Lab 2

Part I: Substitution Cipher

You are provided with a passage that is encrypted with a substitution cipher. You only know a few things about it:

1. It is in "normal" English.
2. Spaces (" ") are preserved (the words are intact).
3. Punctuation may not be preserved.
4. It may consist of any characters included the string.printable set
5. You will recognise it when it is decrypted correctly.

The cipher text is provided in this folder (story_cipher.txt).

Clues:

- Wikipedia: Frequency analysis
- Hints for Frequency Analysis
- The frequency of the letters of the alphabet in English Dictionary
- SAS: The frequency of letters in an English corpus

Practical hints:

- You can use Python's string replace function.
- If you are stuck halfway, visually inspect your current cipher, and see if you recognise any words that are only partially decrypted.
- Make sure you keep track of the replacements to ensure you do not "double replace". All the characters in the cipher are upper-case by design to make it easier for you. You can gradually replace them with your hypothesis of the correct lower-case characters and visually inspect the result.

Write a Python script to decrypt the cipher text, and submit it together with your decrypted plain text.

```
import argparse
import string
from collections import Counter
import re
import nltk
from nltk.corpus import words
import itertools
from tabulate import tabulate

# Download the word list if not already downloaded
try:
    nltk.data.find('corpora/words')
except LookupError:
    nltk.download('words')

def load_text(filename):
    """
    Safely reads the input file without modifying it.
    Returns the content in uppercase for analysis.
    """
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            return file.read().upper()
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        exit(1)
    except PermissionError:
        print(f"Error: Permission denied to read file '{filename}'.")
        exit(1)

def get_letter_frequency(text):   # Count only letters
    letters = re.findall(r'[A-Z]', text)
    return Counter(letters)

def get_english_frequency():
    return {
        'E': 12.70, 'T': 9.10, 'A': 8.20, 'O': 7.50, 'I': 6.97,
        'N': 6.75, 'S': 6.33, 'H': 6.09, 'R': 5.99, 'D': 4.25,
        'L': 4.03, 'C': 2.78, 'U': 2.76, 'M': 2.41, 'W': 2.36,
        'F': 2.23, 'G': 2.02, 'Y': 1.97, 'P': 1.93, 'B': 1.49,
        'V': 0.98, 'K': 0.77, 'J': 0.15, 'X': 0.15, 'Q': 0.10,
        'Z': 0.07
    }

def get_english_words():
    return set(word.upper() for word in words.words())

def create_initial_mapping(text_freq, english_freq): # Sort both frequency distributions
    text_letters = sorted(text_freq.items(), key=lambda x: x[1], reverse=True)
    english_letters = sorted(english_freq.items(), key=lambda x: x[1], reverse=True)
    mapping = {}
    for (encrypted, _), (decrypted, _) in zip(text_letters, english_letters):
        mapping[encrypted] = decrypted
    return mapping

def decrypt_text(text, mapping):
    result = []
    for char in text:
        if char in string.ascii_uppercase:
            result.append(mapping.get(char, char))
        else:
            result.append(char)
    return ''.join(result)

def get_words(text):   # Split text into words, preserving punctuation
    return re.findall(r'\b[A-Z]+\b', text)
```

```
def score_decryption(decrypted_text, english_words):
    words = get_words(decrypted_text)
    if not words:
        return 0   # Count how many words are valid English words
    valid_words = sum(1 for word in words if word in english_words)
    return valid_words / len(words)

def find_best_mapping(text, initial_mapping, english_words, max_iterations=100):
    best_mapping = initial_mapping.copy()
    best_score = score_decryption(decrypt_text(text, best_mapping), english_words)

    for _ in range(max_iterations):
        current_mapping = best_mapping.copy()
        improved = False
        for letter1, letter2 in itertools.combinations(string.ascii_uppercase, 2):
            if letter1 in current_mapping and letter2 in current_mapping:
                current_mapping[letter1], current_mapping[letter2] = current_mapping[letter2], current_mapping[letter1]
                current_score = score_decryption(decrypt_text(text, current_mapping), english_words)
                if current_score > best_score:
                    best_score = current_score
                    best_mapping = current_mapping.copy()
                    improved = True
                else:
                    current_mapping[letter1], current_mapping[letter2] = current_mapping[letter2], current_mapping[letter1]
        if not improved:
            break
    return best_mapping, best_score

def save_solution(decrypted_text, filename='solution.txt'):
    """
    Saves the decrypted text to a file.
    """
    try:
        with open(filename, 'w', encoding='utf-8') as file:
            file.write(decrypted_text)
        print(f"\nDecrypted text has been saved to {filename}")
    except PermissionError:
        print(f"Error: Permission denied to write to file '{filename}'.")
    except Exception as e:
        print(f"Error saving solution: {str(e)}")

def main():
    parser = argparse.ArgumentParser(description='Automatic frequency analysis decryption (read-only)')
    parser.add_argument('-i', '--input', required=True, help='Input file containing encrypted text (will not be modified)')
    args = parser.parse_args()

    # Load and process the text (read-only operation)
    encrypted_text = load_text(args.input)
    text_freq = get_letter_frequency(encrypted_text)
    english_freq = get_english_frequency()
    english_words = get_english_words()

    # Create initial mapping
    initial_mapping = create_initial_mapping(text_freq, english_freq)

    # Find best mapping
    best_mapping, best_score = find_best_mapping(encrypted_text, initial_mapping, english_words)

    # Decrypt text with best mapping
    decrypted_text = decrypt_text(encrypted_text, best_mapping)

    # Prepare table data
    table_data = []
    for letter in string.ascii_uppercase:
        if letter in text_freq:
            table_data.append([
                letter,  # Encrypted letter
                text_freq[letter],  # Frequency in encrypted text
                f"{text_freq[letter]/sum(text_freq.values())*100:.2f}%",  # Percentage
                best_mapping.get(letter, '-'),  # Mapped to
                f"{english_freq.get(best_mapping.get(letter, 'E'), 0):.2f}%"  # Expected frequency
            ])

    # Print results
    print("\nFrequency Analysis Results:")
    print(tabulate(
        table_data,
        headers=['Letter', 'Count', 'Frequency', 'Maps To', 'Expected %'],
        tablefmt='grid'
    ))

    print(f"\nDecryption quality score: {best_score:.2%}")
    print("\nDecrypted text:")
    print(decrypted_text)

    # Save the solution
    save_solution(decrypted_text)

if __name__ == "__main__":
    main()
```

```
MXQJ YI IOCFXEWUQH. VEH Q BEEEEEDW, BEEEDW JYCU Y XQLU DULUH REJXUHUT UDWQWYDW COIUBV YD JXYI VHQDSXYIU. Y TYT DEJ KDTUHIJQDT MXQJ YJ YI.
DEM JXQJ JXU IXEM YI XQLYDW YJI BQIJ IUQIED, Y TUSYTUT JE VYDQBBO WYLU YD, WYLU IOCFXEWUQH Q JHO VHEC JXU LUHO IJQHJ.
Y MEDTUHUT XEM XQLU Y CYIIUT EKJ ED JXU QDYCU EV JXU TUSQTU QBB JXUIU OUQHI.
Y XEDUIJBO TYT DEJ ADEM MXQJ JE UNFUSJ MQJSXYDW JXU LUHO VYHIJ UFYIETU ADEMYDW QRIEBKJUBO DEJXYDW QREKJ JXU VHQDSXYIU.
JXU IXEM JEOUT MYJX CO UCEJYEDI IE CKSX YD JXQJ EFUDYDW IUJFYUSU.
YJ UDTUT KF RUYDW EDU EV JXU CEIJ BYVU QVVYHCYDW IXEMI EKJ JXUHU.
```

```
Q IJKDDYDW TYIFBQO EV YTYESO QDT QSJYED JXQJ YI REJX SXQHCYDW QDT SQFJYLQJYDW.
YJ YI SEDVYTUDJ YD YJI IJHUDWJXI QDT FQHQTUI YJI MUQADUIIUI FHEKTBO, Q IXEM JXQJ YI REJX QBB IJOBU QDT QBB IKRIJQDSU.
EX RKJ CEIJ EV QBB, YJ YI Q JHKU HEBBUH SEQIJUH EV UCEJYEDI, QDT Y TE DEJ KIU JXQJ JUHC BYWXJBO.
Y BQKWXUT, Y SHYUT, Y WEJ VHKIJHQJUT QJ JXU YDUFJYJKTU QDT IJKFYTYJO EV REJX JXU SXQHQSJUHI QDT JXU SHUQJEHI, RKJ CEIJ EV QBB, Y BELUT.
MXUD JXU SKHIU EV RQBQB VUBB YD FBQSU, IEDWI IJYBB CQDQWUT JE RHYTWU JXQJ WQF QDT SEDDUSJ KI QBB JEWUJXUH.
JXU EDU UCEJYED JXU IXEM DULUH VQYBI JE TUBYLUH YI XOFU.
JXU IKHWU EV QTHUDQBYDU QDT UDTEHFXYDI QDT QBB JXU SXUCYSQBI YD OEKH RHQYD MXUDULUH IECUJXYDW QMUIECU YI XQFFUDYDW EDISHUUD YI Q HQHU JHUQJ YD C
RKJ Q SEDIJQDJ ESSKHHUDSU YD JXYI IXEM. JXU XOFU TEUI ESSQIYEDQBBO VQYB JE TUBYLUH, JXU XYWXUH UNFUSJQJYEDI SQD IECUJYCUI RU EX JEE CKSX, RKJ IJ
MXUD OEK XUQH XYRYAY ISHUQC, OEK ADEM IXU CUQDI RKIYDUII QDT ZEYD XUH YD IEDW.
QDT JXEIU IEDWI QHU FHUSYIUBO MXQJ AUUFI IOCFXEWUQH YD JXU CYDTI EV CQDO QBB JXYI MXYBU.
Y QC DEJ QD YTEB QDYCU FUHIED, Y TUIFYIU CEIJ YTEB QDYCU QDT OUJ. QDT OUJ.
JXU CECUDJ Y XUQHT JXU IEDWI VEH JXU VYHIJ JYCU HYWXJ JXUHU ED JXU YTEB SEDSUHJ, Y MQI UDJXHQBBUT.
Y ADUM JXQJ HUWQHTBUII EV XEM JXU IXEM JKHDUT EKJ, YJ MEKBT XQLU WHUQJ CKIYS. SELUHYDW CQDO WUDHUI,
JXU IXEM XQI Q TYLUHIU FQBUJJU EV IEDWI HQDWYDW VHEC SUBJYS HESA JE UDAQ YDIFYHUT JHQSAI, JXUHU YI DE IXEHJQWU EV LQHYUJO.
OUI JXUHU QHU IEDWI JXQJ TE DEJ MEHA MUBB, RKJ JXU EDUI JXQJ HUIEDQJU VQH EKJMUYWX JXU ESSQIYEDQB RQT EDUI.  IOCFXEWUQH XQI JE RU JQAUD YD QI Q
YJ JHKBO YI CEHU JXQD JXU IKC EV YJI FQHJI. BEEAYDW RQSA, OUI Y SQD QWHUU ED QBB JXU VBQMI, IECU CQZEH, JXU IXEM XQI XQT.
OUJ YD IFYJU EV YJ QBB, QSHEII IULUD OUQHI QDT VYLU IUQIEDI. YJ YI MXO Y KBJYCQJUBO TUSYTUT YD QMQHTYDW YJ MYJX EDU EV CO HQHU DYDUI. IECUJXYDW
JXYI YDIFYHQJYEDQB, JXYI BEDW BQIJYDW YI JHKBO QD UNFUHYUDSU JE RUXEBT. VEH RUJJUH VEH MEHIU, YJ YI JXU TUVYDYJYLU QDYCU JXQJ HUFHUIUDJI JXU TUS
YJ CQO DEJ XQLU UDTUT YD JXU MQO Y MQDJUT, QDT YJ CQO DEJ XQLU JXQJ EDU AYII Y MQI BEEAYDW VEH, RKJ YJ VYBBUT JXU XEBU YD CO XUQHJ, QDT MYJX YJ
JXU WQFYDW MEKDT YD CO IEKB CQO DULUH XUQB. CQORU, VEH DEM, Y MYBB IQO JXQJ JXU IXEM UDTUT MUBB. YJ MQI DEJ FUHLUHJUT YDJE Q PECRYU VHQDSXYIU BY
DEH TYT YJ IJKCRBU XQHT YD YJI VYDQB CECUDJ QDT IJHKWWBU JE HUSBQYC YJI FQIJ WBEHO. YJ XQT IXEHJSECYDWI, RKJ JXYDAYDW RQSA,
JXU ZEKHDUO JE JXU UDT XQI RUUD Q IQJYIVOYDW EDU JXHEKWXEKJ. IE MXQJ YI IOCFXEWUQH. YJ YI Q XORHYT YTEB QDYCU. YJ YI QD QDYCU QREKJ VYIJYDW.
YJ YI VYLU IUQIEDI QDT IULUD OUQHI BEDW QDT XQI SQFJYLQJUT JXU XUQHJI EV CQDO. RKJ CEIJ YCFEHJQDJBO, YJ YI RUBYULYDW YD JXU IEDW EV OEKH XUQHJ.
```

```
PS C:\Users\wooai\OneDrive\Documents\Term 5\FCS\Foundation-of-cybersec> cd '.\lab 2\'
PS C:\Users\wooai\OneDrive\Documents\Term 5\FCS\Foundation-of-cybersec\lab 2> & C:/Users/wooai/AppData/Local/Programs/Python/Python313/python.ex
"c:/Users/wooai/OneDrive/Documents/Term 5/FCS/Foundation-of-cybersec/lab 2/ex1.py" -i story_cipher.txt

Frequency Analysis Results:
+----------+---------+-------------+-----------+--------------+
| Letter   |   Count | Frequency   | Maps To   | Expected %   |
+==========+=========+=============+===========+==============+
| A        |      20 | 0.75%       | K         | 0.77%        |
+----------+---------+-------------+-----------+--------------+
| B        |     102 | 3.84%       | L         | 4.03%        |
+----------+---------+-------------+-----------+--------------+
| C        |      70 | 2.64%       | M         | 2.41%        |
+----------+---------+-------------+-----------+--------------+
| D        |     205 | 7.72%       | N         | 6.75%        |
+----------+---------+-------------+-----------+--------------+
| E        |     206 | 7.76%       | O         | 7.50%        |
+----------+---------+-------------+-----------+--------------+
| F        |      43 | 1.62%       | P         | 1.93%        |
+----------+---------+-------------+-----------+--------------+
| H        |     131 | 4.93%       | R         | 5.99%        |
+----------+---------+-------------+-----------+--------------+
| I        |     198 | 7.46%       | S         | 6.33%        |
+----------+---------+-------------+-----------+--------------+
| J        |     263 | 9.91%       | T         | 9.10%        |
+----------+---------+-------------+-----------+--------------+
| K        |      51 | 1.92%       | U         | 2.76%        |
+----------+---------+-------------+-----------+--------------+
| L        |      29 | 1.09%       | V         | 0.98%        |
+----------+---------+-------------+-----------+--------------+
| M        |      50 | 1.88%       | W         | 2.36%        |
+----------+---------+-------------+-----------+--------------+
| N        |       3 | 0.11%       | X         | 0.15%        |
+----------+---------+-------------+-----------+--------------+
| O        |      58 | 2.18%       | Y         | 1.97%        |
+----------+---------+-------------+-----------+--------------+
| P        |       1 | 0.04%       | Q         | 0.10%        |
+----------+---------+-------------+-----------+--------------+
| Q        |     213 | 8.02%       | A         | 8.20%        |
+----------+---------+-------------+-----------+--------------+
| R        |      35 | 1.32%       | B         | 1.49%        |
+----------+---------+-------------+-----------+--------------+
| S        |      61 | 2.30%       | C         | 2.78%        |
+----------+---------+-------------+-----------+--------------+
| T        |      98 | 3.69%       | D         | 4.25%        |
+----------+---------+-------------+-----------+--------------+
| U        |     305 | 11.49%      | E         | 12.70%       |
+----------+---------+-------------+-----------+--------------+
| V        |      50 | 1.88%       | F         | 2.23%        |
+----------+---------+-------------+-----------+--------------+
| W        |      71 | 2.67%       | G         | 2.02%        |
+----------+---------+-------------+-----------+--------------+
| X        |     160 | 6.03%       | H         | 6.09%        |
+----------+---------+-------------+-----------+--------------+
| Y        |     229 | 8.63%       | I         | 6.97%        |
+----------+---------+-------------+-----------+--------------+
| Z        |       3 | 0.11%       | J         | 0.15%        |
+----------+---------+-------------+-----------+--------------+

Decryption quality score: 88.82%

Decrypted text:
WHAT IS SYMPHOGEAR. FOR A LOOOOONG, LOOONG TIME I HAVE NEVER BOTHERED ENGAGING MYSELF IN THIS FRANCHISE. I DID NOT UNDERSTAND WHAT IT IS.
THE MOMENT I HEARD THE SONGS FOR THE FIRST TIME RIGHT THERE ON THE IDOL CONCERT, I WAS ENTHRALLED.
I KNEW THAT REGARDLESS OF HOW THE SHOW TURNED OUT, IT WOULD HAVE GREAT MUSIC. COVERING MANY GENRES,
THE SHOW HAS A DIVERSE PALETTE OF SONGS RANGING FROM CELTIC ROCK TO ENKA INSPIRED TRACKS, THERE IS NO SHORTAGE OF VARIETY.
YES THERE ARE SONGS THAT DO NOT WORK WELL, BUT THE ONES THAT RESONATE FAR OUTWEIGH THE OCCASIONAL BAD ONES.  SYMPHOGEAR HAS TO BE TAKEN IN AS A
IT TRULY IS MORE THAN THE SUM OF ITS PARTS. LOOKING BACK, YES I CAN AGREE ON ALL THE FLAWS, SOME MAJOR, THE SHOW HAS HAD.
YET IN SPITE OF IT ALL, ACROSS SEVEN YEARS AND FIVE SEASONS. IT IS WHY I ULTIMATELY DECIDED IN AWARDING IT WITH ONE OF MY RARE NINES. SOMETHING
THIS INSPIRATIONAL, THIS LONG LASTING IS TRULY AN EXPERIENCE TO BEHOLD. FOR BETTER FOR WORSE, IT IS THE DEFINITIVE ANIME THAT REPRESENTS THE DEC
IT MAY NOT HAVE ENDED IN THE WAY I WANTED, AND IT MAY NOT HAVE THAT ONE KISS I WAS LOOKING FOR, BUT IT FILLED THE HOLE IN MY HEART, AND WITH IT
THE GAPING WOUND IN MY SOUL MAY NEVER HEAL. MAYBE, FOR NOW, I WILL SAY THAT THE SHOW ENDED WELL. IT WAS NOT PERVERTED INTO A QOMBIE FRANCHISE LI
NOR DID IT STUMBLE HARD IN ITS FINAL MOMENT AND STRUGGLE TO RECLAIM ITS PAST GLORY. IT HAD SHORTCOMINGS, BUT THINKING BACK,
THE JOURNEY TO THE END HAS BEEN A SATISFYING ONE THROUGHOUT. SO WHAT IS SYMPHOGEAR. IT IS A HYBRID IDOL ANIME. IT IS AN ANIME ABOUT FISTING.
```

```
IT IS FIVE SEASONS AND SEVEN YEARS LONG AND HAS CAPTIVATED THE HEARTS OF MANY. BUT MOST IMPORTANTLY, IT IS BELIEVING IN THE SONG OF YOUR HEART.

Decrypted text has been saved to solution.txt
```

```
WHAT IS SYMPHOGEAR. FOR A LOOOOONG, LOOONG TIME I HAVE NEVER BOTHERED ENGAGING MYSELF IN THIS FRANCHISE. I DID NOT UNDERSTAND WHAT IT IS.
THE MOMENT I HEARD THE SONGS FOR THE FIRST TIME RIGHT THERE ON THE IDOL CONCERT, I WAS ENTHRALLED.
I KNEW THAT REGARDLESS OF HOW THE SHOW TURNED OUT, IT WOULD HAVE GREAT MUSIC. COVERING MANY GENRES,
THE SHOW HAS A DIVERSE PALETTE OF SONGS RANGING FROM CELTIC ROCK TO ENKA INSPIRED TRACKS, THERE IS NO SHORTAGE OF VARIETY.
YES THERE ARE SONGS THAT DO NOT WORK WELL, BUT THE ONES THAT RESONATE FAR OUTWEIGH THE OCCASIONAL BAD ONES.  SYMPHOGEAR HAS TO BE TAKEN IN AS A
IT TRULY IS MORE THAN THE SUM OF ITS PARTS. LOOKING BACK, YES I CAN AGREE ON ALL THE FLAWS, SOME MAJOR, THE SHOW HAS HAD.
YET IN SPITE OF IT ALL, ACROSS SEVEN YEARS AND FIVE SEASONS. IT IS WHY I ULTIMATELY DECIDED IN AWARDING IT WITH ONE OF MY RARE NINES. SOMETHING
THIS INSPIRATIONAL, THIS LONG LASTING IS TRULY AN EXPERIENCE TO BEHOLD. FOR BETTER FOR WORSE, IT IS THE DEFINITIVE ANIME THAT REPRESENTS THE DEC
IT MAY NOT HAVE ENDED IN THE WAY I WANTED, AND IT MAY NOT HAVE THAT ONE KISS I WAS LOOKING FOR, BUT IT FILLED THE HOLE IN MY HEART, AND WITH IT
THE GAPING WOUND IN MY SOUL MAY NEVER HEAL. MAYBE, FOR NOW, I WILL SAY THAT THE SHOW ENDED WELL. IT WAS NOT PERVERTED INTO A QOMBIE FRANCHISE LI
NOR DID IT STUMBLE HARD IN ITS FINAL MOMENT AND STRUGGLE TO RECLAIM ITS PAST GLORY. IT HAD SHORTCOMINGS, BUT THINKING BACK,
THE JOURNEY TO THE END HAS BEEN A SATISFYING ONE THROUGHOUT. SO WHAT IS SYMPHOGEAR. IT IS A HYBRID IDOL ANIME. IT IS AN ANIME ABOUT FISTING.
IT IS FIVE SEASONS AND SEVEN YEARS LONG AND HAS CAPTIVATED THE HEARTS OF MANY. BUT MOST IMPORTANTLY, IT IS BELIEVING IN THE SONG OF YOUR HEART.
```

Summary of how the code works:

This script uses two main principles:

A. Frequency Analysis

- In English, some letters appear more often than others.
- Example: `'E'` is the most common letter (~12.7%), followed by `'T'`, `'A'`, etc.
- By comparing letter frequencies in the encrypted text with standard English frequencies, we can make educated guesses about substitutions.

B. Dictionary-Based Word Matching

- After creating an initial mapping, the script improves it by checking which swaps increase the number of recognizable English words.
- This makes the decryption more accurate than just frequency matching alone.

1. **Load Encrypted Text**
    - From user-specified file.
    - Only letters are analyzed; punctuation and spaces are preserved during decryption.
2. **Analyze Letter Frequencies**
    - Count how often each letter appears.
3. **Create Initial Letter Mapping**
    - Match highest-frequency encrypted letter to `'E'`, next to `'T'`, etc.
4. **Improve Mapping via Swaps**
    - Try swapping letter mappings to maximize number of real English words.
    - Uses a word list from the `nltk.corpora.words` dataset.
5. **Decrypt Text Using Best Mapping**
    - Apply the final mapping to produce readable output.
6. **Output Results**
    - Shows:
        - Letter frequency table.
        - Decryption score (how many words match English).
        - Sample decrypted text.
    - Saves full decrypted text to `solution.txt`.

Frequency Analysis

```
def get_letter_frequency(text):
    letters = re.findall(r'[A-Z]', text)
    return Counter(letters)
```

- Extracts only alphabetic characters (`A-Z`) from the text.
- Counts how often each letter appears using `collections.Counter`.

English Language Statistics

```
def get_english_frequency():
    return {
        'E': 12.70, 'T': 9.10, 'A': 8.20, 'O': 7.50, 'I': 6.97,
        'N': 6.75, 'S': 6.33, 'H': 6.09, 'R': 5.99, 'D': 4.25,
        'L': 4.03, 'C': 2.78, 'U': 2.76, 'M': 2.41, 'W': 2.36,
        'F': 2.23, 'G': 2.02, 'Y': 1.97, 'P': 1.93, 'B': 1.49,
        'V': 0.98, 'K': 0.77, 'J': 0.15, 'X': 0.15, 'Q': 0.10,
        'Z': 0.07
    }
```

Provides expected frequency percentages for English letters

source: https://en.wikipedia.org/wiki/Letter_frequency

Mapping Creation

```
def create_initial_mapping(text_freq, english_freq): # Sort both frequency distributions
    text_letters = sorted(text_freq.items(), key=lambda x: x[1], reverse=True)
    english_letters = sorted(english_freq.items(), key=lambda x: x[1], reverse=True)
    mapping = {}
    for (encrypted, _), (decrypted, _) in zip(text_letters, english_letters):
        mapping[encrypted] = decrypted
    return mapping
```

The function sorts both encrypted and English letter frequencies in descending order.

This maps the most frequent encrypted letter to the most frequent English letter, and so on. This gives us our **first guess** at the substitution cipher.

Decryption Function

```
def decrypt_text(text, mapping):
    result = []
    for char in text:
        if char in string.ascii_uppercase:
            result.append(mapping.get(char, char))
        else:
            result.append(char)
    return ''.join(result)
```

Applies the current mapping to transform encrypted text into a possible plaintext version.

Scoring Decryption Quality

```
def score_decryption(decrypted_text, english_words):
    words = get_words(decrypted_text)
    if not words:
        return 0   # Count how many words are valid English words
    valid_words = sum(1 for word in words if word in english_words)
    return valid_words / len(words)
```

We split decrypted text into words and compare them to a list of known English words.

We then calculate the **percentage of valid English words** — this is the "score" used to evaluate decryption quality.

Improving the Mapping

```
def find_best_mapping(text, initial_mapping, english_words, max_iterations=100):
    best_mapping = initial_mapping.copy()
    best_score = score_decryption(decrypt_text(text, best_mapping), english_words)

    for _ in range(max_iterations):
        current_mapping = best_mapping.copy()
        improved = False
        for letter1, letter2 in itertools.combinations(string.ascii_uppercase, 2):
            if letter1 in current_mapping and letter2 in current_mapping:
                current_mapping[letter1], current_mapping[letter2] = current_mapping[letter2], current_mapping[letter1]
                current_score = score_decryption(decrypt_text(text, current_mapping), english_words)
                if current_score > best_score:
                    best_score = current_score
                    best_mapping = current_mapping.copy()
                    improved = True
                else:
                    current_mapping[letter1], current_mapping[letter2] = current_mapping[letter2], current_mapping[letter1]
        if not improved:
            break
    return best_mapping, best_score
```

Tries swapping pairs of mappings to see if they result in better decryption scores.

Uses `itertools.combinations` to test all possible letter swaps.

Stops when no further improvement is found or after a set number of iterations.

Part II: Substitution Cipher

In this section, we aim to change an encrypted message without being able to decrypt it.

For example, we can change Student ID 100XXXX gets a total of 0 points! to any message of our choosing.

Your aim is to get change the decrypted plain text response to say you have gotten 4 points, without decrypting it yourself.

For example, the text should say Student ID 100XXXX gets a total of 4 points! after decryption.

In other words, you manipulate the cipher text , so that it decrypts to a plain text of your choosing.

Thus, you are compromising the integrity of the encrypted message.

Hints:

- The ciphertext is encrypted with a OTP. You do not know what the OTP is, it is randomly generated.
- You do not need to know anything about the OTP for this exercise.

You are provided with ex2.py in this folder. Complete it to show that you can change the encrypted message without knowledge of the OTP.

```
def hax():
    # TODO: manipulate ciphertext to decrypt to:
    # "Student ID 100XXXX gets 4 points"
    # Remember your goal is to modify the encrypted message
    # therefore, you do NOT decrypt the message here
    original = b"Student ID 1000000 gets 0 points\n"
    desired = b"Student ID 100XXXX gets 4 points\n"
    difference = XOR(original, desired)
    new_cipher = XOR(original_cipher, difference)
    return new_cipher
```

we can use the equation:

```
cipher = plain ^ OTP
```

```
new_cipher = cipher ^ (plain ^ desired)
           = plain ^ OTP ^ plain ^ desired
           = OTP ^ desired
```

```
PS C:\Users\wooai\OneDrive\Documents\Term 5\FCS\Foundation-of-cybersec> & C:/Users/wooai/AppData/Local/Programs/Python/Python313/python.exe "c:/U
b'Student ID 1000000 gets 0 points\n'
b'Student ID 100XXXX gets 4 points\n'
```