[REPORT - 실습3]

-Caesar Cipher Cracking Program-

과 목 명 : 컴퓨터보안 - 02
교 수 : 김영부 교수님
학번 : 2020112119
이름 : 강동희
제출일 : 2023.04.04

목 차

1. 서론

1.1. 문제 정의
1.2. 문제 분석

2. 본론

2.1. 실습과제 : 시저 암호 생성 및 암호 해독 실습
2.2. 부가과제 : 전치 암호 생성 및 암호 해독 실습

3. 결론

3.1. 실습 결과 및 분석
3.2. 고찰(Discussion)

APPENDIX

A. 소스 코드

1 서 론

1.1 문제 정의

1. 시저 암호 생성 및 암호 해독 실습
   영단어 7~10개 사이로 구성된 문장을 생성하는 기능을 통해 해당 문장(평문)을 시저 암호를 적용하여 암호문을 생성한다. 이때 shift하는 정도는 랜덤으로 1~25사이의 값으로 부여하여 암호화 시킨다. 또한, 생성된 암호문을 대상으로 암호를 해독하여 평문으로 되돌리는 기능을 구현한다.

2. 전치 암호 생성 및 암호 해독 실습
   1과 같이 랜덤으로 문장을 생성하여 전치 암호를 적용한 암호문을 생성하며, 생성된 암호문들 대상으로 다시 평문으로 되돌리는 기능을 구현한다.

1.2 문제 분석

1. 치환 암호(Substitution Cipher)[1]
   일정한 법칙에 따라 평문(Plaintext)의 문자 단위를 다른 문자 단위로 치환하는 암호화 방식이다. 이때 암호화 시킬 때의 단위(Key)는 문자일 수도 있고, 비트 혹은 문자 여러 개로 대응시킬 수 있다. 실습에서 진행하는 치환 단위는 한 문자에 해당하여, 한 문자에 정해진 규칙에 따라 shift하여 암호화를 진행한다.
   치환 암호는 크게 '단일 치환암호'와 '다중 치환암호'로 분류할 수 있다. 치환 암호의 가장 기본 방식인 단일 치환암호로는 알파벳에서 Key 값을 이용해서 다른 문자가 되도록 치환하는 방식으로, 덧셈암호, 곱셈암호, 그리고 두 개의 암호법을 합친 아핀암호가 있다.
   다중 치환암호는 단일 치환암호의 단점인 알파벳의 빈도 정보를 분석해 유추할 수 있다는 단점을 보완하기 위해 빈도 정보를 이용할 수 없도록 암호화하는 방식이다. 대표적으로 자동키암호, 플레이페어 암호, 비즈네르 암호 등이 있다.

2. 시저 암호(Caesar Cipher)[2]
   치환 암호 중 하나로 문자를 일정한 단위(Key)만큼 이동하여 치환하여 암호화하는 방식이다. 시저 암호의 가장 취약한 단점으로는 중복되는 단어가 나열될 경우 유추가 쉬워진다는 점이다. 이를 통해 암호화에 사용된 Key

[1] https://ko.wikipedia.org/wiki/치환_암호

[2] https://en.wikipedia.org/wiki/Caesar_cipher

값을 알아내 다시 평문으로 되돌릴 수 있다.

시저 암호의 암호화 방식을 수식으로 정리하면

$$E(k,p) = (p+k) mod\ 26$$

수식 1. 시저암호의 암호화 함수

이다. E는 암호화 함수(Encryption function)이며, 암호화 함수에 사용되는 파라미터로 P(Plain Text), K(key)가 있다. 평문의 단어를 K만큼 이동하여 암호화 하는 방식이다. 이와 반대로 암호문을 다시 평문으로 되돌리기 위해 K값을 반대로 빼면 평문으로 만들 수 있다.

$$D(k,c) = (c-k) mod\ 26$$

수식 2. 시저암호의 복호화 함수

3. 전치 암호(Transposition Cipher)

전치 암호는 평문 배열을 특정 Key에 따라 암호화하는 방식이다. 평문을 Key의 길이에 따라 일정 간격으로 나누고, 나눈 문자 블록을 재배열 순서에 따라 재배치하는 방식으로 암호화가 된다. 전치 암호는 시저 암호의 단점이였던 빈도 분석법을 이용해도 다른 알파벳을 찾지 못해 해독하는 데 시간이 비교적으로 오래 걸린다는 것을 알 수 있다.

2 본 론

2.1 실습과제 : 시저 암호 생성 및 암호 해독 실습

1. 평문 생성

단어 데이터베이스는 nltk 라이브러리에서 제공하는 words 데이터베이스를 사용하여 단어를 랜덤으로 추출하였다. 7~10 개의 단어를 랜덤으로 정하여 한 문장을 구성하여 이를 100 개 생성하여 텍스트 파일에 저장하도록 구현하였다.

```
#단어 데이터베이스 사용을 위한 라이브러리
import nltk
import random
#로컬에 nltk 에서 제공하는 words 데이터베이스를 다운받는다.
nltk.download('/Users/kangdonghee/nltk_data')
nltk.download('words')
# 100 개의 문장 생성
num_sentences = 100
sentences = []
for i in range(num_sentences):
# 7~10 단어를 무작위로 조합하여 문장 생성
num_words = random.randint(7, 10)
words = [random.choice(nltk.corpus.words.words()) for _ in
range(num_words)]
sentence = ' '.join(words)
print(f'문장 생성 {i+1}/100 : {sentence}')
sentences.append(sentence)
# 파일에 저장
with open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt', 'w') as file:
file.write('\n'.join(sentences))
```

코드 1. generate_sentences.py 전체 코드

2. 시저암호 방식의 암호화

시저 암호의 방식에 따라 평문 텍스트 파일을 불러와 각 문장별로 1~25 의 랜덤한 정수를 암호화 키 값으로 부여하여 해당 키 값만큼 이동하여 암호화를 진행하였다. 마찬가지로 암호화 된 문장도 텍스트 파일에 저장하였다.

```
import random
def caesar_cipher(text, key):
    #반환할 암호문 선언
    shifted_text = ""
    for char in text:
        if char.isalpha():
            #시저암호의 수식 이용하여 암호화 진행
            if char.isupper(): #아스키코드값을 이용해 대문자일 경우 shift 진행
                shifted_char = chr((ord(char) - 65 + key) % 26 + 65)
            else: #소문자에 대해서도 shift 진행
```

```
                shifted_char = chr((ord(char) - 97 + key) % 26 + 97)
            else:
                shifted_char = char
            shifted_text += shifted_char
        print(f'평문 : {text}')
        print(f'암호화 Key(shift) : {key}')
        print(f'암호화 : {shifted_text}')
        return shifted_text
# generate_sentences.txt 파일에서 100 개의 문장을 가져온다.
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt", "r") as f:
    sentences = f.readlines()
# 각 문장을 caesar_cipher(text,key)를 적용하여 암호화.
encrypted_sentences = []
count = 0
for sentence in sentences:
    # 1 에서 25 사이의 랜덤한 정수로 key 값을 생성한다. (각 문장마다 랜덤한
키값으로 암호화 진행)
    key = random.randint(1, 25)
    encrypted_sentence = caesar_cipher(sentence.strip(), key)
    encrypted_sentences.append(encrypted_sentence)
    count +=1
    print(f'진행도 : {count}/100')
    print()
# 암호화된 100 개의 문장을 파일에 저장.
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]encrypted_sentences.txt", "w") as f:
    for sentence in encrypted_sentences:
        f.write(sentence + "\n")
```

코드 2. Caesar_Cipher.py 전체 코드

3. 시저 암호의 해독

암호화한 문장이 저장된 텍스트 파일을 불러와 해독하는 코드이다. 암호화에 사용된 암호화 키를 모르는 상태이므로, shift 키 값을 1 부터 25 까지 shift 하여 문장을 해독한다. 이때 해독한 문장 중 기존 단어 데이터베이스와 비교하여 일치한 단어가 가장 많은 문장을 후보군으로 선정하여 텍스트 파일에 저장하도록 구현하였다.

```
# 암호문이 저장된 파일 불러오기
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]encrypted_sentences.txt", "r") as
input_file:
    # 각각의 암호문 해독하기
    count = 0
    for line in input_file:
        print(f'{count + 1}/100 해독 진행..')
        encrypted = line.strip()
        # 모든 암호화 키 시도
        candidates = decrypt(encrypted)
        count += 1
```

```
        # 후보군 리스트에 저장된 문장들 중 일치하는 단어가 많은 문장 선택
        max_match = 0
        selected_candidate = ''
        for candidate in candidates:
            match_count = sum([word in word_list for word in
candidate.split()])
            if match_count > max_match:
                max_match = match_count
                selected_candidate = candidate
        # words 와 일치하는 단어가 많은 경우에만 파일에 입력
        if max_match > 0:
            output_file.write(selected_candidate + "\n")
output_file.close()
```

코드 3. Caesar_Decipher.py 중 메인 코드 부분

2.2 부가과제 : 전치 암호 생성 및 암호 해독 실습

1. 전치 암호의 암호화

전치 암호를 이용해 평문을 암호화 하기 위해 먼저 생성된 평문을 2 차원 배열로 변환하는 작업이 필요하다. 랜덤으로 row 를 선언한 뒤 문자열의 길이에 맞게 col 을 계산하고, 행렬의 크기가 문자열의 길이보다 클 경우를 대비해 padding 을 랜덤 알파벳으로 삽입하여 full-size 의 행렬로 변환한다.

```
def add_padding(text):
    #띄어쓰기를 포함할 경우 전치암호화하는 과정에서 줄바꿈이 발생하는 문제점 해결
    text = ''.join(text.split())
    text_len = len(text)
    # row, col 계산
    row = math.ceil(text_len**0.5)
    col = math.ceil(text_len/row)
    # padding 에 추가할 문자 계산
    pad_len = row * col - text_len
    pad = ''.join(random.choices(string.ascii_lowercase, k=pad_len))
    # padding 추가
    padded_text = text + pad
    # 이차원 배열에 넣기
    matrix = [list(padded_text[i:i+col]) for i in range(0, row*col, col)]
    return matrix
```

<코드 4. transposition_cipher.py 중 문자열을 행렬로 변환 과정>

전치 암호화 방식을 적용하기 위해 col을 랜덤한 순서로 선택하기 위한 key 리스트를 생성하고 해당 순서에 맞게 열을 선택해 row로 읽어 문자열로 변환하여 암호화문을 저장하도록 했다.

```
def transposition_cipher(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    # 열을 랜덤한 순서로 선택하기 위해 인덱스 리스트를 생성
    key = list(range(num_cols))
    random.shuffle(key)
    print(f'암호키(랜덤 생성) : {key}')
    # 랜덤한 순서로 열을 선택하여 1 차원 리스트로 저장
    result = []
```

```
    for col_idx in key:
        for row_idx in range(num_rows):
            result.append(matrix[row_idx][col_idx])

    # 1 차원 리스트를 문자열로 변환하여 반환
    return ''.join(result)
```

<코드 5. transposition_cipher.py 중 전치 암호화 과정>

2. 전치 암호의 해독

키 값을 랜덤으로 적용하는 방식으로 시도하고자 했으나, 성공하지 못해 <코드 5>에서 암호화한 Key 리스트를 따로 텍스트 파일에 저장해 해독 시 사용하도록 구현해보았다.

```
def decrypt_transposition_cipher(ciphertext, key, count):
    key = [int(char) for char in key]
    num_rows = math.ceil(len(ciphertext)/len(key))
    num_cols = len(key)
    print(f'해독 진행 : {count+1}/100')
    print(f'적용 키 값 : {key}')
    # 인덱스 리스트를 생성하여 역순으로 정렬
    try:
        key_inv = [key.index(i) for i in range(num_cols)]
    except ValueError:
        return None
    # 1 차원 리스트를 이차원 배열에 복원
    matrix = [list(ciphertext[i:i+num_cols]) for i in range(0, num_rows*num_cols,
num_cols)]
    # 복호화된 결과를 저장할 리스트
    result = []
    # 역순으로 정렬된 인덱스를 사용하여 원래의 순서로 문자열을 복원
    for col_idx in key_inv:
        for row_idx in range(num_rows):
            result.append(matrix[row_idx][col_idx])
    # 복호화된 결과를 문자열로 변환하여 반환
    final = ''.join(result)
    print(f'해독 결과 : {final}')
    return final
```

<코드 6. transposition_Decipher.py의 전치암호의 해독 과정>

3 결 론

3.1 실습 결과 및 분석

실습 1: 시저 암호 생성 및 암호 해독 실습

1) 평문 생성 결과

nltk 라이브러리의 words 단어 데이터베이스를 이용해 7 ~10개 단어를 랜덤으로 조합하여 100개의 문장을 생성 및 텍스트 파일로 저장되었다.

```
[nltk_data] Downloading package words to
[nltk_data]     /Users/kangdonghee/nltk_data...
[nltk_data]   Package words is already up-to-date!
문장 생성 1/100 : windfirm bacury iddat Tolstoyism radiostereoscopy configurationist rhomborectangular Zoraptera
문장 생성 2/100 : fractuosity pricker appliable Artocarpaceae upflung autolytic overcomplacently gassy
문장 생성 3/100 : Paleozoic arteriology descriptory Urocystis subtotem nodicorn countermutiny sorryhearted
문장 생성 4/100 : polycentric aware brettice Xiphura morosely narrowness xiphoidal spirality braird
문장 생성 5/100 : preadvertise metapodium malady falsework undertapster theomythology coeldership comitatus
문장 생성 6/100 : chessboard Ignatius inaugurative Gleditsia ovigerm uningenuous obsoletism empocket diarrhetic untaut
문장 생성 7/100 : itself meretriciousness vixen anticoagulative pegasoid perceptual nonmicrobic cradleside wolframate tenio
문장 생성 8/100 : diagraphic structuralism unironical aethalioid unsubscribed derivation taxicab entomologize
문장 생성 9/100 : terpeneless emberizine balustrade Hemipodius cud realist nepheline despitefulness Mathurin Panboeotian
문장 생성 10/100 : Demogorgon acrotomous Kluxer anteopercle ancience jellico algraphic fibula unsubsidiary
문장 생성 11/100 : tibiotarsus forejudge gauteite manumisable conscionably aboveproof canaliculi experimentalism anthracemia
문장 생성 12/100 : unpacker nonsubsiding archknave slotwise totemist dali confederatism
문장 생성 13/100 : polypigerous philosophistic appendiculate triangulopyramidal dishboard tergiversatory thither cased
문장 생성 14/100 : diaphane osmogene humbugable happily delayful news Jovinian palas serrirostrate
문장 생성 15/100 : comatulid sulphidize mellonides Pterocletes overget phaseal ventilable pseudopionnotes ataman unconciliating
문장 생성 16/100 : handistroke unsad Pterobranchia rapturousness absquatulate phenylglycolic pyelitis soundable
문장 생성 17/100 : semirelief lucubration Shulamite astraeiform coreceiver Hippocratea Guinevere optically
문장 생성 18/100 : amula fruitiness craniotabes Anteva Carborundum cyanhydric geisha clasper ghastily venturine
```

```
문장 생성 90/100 : soapbubbly unmet acquiescent arcocentrous harpsichordist superintellectual undistinguishing myophysical disjunct
문장 생성 91/100 : stylebook capewise bush cop transitivity gradation elenchtical ophthalmatrophia Adenostoma sevenfold
문장 생성 92/100 : ghurry subcrureal Geraniaceae premedieval optative quatorzain phanerogamic
문장 생성 93/100 : nonrationalized histogenetically edgemaking incompletable diacetamide miserhood saprophilous darkhearted
문장 생성 94/100 : numskull hepatoduodenal nonconsultative thereacross frankfurter jongleur grateman Myrmeleon chlamys
문장 생성 95/100 : sulfostannide Taurini grumph metastigmate paleoeremology hypopituitarism anticontagious
문장 생성 96/100 : eozoon dragnet multifidly calfbound shamefaced didynamian heavenliness xeromyron brutelike
문장 생성 97/100 : unlevel removable archiblastoma phosphoriferous expectation coxcombical Thondrakians Nectariniidae
문장 생성 98/100 : intentness pileorhize electrothanatosis lightfulness rootwise upridge orometer neurobiotaxis
문장 생성 99/100 : undertint eparcuale barbarousness woodware garnisher vasostomy resew otogenous gooseberry punctal
문장 생성 100/100 : catstone equidivision pantarbe supersquamosal crenology sialadenitis ikra Gratiola unapprehension rucervine
```

<그림 1. 평문 생성 결과(상: 실행 직후, 하: 실행 완료)>

Generation > generated_sentences.txt

```
1   windfirm bacury iddat Tolstoyism radiostereoscopy configurationist rhomborectangular Zoraptera
2   fractuosity pricker appliable Artocarpaceae upflung autolytic overcomplacently gassy
3   Paleozoic arteriology descriptory Urocystis subtotem nodicorn countermutiny sorryhearted
4   polycentric aware brettice Xiphura morosely narrowness xiphoidal spirality braird
5   preadvertise metapodium malady falsework undertapster theomythology coeldership comitatus
6   chessboard Ignatius inaugurative Gleditsia ovigerm uningenuous obsoletism empocket diarrhetic untaut
7   itself meretriciousness vixen anticoagulative pegasoid perceptual nonmicrobic cradleside wolframate tenio
8   diagraphic structuralism unironical aethalioid unsubscribed derivation taxicab entomologize
9   terpeneless emberizine balustrade Hemipodius cud realist nepheline despitefulness Mathurin Panboeotian
10  Demogorgon acrotomous Kluxer anteopercle ancience jellico algraphic fibula unsubsidiary
11  tibiotarsus forejudge gauteite manumisable conscionably aboveproof canaliculi experimentalism anthracemia
12  unpacker nonsubsiding archknave slotwise totemist dali confederatism
13  polypigerous philosophistic appendiculate triangulopyramidal dishboard tergiversatory thither cased
14  diaphane osmogene humbugable happily delayful news Jovinian palas serrirostrate
15  comatulid sulphidize mellonides Pterocletes overget phaseal ventilable pseudopionnotes ataman unconciliating
```

```
89  scrawny fencelet homelet undisintegrated uncurious pinhold kakariki unwastingly pleuroperitonaeal
90  soapbubbly unmet acquiescent arcocentrous harpsichordist superintellectual undistinguishing myophysical disjunct
91  stylebook capewise bush cop transitivity gradation elenchtical ophthalmatrophia Adenostoma sevenfold
92  ghurry subcrureal Geraniaceae premedieval optative quatorzain phanerogamic
93  nonrationalized histogenetically edgemaking incompletable diacetamide miserhood saprophilous darkhearted
94  numskull hepatoduodenal nonconsultative thereacross frankfurter jongleur grateman Myrmeleon chlamys
95  sulfostannide Taurini grumph metastigmate paleoeremology hypopituitarism anticontagious
96  eozoon dragnet multifidly calfbound shamefaced didynamian heavenliness xeromyron brutelike
97  unlevel removable archiblastoma phosphoriferous expectation coxcombical Thondrakians Nectariniidae
98  intentness pileorhize electrothanatosis lightfulness rootwise upridge orometer neurobiotaxis
99  undertint eparcuale barbarousness woodware garnisher vasostomy resew otogenous gooseberry punctal
100 catstone equidivision pantarbe supersquamosal crenology sialadenitis ikra Gratiola unapprehension rucervine
```

<그림 2. 평문 생성 후 txt 저장 결과>

2) 시저 암호 생성 결과

평문을 시저 암호로 생성하기 위해 1~25 정수 중 랜덤 키 값을 불러와 암호화를 진행하였다. 각 문장마다 키 값은 모두 다르며 암호화된 암호문은 텍스트 파일에 저장하였다.

```
평문 : windfirm bacury iddat Tolstoyism radiostereoscopy configurationist rhomborectangular Zoraptera
암호화 Key(shift) : 22
암호화 : sejzbeni xwyqnu ezzwp Pkhopkueoi nwzekopanakoyklu ykjbecqnwpekjeop ndkixknaypwjcqhwn Vknwlpanw
진행도 : 1/100

평문 : fractuosity pricker appliable Artocarpaceae upflung autolytic overcomplacently gassy
암호화 Key(shift) : 5
암호화 : kwfhyztxnyd uwnhpjw fuuqnfgqj Fwythfwufhjfj zukqzsl fzytqdynh tajwhtruqfhjsyqd lfxxd
진행도 : 2/100

평문 : Paleozoic arteriology descriptory Urocystis subtotem nodicorn countermutiny sorryhearted
암호화 Key(shift) : 16
암호화 : Fqbuepeys qhjuhyebewo tuishyfjeho Khesoijyi ikrjejuc detysehd sekdjuhckjydo iehhoxuqhjut
진행도 : 3/100
```

```
평문 : intentness pileorhize electrothanatosis lightfulness rootwise upridge orometer neurobiotaxis
암호화 Key(shift) : 11
암호화 : tyepyeypdd atwpzcstkp pwpneczeslylezdtd wtrseqfwypdd czzehtdp factorp zczxpepc ypfczmtzelitd
진행도 : 98/100

평문 : undertint eparcuale barbarousness woodware garnisher vasostomy resew otogenous gooseberry punctal
암호화 Key(shift) : 10
암호화 : exnobdsxd ozkbmekvo lkblkbyecxocc gyyngkbo qkbxscrob fkcycdywi bocog ydyqoxyec qyycolobbi zexmdkv
진행도 : 99/100

평문 : catstone equidivision pantarbe supersquamosal crenology sialadenitis ikra Gratiola unapprehension rucervine
암호화 Key(shift) : 3
암호화 : fdwvwrqh htxlglylvlrq sdqwdueh vxshuvtxdprvdo fuhqrorjb vldodghqlwlv lnud Judwlrod xqdssuhkhqvlrq uxfhuylqh
진행도 : 100/100
```

<그림 3. 평문을 불러와 시저 암호 생성 결과>

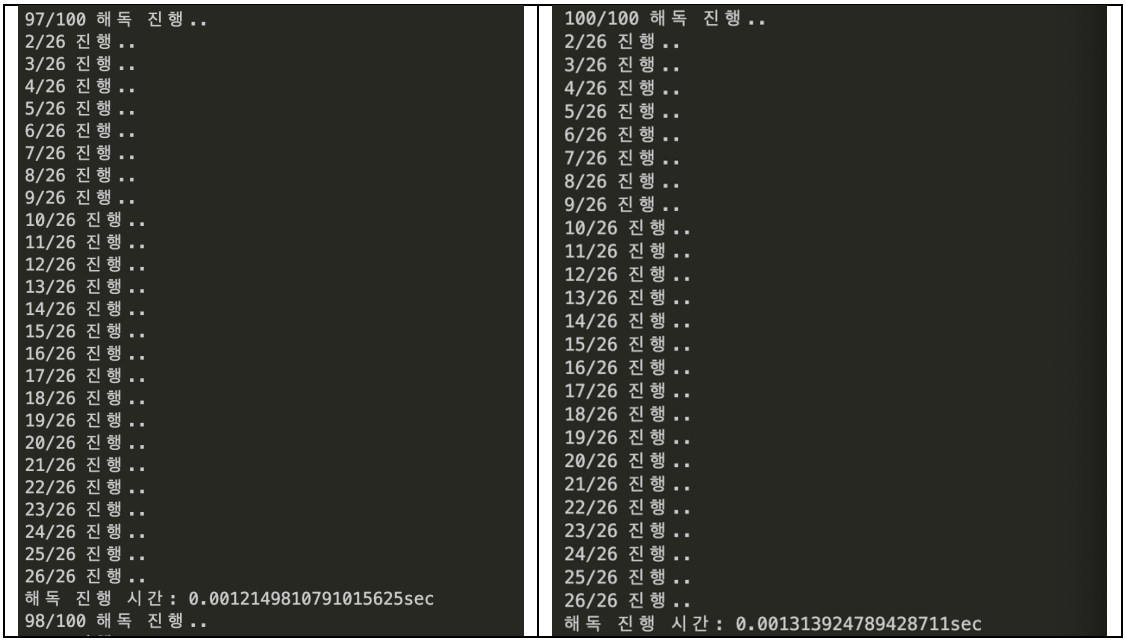
Caesar > [Caesar]encrypted_sentences.txt

```
1  sejzbeni xwyqnu ezzwp Pkhopkueoi nwzekopanakoyklu ykjbecqnwpekjeop ndkixknaypwjcqhwn Vknwlpanw
2  kwfhyztxnyd uwnhpjw fuuqnfgqj Fwythfwufhjfj zukqzsl fzytqdynh tajwhtruqfhjsyqd lfxxd
3  Fqbuepeys qhjuhyebewo tuishyfjeho Khesoijyi ikrjejuc detysehd sekdjuhckjydo iehhoxuqhjut
4  rqnaegpvtke cyctg dtgvvkeg Zkrjwtc oqtqugna pcttqypguu zkrjqkfcn urktcnkva dtcktf
5  qsfbewfsujtf nfubqpejvn nbmbez gbmtfxpsl voefsubqtufs uifpnzuipmphz dpfmefstijq dpnjubuvt
6  rwthhqdpgs Xvcpixjh xcpjvjgpixkt Vatsxihxp dkxvtgb jcxcvtcjdjh dqhdatixhb tbedrzti sxpggwtixr jcipji
7  nyxjqk rjwjywnhntzxsjxx ancjs fsynhtflzqfynaj ujlfxtni ujwhjuyzfq stsrnhwtgnh hwfiqjxnij btqkwfrfyj yjsnt
8  fkcitcrjke uvtwevwtcnkuo wpktqpkecn cgvjcnkqkf wpuwduetkdgf fgtkxcvkqp vczkecd gpvqoqnqikbg
```

<그림 4. 생성된 시저 암호를 텍스트 파일에 저장한 결과>

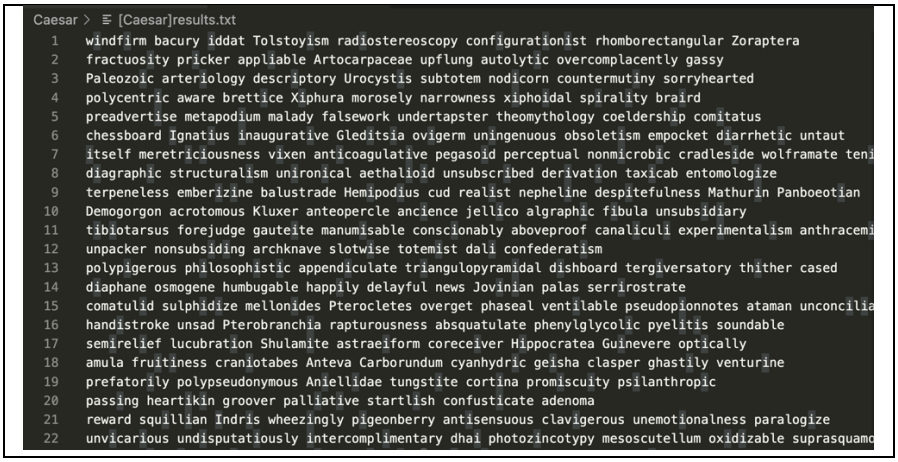
3) 시저 암호 해독 결과

실습1에서 암호화를 진행할 때 암호에 사용된 키 값을 따로 저장하지 않았기 때문에 암호를 해독하기 위해서는 키 값을 1~25까지 모두 적용하여 해독을 진행하였다. 이때, 해독 결과를 가져오기 위해서 후보군을 선정해야 하는데, 선정 기준으로 nltk 라이브러리에 words 단어 데이터베이스와 일치한지 비교하여 해독한 문장의 후보군 중 가장 일치한 단어가 많은 경우 해독 결과로 선정하도록 구현하여 결과를 확인하였다. 또한 효율적으로 코드를 구현하였는지 확인하고자 각 문장별로 해독한 시간을 기록하였고, 마지막에 각 문장별 해독 소요 시간과 평균 시간을 구해 출력하여 확인하였다.



<그림 5. 시저 암호 해독 진행 화면>

<그림 6. 시저 암호 해독 시간 및 전체 해독 평균 시간>



<그림 7. 시저 암호 해독 후 텍스트 파일 저장 결과>

<그림 7>과 같이 해독 결과의 텍스트 파일과 <그림 2>의 평문 생성 결과 텍스트 파일의 유사도를 판단하여 해독 코드의 정확도를 3.2 고찰에서 작성한다.

부가 실습: 전치 암호 생성 및 암호 해독 실습

1) 전치 암호화 결과

평문을 전치 암호로 생성하기 위해 평문을 행렬(2차원 배열)로 변환하였고, 암호키를 랜덤으로 생성하여 전치 암호의 절차에 맞게 생성하여 텍스트 파일에 저장하였다.

```
진행도 : 1/100
암호키(랜덤 생성) : [6, 5, 8, 1, 3, 2, 7, 0, 4]
암호화 이전 : windfirm bacury iddat Tolstoyism radiostereoscopy configurationist rhomborectangular

암호화 결과 : rdoooutcZdiiticgserabaityaharuicTmroimgpdrlaofnolenuorenobutmdysprrtoawatsectonafysds

진행도 : 2/100
암호키(랜덤 생성) : [3, 5, 6, 0, 7, 2, 4, 8, 1]
암호화 이전 : fractuosity pricker appliable Artocarpaceae upflung autolytic overcomplacently gassy

암호화 결과 : crloeuvcsucaaporncokbrflctdftaAcnipgselplyolkapptaaoastiicuteeyireautmycrypregcla

진행도 : 3/100
암호키(랜덤 생성) : [7, 6, 1, 4, 2, 3, 0, 8, 5]
암호화 이전 : Paleozoic arteriology descriptory Urocystis subtotem nodicorn countermutiny sorryhea

암호화 결과 : ilictceoeoorobitstaryotenuyorsUsounaltdrimctheeeysnoiePagtstrmrcopyoorrdzicrudnyr

진행도 : 4/100
암호키(랜덤 생성) : [5, 7, 0, 6, 4, 1, 3, 2, 8]
암호화 이전 : polycentric aware brettice Xiphura morosely narrowness xiphoidal spirality braird

암호화 결과 : ereoriprotbioohricpiehenildneXrrpiaccacmaxsbgoctuledivywianslyflatrysatsrrpswoaru
```

```
진행도 : 97/100
암호키(랜덤 생성) : [8, 0, 3, 4, 9, 1, 6, 2, 7, 5]
암호화 이전 : unlevel removable archiblastoma phosphoriferous expectation coxcombical Thondrakians Nectariniida

암호화 결과 : ecafccoNdtuoihramdceebapsockrzvlshenaiiwmhpetoneaunvbootbrtplaorpoTniklalsuiiaaurrmiexhsifeetoxcl

진행도 : 98/100
암호키(랜덤 생성) : [7, 2, 3, 5, 4, 0, 8, 6, 1]
암호화 이전 : intentness pileorhize electrothanatosis lightfulness rootwise upridge orometer neurobiotaxis

암호화 결과 : ehrsutdeobtieaiseruiellngsuorstocttorebaneeahrpmosisztsnienasioilwgrtinrtofoitijnpehlesoex

진행도 : 99/100
암호키(랜덤 생성) : [2, 5, 6, 1, 4, 7, 0, 3, 8]
암호화 이전 : undertint eparcuale barbarousness woodware garnisher vasostomy resew otogenous gooseberry punctal

암호화 결과 : darserygsnturorssoeaiaoonoeurlnpaeremoourcawaaenbtnludiswsrjuebnahotoperbsgvreecteswstogyq

진행도 : 100/100
암호키(랜덤 생성) : [4, 8, 5, 6, 2, 9, 7, 0, 1, 3]
암호화 이전 : catstone equidivision pantarbe supersquamosal crenology sialadenitis ikra Gratiola unapprehension
암호화 결과 : tvaqelkanveosoonrpnuoirunarusinsbaodanintdnrcisoheqnusgiaprzeiemleGaoecuppayttruaiaelsiiecsitsrai
```

<그림 8. 전치 암호화 실행 출력 결과>

Transposition_Cipher > ☰ [Trans]encrypted_sentences.txt

```
1  rdoooutcZdiiticgserabaityaharuicTmroimgpdrlaofnolenuorenobutmdysprrtoawatsectonafysdsiirar
2  crloeuvcsucaaporncokbrflctdftaAcnipgselplyolkapptaaoastiicuteeyireautmycrypregcla
3  ilictceoeoorobitstaryotenuyorsUsounaltdrimctheeeysnoiePagtstrmrcopyoorrdzicrudnyr
4  ereoriprotbioohricpiehenildneXrrpiaccacmaxsbgoctuledivywianslyflatrysatsrrpswoaru
5  todrpmciuhrsifuttecopidyksyopsdemsetleiuvtaerhorttealwtegsadeeuanehlolrpaoaoyhtxammldrodmt
6  hgulielcecrivaibmrarstriroitcfbiatmusdunsaudeuteinasiinoeatvoutsusminpdneonspruocIaGvgoohtengegnektm
7  sivoprnroetrsceeocwteetidlidtfeciaecmalnmunlstreaqesaaouosmaitsivpnceelixggeidfifoeuapclrornntiabiaz
8  dcaohsdoeiptnadiacluhuieubtaogatiilbetteaculirvioqrumaocixmkislnaudnnzirrtneibgjgrscisraoa
9  lzrdlifhenrmaeuessPneebHcneenapblmdppsaksnduselrtzeeuirhiMnmeiaiinuuoyeitoaletodnrspeetabstseestdnii
10 gmrccafubroernocsyotxeecinrgoupiihuaooalelibjDnunejgbsmcKenlaldeastaeruiorloclpai
11 boumsbaxlctjiaoelrmiautbnpiiaaifaunaceaairticonpierdelarcmnwsgmebouetjoeesivaesmueaclolnhytsgnoyfitr
12 kbrstdemcuaeetfspnnaiiotasgvssnirihotlekundkteirescloadanoinwmca
13 yhtcniartbgoaallttropiiugdrshuropepirrakosidiabvhdesptodeocilpsiamoeipopetysgysbpuhnrrhiteilcluadaey
14 pobalwaerammheeistdohlyliaoisuednnssaegpyJprteebluvlrpnnaifoaiehgupasnra
15 smPsaloanklidetnetnimplrealomludileesoutcuetosaptcollevebiaiizetptuecnahoorlenaitincgvpnnadesehidsog
16 tdnuaagyukisrpsuycsphkeiuqpcianuornaelilrPcrbtlenmothoseyldzsaatsllpoodnbaetniseaerasuhotb
17 iaaariaeyelotfeoitsfiiicpuplrlroHerlruhsmeavaebutcrteletmeepGouicSarvrecmuneoicni
18 rceamilsupiaAbygsiizunoeuhsrvqfsbCurcatrletvnyhgetatnnoaeplnursrccatrqasaaddahncmiitrnieye
19 epoiuomsprydntcopotymlsicabiesatpthupluAeeryralylgtslcrsudiaiteopoitnuncfonenriii
```

<그림 9. 생성된 전치 암호문의 텍스트 파일 저장 결과>

2) 전치 암호문 해독 결과

암호화에 사용된 키 값을 가져와 해당 키를 적용하여 암호화의 반대로 적용하여 복호화를 구현하도록 시도했다.

```
해독 진행 : 1/100
적용 키 값 : [6, 5, 8, 1, 3, 2, 7, 0, 4]
해독 결과 : csaraodwaaotaiplbrcsuctTrntooioiicdeuttiZehooryafrdiarmnnpssrdraifestytgymlumanroibugoored
해독 진행 : 2/100
적용 키 값 : [3, 5, 6, 0, 7, 2, 4, 8, 1]
해독 결과 : oarAptcarscdgksycauolnyattgcuofsatirvrciooemcrcktepirylabalpiepepfclauuecntplaeyl
해독 진행 : 3/100
적용 키 값 : [7, 6, 1, 4, 2, 3, 0, 8, 5]
해독 결과 : etnucorrnirysdegpctbtsissoucieomntodcooUrytyretyaherdrlorrteaoiioaolePczosuntimry
해독 진행 : 4/100
적용 키 값 : [5, 7, 0, 6, 4, 1, 3, 2, 8]
해독 결과 : eieXctitpihnpxessorilabiytrpriisdlaaroeralnywetpncoylroohrmuarsrbieacwarocdcgvfsu
해독 진행 : 5/100
적용 키 값 : [8, 1, 0, 4, 5, 6, 2, 7, 3]
해독 결과 : dspsirgealoropeoenomctksaaeltdueytrwupatriidutshodpfdevtalyrmuymtedohoitseelerxmthcolhtaam
해독 진행 : 6/100
적용 키 값 : [1, 8, 4, 5, 3, 7, 6, 9, 0, 2]
해독 결과 : eacuitnuhthrsbsaodcecrfnnvpotmiirmenunvguvrauiteaglaitdisoGelmistempoeebouuossgngitiasunIncrtdeairok
해독 진행 : 7/100
적용 키 값 : [2, 1, 9, 3, 6, 7, 0, 4, 5, 8]
해독 결과 : noimronicbirecustionsteemeilfrocialaigutrcdaescdliowtlamefrapedesovgairelctupepaetfnqaeiozvstinasxen
해독 진행 : 8/100
적용 키 값 : [0, 6, 7, 2, 5, 4, 1, 8, 3]
해독 결과 : dilaeronrrdiuiuuizgaonuaeoxdeiectbialrgasdeicrknbrhaitaqmniscpuotvcatsathgtiiuncoabllmsijo
해독 진행 : 9/100
적용 키 값 : [7, 2, 1, 3, 9, 4, 8, 6, 5, 0]
해독 결과 : nnakzmydsirabldUatsezmebneiirsdeHmuiiopeienpehnletePnatnoobihsesrMutanlrepseeentfsepliuetdlucdsriaes
해독 진행 : 10/100
적용 키 값 : [7, 6, 5, 4, 8, 0, 2, 1, 3]
해독 결과 : aociljlelusnubbluafcihigarpbyrajsdiicreplnetorexuauKslmotooncargrogoDmeocneieenac
적용 키 값 : [2, 7, 5, 0, 6, 3, 1, 4, 8]
해독 결과 : ulueelrvidehdbatottoztimancoyiaisflojboseeaehlnodkdtntnynnemaacgdnuxleerfesnmfdfrriydnimaj
해독 진행 : 97/100
적용 키 값 : [8, 0, 3, 4, 9, 1, 6, 2, 7, 5]
해독 결과 : coblhvaarecaonotoixcNdkiernasaaiaspbolmtfhpheorsiotezwupkufjomcanbTihldcriatiaineuevmnllrecrsetopuex
해독 진행 : 98/100
적용 키 값 : [7, 2, 3, 5, 4, 0, 8, 6, 1]
해독 결과 : tinceiattsoruohtornxhtirepnloeriesbmiwfhualonsnrieseltaoegoleestazinjeebuorrsitpdsgtessiio
해독 진행 : 99/100
적용 키 값 : [2, 5, 6, 1, 4, 7, 0, 3, 8]
해독 결과 : ysnealupegsranubsovsdneumainbegsorwueecyeooprtrtrtataroesassruilonwhgwrroacnjoeosoeeadbrtq
해독 진행 : 100/100
적용 키 값 : [4, 8, 5, 6, 2, 9, 7, 0, 1, 3]
해독 결과 : apunopatilnnsihroreeeonocglalrvuinezeucrteontqecasasrbnuepatqouarsmpeskrrasaGtiivoisdniuiilnadiieysa
```

<그림 10. 전치 암호문의 해독 진행 화면>

Transposition_Cipher > [Trans]Decrypted_sentences.txt

```
1  csaraodwaaotaiplbrcsuctTrntooioiicdeuttiZehooryafrdiarmnnpssrdraifestytgymlumanroibugoored
2  oarAptcarscdgksycauolnyattgcuofsatirvrciooemcrcktepirylabalpiepepfclauuecntplaeyl
3  etnucorrnirysdegpctbtsissoucieomntodcooUrytyretyaherdrlorrteaoiioaolePczosuntimry
4  eieXctitpihnpxessorilabiytrpriisdlaaroeralnywetpncoylroohrmuarsrbieacwarocdcgvfsu
5  dspsirgealoropeoenomctksaaeltdueytrwupatriidutshodpfdevtalyrmuymtedohoitseelerxmthcolhtaam
6  eacuitnuhthrsbsaodcecrfnnvpotmiirmenunvguvrauiteaglaitdisoGelmistempoeebouuossgngitiasunIncrtdeairok
7  noimronicbirecustionsteemeilfrocialaigutrcdaescdliowtlamefrapedesovgairelctupepaetfnqaeiozvstinasxen
8  dilaeronrrdiuiuuizgaonuaeoxdeiectbialrgasdeicrknbrhaitaqmniscpuotvcatsathgtiiuncoabllmsijo
9  nnakzmydsirabldUatsezmebneiirsdeHmuiiopeienpehnletePnatnoobihsesrMutanlrepseeentfsepliuetdlucdsriaes
10 aociljlelusnubbluafcihigarpbyrajsdiicreplnetorexuauKslmotooncargrogoDmeocneieenac
11 ciaaewjmyrmabuilescnbtaiirsoutsonncabiloxriepmeeniojufrdgeesuitatemeagbepaorovoyalicncualflmaaintsht
12 dtisleammstikraaranghdciseavoklnbunsinsokcparuenefoneidcteisttow
13 trhahcistenlgpioayruatrrbeoghdygproelopitaipipsehcboukdipbeyhoiossppulildiadmsracauedtitnlrtsrvoeyia
14 bmdospvfpamoisrloaoatanJuiuprsinylageildpbnhalhhsatrasaeyegenerwelueepin
15 ateeovprghlnaesetlviPdlieleonsatoopiennsnnmutacanoserltetocemipdulzhieoelsabuepdkiltciniagslmucoiatd
16 nshaieyleaucpaboanaburknlnlltsyyqnrhadetdipillesarguurctsosotkscetstbeapeurmdpnuasioPozoih
17 aoilhumSnalfrubtcuytplalucieiurveoenrfiomceroiecHerpviatirsteaeieslreeimaopeatGrc
```

<그림 11. 전치 암호의 해독 결과 저장 파일>

전치 암호를 해독하는 과정의 정확도 및 고려 사항에 대해서는 3.2 고찰에서 다루었다.

3.2 고찰(Discussion)

1) 시저 암호 해독 정확도 분석

시저 암호의 해독 결과를 분석하기 위해서 암호화 하기 이전의 평문 텍스트 파일과 암호를 해독한 결과가 저장된 텍스트 파일을 비교하여 정확도를 파악해보았다.

```
matching = 0
with open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt', 'r') as file1, \
    open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]results.txt', 'r') as file2:
    for line1, line2 in zip(file1, file2):
        if line1.strip() == line2.strip():
            print("Matching line: ", line1.strip())
            matching += 1
        else:
            print("Non-matching line: ")
            print("File1: ", line1.strip())
            print("File2: ", line2.strip())

    print(f'해독 성공률 : {matching/100*100}%')
```

<코드 7. 시저 암호의 해독 정확도 분석 코드>

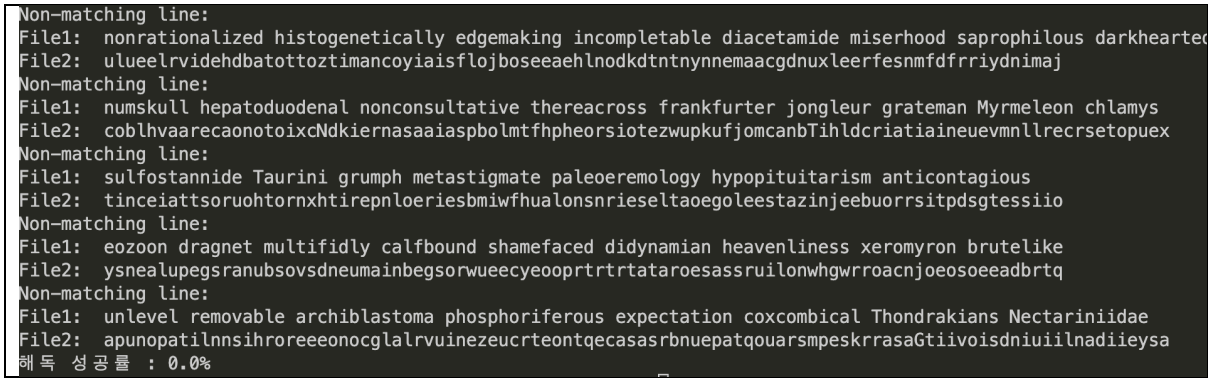
```
97번 문장 Matching line:  unlevel removable archiblastoma phosphoriferous expectation coxcombical Thondrakia
ns Nectariniidae
98번 문장 Matching line:  intentness pileorhize electrothanatosis lightfulness rootwise upridge orometer neu
robiotaxis
99번 문장 Matching line:  undertint eparcuale barbarousness woodware garnisher vasostomy resew otogenous goo
seberry punctal
100번 문장 Matching line:  catstone equidivision pantarbe supersquamosal crenology sialadenitis ikra Gratiol
a unapprehension rucervine
해독 성공률 : 100.0%
```

<그림 12. 매칭 결과 및 성공률 출력 결과>

평문과 해독한 문장이 일치할 경우 “Maching line”으로 결과를 출력하였고, 만약 매칭되지 않을 경우, 평문과 해독한 결과의 문장을 비교할 수 있게 출력하도록 구현하였다. 또한, 매칭된 문장의 개수를 구해 100으로 나누고 확률로 나타내기 위해 100을 곱하여 성공률을 확인하였다. Nltk의 단어 데이터베이스를 기반으로 후보군을 선정하여 해독 결과를 저장하여 모든 시저암호의 해독 결과를 평문과 똑같이 100%로 찾아낼 수 있었다는 것을 알 수 있다. 만약 다른 word 데이터베이스를 사용하여 후보군을 선정할 경우, 결과가 달랐을 것으로 예상한다.

2) 전치 암호 해독 정확도 분석

시저 암호와 마찬가지로 전치 암호의 해독 결과를 분석하기 위해 <코드 7>과 유사하게 구현하여 정확도를 파악하였다.



<그림 13. 해독 결과 및 성공률 출력 결과>

단 하나의 암호문도 해독하지 못했다. 암호화에 사용된 키 값을 불러와 적용했음에도 불구하고 평문과 유사한 형태로 해독하지 못한 이유는 암호문을 행렬(2차원 배열)로 다시 생성하고 암호화 키 값을 적용해야 하는데, 그렇게 될 경우 암호 해독 시간이 매우 길어질 것으로 예상한다. 암호화 키 값도 모든 경우를 대입해야 하고, 행렬의 크기 또한 모든 경우의 수를 생성해야 하게 된다고 판단했다. 따라서 전치 암호의 해독 코드의 핵심은 암호 키의 경우의 수와 행렬 크기의 경우의 수를 모두 계산하여 구현해야 된다고 생각한다.

3.3 느낀 점

전통적인 암호화 방식에 대해 배울 수 있었으며, 프로그래밍을 통해 직접 구현하여 암호화 방식의 알고리즘에 대해 고민하고 접근할 수 있었다. 시저 암호에 대한 과제를 수행하면서 파이썬 라이브러리 중 nltk를 사용하여 쉽게 평문을 생성할 수 있었고, 또한 이를 이용하여 해독하는 과정에서 해독 결과 후보군을 선정하여 최종 해독문을 선정하는데 많은 도움이 되었던 것 같다. 만약 nltk의 단어 데이터베이스를 사용하지 않았을 경우, 후보군 선정하는데 있어서 데이터베이스를 추려내는데 오랜 시간이 걸렸을 것으로 예상되었다. 또한, 전치 암호에 대한 실습과제를 수행하면서 전치 암호의 암호화 방식을 이해하는 데 비교적 오래 걸렸던 것 같다. 이해하고 이를 또 알고리즘으로 설계하는 데 있어서 많은 고민이 필요했다. 평문을 행렬로 변환할 때 행과 열의 크기를 어떻게 설정할 지, 또 해독을 수행하기 위해 반대로 설계를 해야 한다는 점에서도 변수 선언부터 반복문, 변수 계산 등 여러 오차가 많이 있었다. 아쉬웠던 점은 전치 암호를 해독하는 과정에 있어서 행렬 크기를 설정하는 방법에 대한 접근이 되지 않아 해독률 0%라는 결과를 확인하여 아쉬웠다. 앞으로 설계할 때에 있어서 신중하게 고려해서 접근하도록 해야겠다.

4 APPENDIX

4.1 소스코드

〈소스코드1. generate_sentences.py〉

```
#단어 데이터베이스 사용을 위한 라이브러리
import nltk
import random
#로컬에 nltk 에서 제공하는 words 데이터베이스를 다운받는다.
nltk.download('/Users/kangdonghee/nltk_data')
nltk.download('words')
# 100 개의 문장 생성
num_sentences = 100
sentences = []
for i in range(num_sentences):
    # 7~10 단어를 무작위로 조합하여 문장 생성
    num_words = random.randint(7, 10)
    words = [random.choice(nltk.corpus.words.words()) for _ in range(num_words)]
    sentence = ' '.join(words)
    sentences.append(sentence)
# 파일에 저장
with open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/generated_sentences.txt', 'w') as file:
    file.write('\n'.join(sentences))
```

〈소스코드2. Caesar_Cipher.py〉

```
import random
def caesar_cipher(text, key):
    #반환할 암호문 선언
    shifted_text = ""
    for char in text:
        if char.isalpha():
            #시저암호의 수식 이용하여 암호화 진행
            if char.isupper(): #아스키코드값을 이용해 대문자일 경우 shift 진행
                shifted_char = chr((ord(char) - 65 + key) % 26 + 65)
            else: #소문자에 대해서도 shift 진행
                shifted_char = chr((ord(char) - 97 + key) % 26 + 97)
        else:
            shifted_char = char
        shifted_text += shifted_char
    print(f'평문 : {text}')
    print(f'암호화 Key(shift) : {key}')
    print(f'암호화 : {shifted_text}')
    return shifted_text
# generate_sentences.txt 파일에서 100 개의 문장을 가져온다.
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt", "r") as f:
    sentences = f.readlines()
# 각 문장을 caesar_cipher(text,key)를 적용하여 암호화.
encrypted_sentences = []
count = 0
```

```
for sentence in sentences:
    # 1 에서 25 사이의 랜덤한 정수로 key 값을 생성한다. (각 문장마다 랜덤한 키값으로 암호화 진행)
    key = random.randint(1, 25)
    encrypted_sentence = caesar_cipher(sentence.strip(), key)
    encrypted_sentences.append(encrypted_sentence)
    count +=1
    print(f'진행도 : {count}/100')
    print()
# 암호화된 100 개의 문장을 파일에 저장.
with open("/Users/kangdonghee/Desktop/Computer Security/실습/week3/Caesar/[Caesar]encrypted_sentences.txt", "w") as f:
    for sentence in encrypted_sentences:
        f.write(sentence + "\n")
```

〈소스코드3.Caesar_Decipher.py〉

```
import nltk
import time
nltk.download('words')
from nltk.corpus import words
word_list = set(words.words())
# 암호해독 함수
decrypt_time = []
def decrypt(text):
    start = time.time()
    candidates = []
    for key in range(1, 26):
        print(f'{key+1}/26 진행..')
        decrypted = ""
        for char in text:
            if char.isalpha():
                char_code = ord(char)
                char_code -= key
                if char.isupper():
                    if char_code < ord('A'):
                        char_code += 26
                    elif char_code > ord('Z'):
                        char_code -= 26
                else:
                    if char_code < ord('a'):
                        char_code += 26
                    elif char_code > ord('z'):
                        char_code -= 26
                decrypted += chr(char_code)
            else:
                decrypted += char
        # 후보군 리스트에 저장
        if any(word in word_list for word in decrypted.split()):
            candidates.append(decrypted)
    end = time.time()
    print(f'해독 진행 시간: {end - start}sec')
    decrypt_time.append(end-start)
    return candidates
# 해독 결과를 저장할 파일
```

```
output_file = open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]results.txt", "w")
# 암호문이 저장된 파일 불러오기
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]encrypted_sentences.txt", "r") as input_file:
    # 각각의 암호문 해독하기
    count = 0
    for line in input_file:
        print(f'{count + 1}/100 해독 진행..')
        encrypted = line.strip()
        # 모든 암호화 키 시도
        candidates = decrypt(encrypted)
        count += 1
        # 후보군 리스트에 저장된 문장들 중 일치하는 단어가 많은 문장 선택
        max_match = 0
        selected_candidate = ''
        for candidate in candidates:
            match_count = sum([word in word_list for word in candidate.split()])
            if match_count > max_match:
                max_match = match_count
                selected_candidate = candidate
        # words 와 일치하는 단어가 많은 경우에만 파일에 입력
        if max_match > 0:
            output_file.write(selected_candidate + "\n")
output_file.close()
line_count = 0
total_time = 0
for i in decrypt_time:
    print(f'{line_count+1}번 라인 해독 시간 : {i:.4f}')
    line_count += 1
    total_time += i
print(f'평균 해독 시간 : {total_time/100:.4f} 초')
```

〈소스코드4. Caesar_compare.py〉

```
matching = 0
with open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt', 'r') as file1, \
    open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Caesar/[Caesar]results.txt', 'r') as file2:
    line = 0
    for line1, line2 in zip(file1, file2):
        if line1.strip() == line2.strip():
            print(f"{line+1}번 문장 Matching line: ", line1.strip())
            matching += 1
            line += 1
        else:
            print("Non-matching line: ")
            print("File1: ", line1.strip())
            print("File2: ", line2.strip())
            line += 1

    print(f'해독 성공률 : {matching/100*100}%')
```

<소스코드5. Transposition_Cipher.py>

```
import math
import random
import string
def add_padding(text):
    #띄어쓰기를 포함할 경우 전치암호화하는 과정에서 줄바꿈이 발생하는 문제점 해결
    text = ''.join(text.split())
    text_len = len(text)
    # row, col 계산
    row = math.ceil(text_len**0.5)
    col = math.ceil(text_len/row)
    # padding 에 추가할 문자 계산
    pad_len = row * col - text_len
    pad = ''.join(random.choices(string.ascii_lowercase, k=pad_len))
    # padding 추가
    padded_text = text + pad
    # 이차원 배열에 넣기
    matrix = [list(padded_text[i:i+col]) for i in range(0, row*col, col)]
    return matrix
def transposition_cipher(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    # 열을 랜덤한 순서로 선택하기 위해 인덱스 리스트를 생성
    key = list(range(num_cols))
    random.shuffle(key)
    print(f'암호키(랜덤 생성) : {key}')
    # 랜덤한 순서로 열을 선택하여 1 차원 리스트로 저장
    result = []
    for col_idx in key:
        for row_idx in range(num_rows):
            result.append(matrix[row_idx][col_idx])

    # 1 차원 리스트를 문자열로 변환하여 반환
    return ''.join(result)
#평문을 불러와 암호화 진행
#---------------------------------
# generate_sentences.txt 파일에서 100 개의 문장을 가져온다.
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt", "r") as f:
    sentences = f.readlines()
count = 0
encrypted_sentences = []
for sentence in sentences:
    print(f'진행도 : {count+1}/100')
    matrix_text = add_padding(sentence)
    encrypted_sentence = transposition_cipher(matrix_text)
    encrypted_sentences.append(encrypted_sentence)
    count +=1
    print(f'암호화 이전 : {sentence}')
    print(f'암호화 결과 : {encrypted_sentence}')
    print()
# 암호화된 100 개의 문장을 파일에 저장.
```

```
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Transposition_Cipher/[Trans]encrypted_sentences.txt", "w") as f:
    f.write('\n'.join(encrypted_sentences))
```

〈소스코드6. Transposition_Decipher.py〉

```
import math
def decrypt_transposition_cipher(ciphertext, key, count):
    key = [int(char) for char in key]
    num_rows = math.ceil(len(ciphertext)/len(key))
    num_cols = len(key)
    print(f'해독 진행 : {count+1}/100')
    print(f'적용 키 값 : {key}')
    # 인덱스 리스트를 생성하여 역순으로 정렬
    try:
        key_inv = [key.index(i) for i in range(num_cols)]
    except ValueError:
        return None
    # 1 차원 리스트를 이차원 배열에 복원
    matrix = [list(ciphertext[i:i+num_cols]) for i in range(0, num_rows*num_cols,
num_cols)]
    # 복호화된 결과를 저장할 리스트
    result = []
    # 역순으로 정렬된 인덱스를 사용하여 원래의 순서로 문자열을 복원
    for col_idx in key_inv:
        for row_idx in range(num_rows):
            result.append(matrix[row_idx][col_idx])
    # 복호화된 결과를 문자열로 변환하여 반환
    final = ''.join(result)
    print(f'해독 결과 : {final}')
    return final
# 암호화된 문장을 불러온다.
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Transposition_Cipher/[Trans]encrypted_sentences.txt", "r") as f:
    encrypted_sentences = f.readlines()
# 암호화된 문장을 복호화
# 저장된 키 값을 불러와 리스트로 변환
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Transposition_Cipher/[Trans]key.txt", "r") as f:
    key_str = f.read()
key_list = key_str.split()
key_num = 0
decrypted_sentences = []
count = 0
for encrypted_sentence in encrypted_sentences:
    decrypted_sentence = decrypt_transposition_cipher(encrypted_sentence.strip(),
key_list[key_num], count)
    if decrypted_sentence:
        decrypted_sentences.append(decrypted_sentence)
    key_num += 1
    count += 1
# 복호화된 문장을 파일에 저장한다.
```

```
with open("/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Transposition_Cipher/[Trans]decrypted_sentences.txt", "w") as f:
    f.write('\n'.join(decrypted_sentences))
```

〈소스코드7. Transposition_compare.py〉

```
matching = 0
with open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Generation/generated_sentences.txt', 'r') as file1, \
    open('/Users/kangdonghee/Desktop/Computer
Security/실습/week3/Transposition_Cipher/[Trans]Decrypted_sentences.txt', 'r') as file2:
    line = 0
    for line1, line2 in zip(file1, file2):
        line1_stripped = line1.replace(" ", "").replace("\n", "")
        line2_stripped = line2.strip()
        if line1_stripped == line2_stripped:
            print(f"{line+1}번 문장 Matching line: ", line1.strip())
            matching += 1
            line += 1
        else:
            print("Non-matching line: ")
            print("File1: ", line1.strip())
            print("File2: ", line2.strip())
            line += 1

    print(f'해독 성공률 : {matching/100*100}%')
```