密碼工程 Quiz3

學號：113550021 姓名：陳孟楷

Problem 1

Question 1-1.

Feature	SHA-256	SHA-512/256
Bit Size of Internal State	256-bit	512-bit
Block Size	512-bit	1024-bit
Word Size	32-bit	64-bit
Number of Rounds	64	80
Final Output Size	256-bits	256-bits
Preimage Resistance	2^{256}	2^{256}
Second Preimage Resistance	2^{256}	2^{256}
Collision Resistance	2^{128}	2^{128}

Table 1. Structure and security comparison

Question 1-2.

Generally, they provide similar security level. However, when it comes to length-extension attacks, SHA-512/256 has stronger resistance.

Question 1-3.

SHA-512/256 is generally better than SHA-256 because:

1. It performs better on 64-bit processors, and modern CPUs are mostly 64-bit.
2. It is more resistant to length extension attacks as mentioned in Question1-2.
3. It owns higher efficiency for large data processing since it owns larger block size, which can iterate less time for the same amount of data.

Question 2-1.

Traditional SHA functions are designed to be one-way, which makes them unsuitable as conventional encryption algorithms. However, if we put some modifications on it, it could be adapted to symmetric encryption.

Question 2-2.

Because it has an extendable output, stronger security guarantees and it's more resistant to length extension attacks, it is suitable for encryption.

Question 3.

```
Input the plaintext: Do you wanna build a snowman
Key:  b"Z\x08\xc9\xd0b\xf9\x18\xd4'\x97\x11\xde&c\xd5]1*\x91\xa3\xa2\xe3E\xd4\x9fJ'3>\x86\x93\xe1"
Encrypted: c5975c15fb090ee836e2fe9f5a36e9ba33895a15e32532e4eac3b9ba
Decrypted: Do you wanna build a snowman
```

Figure 1. Run the code with the example "Do you wanna build a snowman"

Problem 2

Question 1.

f_i is the frequency of each letter in the text, while N is the total number of letters in the text.

Question 2-1.

```
Index of Coincidence: 0.067
```

Figure 2. Index of coincidence (IC) calculated by Python

Question 2-2.

From the reference, we found that the IC of English is about 0.068, which is closed to the result we calculated. Therefore, I guess it's English.

Question 3-1.

From the reference, we know that IC of English is about 0.068 and IC of randomly generated text (for 26 letters) is about 0.038. As for Chinese, the IC is closed to 0 (about 0.02 to 0.03).

Question 3-2.

English has a high due to its small alphabets and uneven letter distribution, making frequency analysis effective for decryption. Chinese, with thousands of characters has a lower IC since its character distribution is more balance, making frequency-based decryption harder. Randomly generated text has an IC of about 0.038, as characters appear with roughly equal probability, making the frequency-based decryption harder as well.

Question 4.

The ciphertext seems to be the Caesar cipher. So, I wrote a Python code for decrypting the ciphertext to parallelly shift the alphabets. And finally get the plaintext

```
Shift 3: THERE ARETW OWAYS OFCON STRUC TINGA SOFTW AREDE SIGNO NEWAY ISTOM AKEIT SOSIM PLETH ATTHE REARE LYVIO USLYN ODEFI CIENC IESAN DTHEO THERW AYIST OMAKE
 ITSOC OMPLI CATED THATT HEREA RENOO LYVIOU SDEFI ZIENC IESTH EFIRS TMETH ODICF ARMOR EDIFF ICULT
```

Figure 3. Plaintext

Problem 3

Question 1.

Using IC analysis, the most probable key length is 4.

```python
# Get key length (Question 1)
def get_key_length(ciphertext, max_key_length=7):
    def index_of_coincidence(text):
        freqs = Counter(text)
        N = len(text)
        return sum(f * (f - 1) for f in freqs.values()) / (N * (N - 1)) if N > 1 else 0

    best_guess = 1
    best_ic = 0
    for key_len in range(1, max_key_length + 1):
        segments = [''.join(ciphertext[i::key_len]) for i in range(key_len)]
        avg_ic = np.mean([index_of_coincidence(segment) for segment in segments])
        if avg_ic > best_ic:
            best_ic = avg_ic
            best_guess = key_len
    return best_guess
```

Figure 4. Get the key length with Python code

```
Probable Key Length: 4
```

Figure 5. Result for the most probable key length

Question 2.

Apply Chi-square test and English letter frequency analysis, the key is “NYCU”.

```python
# Chi-square test (Question 2)
def chi_square_stat(observed, expected):
    return sum(((observed[c] - expected[c]) ** 2) / expected[c] for c in string.ascii_uppercase if expected[c] > 0)

# Find key (Question 2)
def find_key(ciphertext, key_length, freq_map):
    key = ''
    expected_freqs = {c: freq * len(ciphertext) / key_length for c, freq in freq_map.items()}
    for i in range(key_length):
        segment = ''.join(ciphertext[i::key_length])
        min_chi_sq = float('inf')
        best_shift = 0
        for shift in range(26):
            shifted_segment = ''.join(chr((ord(c) - ord('A') - shift) % 26 + ord('A')) for c in segment)
            observed_freqs = Counter(shifted_segment)
            chi_sq = chi_square_stat(observed_freqs, expected_freqs)
            if chi_sq < min_chi_sq:
                min_chi_sq = chi_sq
                best_shift = shift
        key += chr(best_shift + ord('A'))
    return key
```

Figure 6. Get the key with Python code

```
Identified Key: NYCU
```

Figure 7. Result for the key

Question 3.

The plaintext is **The Declaration of Independence**.

```python
# Decrypt (Question 3)
def decrypt_vigenere(ciphertext, key):
    """Decrypts the Vigenère cipher using the identified key."""
    plaintext = []
    key_length = len(key)

    for i, char in enumerate(ciphertext):
        shift = ord(key[i % key_length]) - 65
        decrypted_char = chr(((ord(char) - shift - 65) % 26) + 65)
        plaintext.append(decrypted_char)

    return ''.join(plaintext)
```

Figure 8. Decrypt the ciphertext with Python code