정하는 방법은 너무 긴 문자열을 필요로 하여 힘들었습니다.

때문에 5*5 key 전체가 아닌 하나의 행만 추측하도록 하여 그 경우의 수를 26 의 5 제곱으로 낮추어 계산하였습니다. 즉, Ciphertext 를 5 글자 단위로 분리한 후 key 의 일부로 가정한 어떠한 5*1 행렬과 곱셈을 수행하면 plaintext 의 일부를 복구할 수 있습니다. 이 plaintext 의 일부분 또한 영어의 문자 빈도 특성을 띌 것이므로 모든 5*1 의 경우의 수에 대해 복호화를 진행 후 각각에 대해 단어 빈도수를 체크했습니다.

```
keysize = 5;
int plain_size = len / keysize;
int count_temp[26];
int sum;

char* plain_candidate[100];
for (int i = 0; i < 100; i++) {
    plain_candidate[i] = (char*)malloc(sizeof(char)*(plain_size+1));
}
int plain_cnt = 0;

for (int a11 = 0; a11 < 26; a11++) {
    for (int a21 = 0; a21 < 26; a21++) {

        for (int a31 = 0; a31 < 26; a31++) {

            for (int a41 = 0; a41 < 26; a41++) {

                for (int a51 = 0; a51 < 26; a51++) {

                    for (int i = 0; i < 26; i++) {
                        count_temp[i] = 0;
                    }

                    for (int i = 0; i < len - keysize; i += keysize) {
                        for (int j = i; j < i + keysize; j++) {
                            sum = (str[i] - 'A') * a11 + (str[i + 1] - 'A') * a21 + (str[i + 2] - 'A') * a31 + (str[i + 3] - 'A') * a41 + (str[i + 4] - 'A') * a51;
                        }
                        count_temp[sum % 26]++;
                    }

                    if (count_temp['I' - 'A'] > plain_size*(5.0 / 100.0) && count_temp['A' - 'A'] > plain_size*(6.0 / 100.0) &&
                        count_temp['E' - 'A'] > plain_size*(7.0 / 100.0) && count_temp['T' - 'A'] > plain_size*(7.0 / 100.0) &&
                        count_temp['J' - 'A'] < plain_size*(1.0 / 100.0) && count_temp['Z' - 'A'] < plain_size*(1.0 / 100.0) &&
                        count_temp['X' - 'A'] < plain_size*(1.0 / 100.0) && count_temp['Q' - 'A'] < plain_size*(1.0  / 100.0)) {

                        for (int i = 0; i < len; i += keysize) {
                            for (int j = i; j < i + keysize; j++) {
                                sum = (str[i] - 'A') * a11 + (str[i + 1] - 'A') * a21 + (str[i + 2] - 'A') * a31 + (str[i + 3] - 'A') * a41 + (str[i + 4] - 'A') * a51;
                            }
                            plain_candidate[plain_cnt][i/keysize] = sum % 26 + 'A';

                        }
                        plain_candidate[plain_cnt][len / keysize] = '\0';
                        plain_cnt++;
```

위 코드가 해당 과정을 진행하는 코드이며, a11~a51 변수들이 0~25 까지 반복하며 모든 값에 대해 복호화를 하여, 각 알파벳의 빈도수를 측정합니다. 그리고 해당 빈도수가 영어에서 나타나는 빈도수와 유사한 경향성이 있을 경우, 그 문자열을 plain_candidate 변수에 저장하도록 했습니다.

그 결과 조건을 만족하는 문자열, plaintext 후보가 총 6 종류 확인되었습니다.

```
void per(int* arr, int loc, int size, char**plain_candidate) {
    if (loc == size) {
        print_plain(arr, size, plain_candidate);
    }
    else {
        for (int i = loc; i < size; i++) {
            int temp = arr[i];
            arr[i] = arr[loc];
            arr[loc] = temp;

            per(arr, loc+1, size, plain_candidate);

            temp = arr[i];
            arr[i] = arr[loc];
            arr[loc] = temp;
        }
    }
}
```

결과로 나온 6 가지 문자열 중 5 개를 선택해서 순열을 만들면 그 순서로 plaintext 전체를 복구할 수 있으며, 이를 위해 1~6 의 수에서 5 개를 선택하는 모든 경우의 수를 순회하는 함수를 작성해, 각 순열의 경우마다 plaintext 를 복구하도록 했습니다.

```
void print_plain(int* arr, int size, char**plain_candidate) {
    int cnt = 0;
    char prev='0';
    for (int i = 0; i < len / size; i++) {
        for (int j = 0; j < size; j++) {
            if (plain_candidate[arr[j] - 1][i] == 'H' && prev == 'T') {
                cnt++;
            }
            prev = plain_candidate[arr[j] - 1][i];
        }
    }
    if (cnt > 30) {
        for (int i = 0; i < len / size; i++) {
            for (int j = 0; j < size; j++) {
                printf("%c", plain_candidate[arr[j] - 1][i]);
            }
        }
        printf("\n\n\n");
    }
}
```

이때 6 종류 숫자 중 5 개를 선택해서 만들 수 있는 순열의 가짓수는 6*5*4*3*2 = 720 가지이므로 이를 육안으로 분석하는 것은 힘들다고 판단했습니다. 따라서 plaintext 의 이전 문자와 현재 문자를 검사해, 가장 자주 나타나는 bigram 인 'TH'를 카운트했으며, TH 의 등장 빈도인 3.5%에서 오차를 감안해 2.5%, 약 30 회 이상의 빈도를 보였을 경우에만 전체 plaintext 를 출력하도록 하였습니다.

```
CRYPTANALYSISISTHESTUDYOFANALYXINGINFORMATIONSYSTEMSINTHESTUDYOFANALYZINGINFORMATIONSYSTEMSINORDERTOSTUDYTHEHIDDENASPECT
SOFTHESYSTEMSCRYPTANALYSISISUSEDTOBREACHCRYPTOGRAPHICSECURITYSYSTEMSANDGAINACCESSTOTHECONTENTSOFENCRYPTEDMESSAGESEVENIFT
HECRYPTOGRAPHICKEYISUNKNOWNINADDITIONTOMATHEMATICALANALYSISOFCRYPTOGRAPHICALGORITHMSCRYPTANALYSISINCLUDESTHESTUDYOFSIDEC
HANNELATTACKSTHATDONOTTARGETWEAKNESSESINTHECRYPTOGRAPHICALGORITHMSTHEMSELVESBUTINSTEADEXPLOITWEAKNESSESINTHEIRWEAKIMPLEM
ENTATIONEVENTHOUGHTHEGOALHASBEENTHESAMETHEMETHODSANDTECHNIQUESOFCRYPTANALYSISHAVECHANGEDDRASTICALLYTHROUGHTHEHISTORYOFCR
YPTOGRAPHYADAPTINGTOINCREASINGCRYPTOGRAPHICCOMPLEXITYRANGINGFROMTHEPENANDPAPERMETHODSOFTHEPASTTHROUGHMACHINESLIKETHEBRIT
ISHBOMBESANDBOLOSSUSCOMPUTERSATBLETCHLEYPARKINWORLDWARTWOTOTHEMATHEMATICALLYADVANCEDCOMPUTERIZEDSCHEMESOFTHEPRESENTMETHO
DSFORBREAKINGMODERNCRYPTOSYSTEMSOFTENINVOLVESOLVINGCAREFULLYCONSTRUCTEDPROBLEMSINPUREMATHEMATICSTHEBESTKNOWNBEINGINTEGER
FACTORIZATIONGIVENSOMEENCRYPTEDDATATHEGOALOFTHECRYPTANALYSTISTOGAINASMUCHINFORMATIONASPOSSIBLEABOUTTHEORIGINALUNENCRYPTE
DDATATISUSEFULTOCONSIDERTWOASPECTSOFACHIEVINGTHISTHEFIRSTISBREAKINGTHESYSTEMTHATISDISCOVERINGHOWTHEENCIPHERMENTPROCESSWO
RKSTHESECONDISSOLVINGTHEKEYTHATISUNIQUEFORAPARTICULARENCRYPTEDMESSAGEORGROUPOFMESSAGE
```

그 결과로 출력된 하나의 plaintext 후보입니다. CRYPTANALYSIS 라는 존재하는 영단어로 시작하기 때문에 올바른 plaintext 일 가능성이 크다고 판단하였습니다.

```
CRYPTANALYSIS IS THE STUDY OF ANALYXING INFORMATION SYSTEMS IN THE STUDY OF ANALYZING INFORMATION SYSTEMS IN ORDER TO STUDY THE HIDDEN ASPECTS OF THE SYSTEMS

CRYPTANALYSIS IS USED TO BREACH CRYPTOGRAPHIC SECURITY SYSTEMS AND GAIN ACCESS TO THE CONTENTS OF ENCRYPTED MESSAGES EVEN IF THE CRYPTOGRAPHIC KEY IS UNKNOWN

IN ADDITION TO MATHEMATICAL ANALYSIS OF CRYPTOGRAPHIC ALGORITHMS CRYPTANALYSIS INCLUDES THE STUDY OF SIDE CHANNEL ATTACKS THAT
DO NOT TARGET WEAKNESSES IN THE CRYPTOGRAPHIC ALGORITHMS THEMSELVES BUT INSTEAD EXPLOIT WEAKNESSES IN THEIR WEAK IMPLEMENTATION

EVEN THOUGH THE GOAL HAS BEEN THE SAME THE METHODS AND TECHNIQUES OF CRYPTANALYSIS HAVE CHANGED DRASTICALLY THROUGH
THE HISTORY OF CRYPTOGRAPHY ADAPTING TO INCREASING CRYPTOGRAPHIC COMPLEXITY RANGING FROM
THE PEN AND PAPER METHODS OF THE PAST THROUGH MACHINES LIKE THE BRITISH BOMBES AND BOLOSSUS COMPUTERS AT BLETCHLEY
PARK IN WORLDWAR TWO TO THE MATHEMATICALLY ADVANCED COMPUTERIZED SCHEMES OF THE PRESENT

METHODS FOR BREAKING MODERN CRYPTOSYSTEMS OFTEN INVOLVE SOLVING CAREFULLY CONSTRUCTED PROBLEMS IN PURE MATHEMATICS THE BEST KNOWN BEING INTEGER FACTORIZATION

GIVEN SOME ENCRYPTED DATA THE GOAL OF THE CRYPTANALYSTIS TO GAIN AS MUCH INFORMATION AS POSSIBLE ABOUT THE ORIGINAL UNENCRYPTED DATA

TIS USEFUL TO CONSIDER TWO ASPECTS OF ACHIEVING THIS

THE FIRST IS BREAKING THE SYSTEM THAT IS DISCOVERING HOW THE ENCIPHERMENT PROCESS WORKS

THE SECOND IS SOLVING THE KEY THAT IS UNIQUE FOR A PARTICULAR ENCRYPTED MESSAGE OR GROUP OF MESSAGE
```

직접 단어 사이에 띄어쓰기를 추가하고 문장으로 추정되는 부분을 분리한 결과입니다. 일부 부정확한 단어(ANALXING)가 존재하긴 했지만, 충분히 해석 가능한 영어 문장이 완성되었으므로 plaintext 의 복구를 완료한 것으로 판단하였습니다.

참고자료

English letter frequency

http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/

Cryptanalysis of the Hill Cipher

http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-hill-cipher/

http://alexbarter.com/cryptanalysis/breaking-hill-cipher/