密碼工程 Quiz2

學號：113550021 姓名：陳孟楷

Problem 1

Question 1.

The weakness is “easy to be cracked while using the same device to do 2FA” and “inconvenient”.

The solutions are “separate 2FA on different devices” and “biometric authentication”.

Ex.

1. Nowadays, we usually use the same device to do 2FA. However, "once the device is hacked, it's easy for the hacker to crack both authentications." -> We can separate two authentications on two different devices.

2. While I'm using the NYCU VPN, I felt quite "inconvenient" since I always need to take my phone out to do the 2FA when I want to log in NYCU Portal. -> We can add some biometric authentications such as fingerprint or FACE ID, which may become more convenient.

Question 2.

We can establish a site-to-site VPN, intergrating with SSO, using multiple VPN gateways, security monitering and secure data transmission.

Ex.

1. Site-to-site VPN: We can establish a secure IPsec-based site-to-site VPN between two campuses, allowing seamingless resource sharing while ensuring encrypted communication.
2. Intergrating with SSO: We can use SSO for unified identity management across both campuses.
3. Multiple VPN gateways: We can deplot multiple VPN gateways to distribute traffic efficiency and prevent network congestion.
4. Security monitering: We can use IDPS to moniter VPN traffic for potential threats and anomalies.
5. Sucure data transmission: We can use AES-256 encryption for VPN tunnels to protect research data from interception.

Problem 2

Results

Easy hash

```
Hash: 884950a05fe822dddee8030304783e21cdc2b246
Password: scorpion
Took 302 attempts to crack message.
```

Medium hash

```
Hash: 9b467cbabe4b44ce7f34332acc1aa7305d4ac2ba
Password: wh00sh
Took 939438 attempts to crack message.
```

Leet hacker hash

```
Hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: puppy
Took 2854 attempts to crack message.
```

Description for the program

Use sha1 function in hashlib in Python to solve

```
4   def break_sha1_hash(target_hash, password_list, mode):
5       attempts = 0
6       for password in password_list:
7           if mode == 1:
8               password=(salt_answer+password)
9           hash_attempt = hashlib.sha1(password.encode()).hexdigest()
10          attempts += 1
11          if hash_attempt == target_hash:
12              return password, attempts
13      return None, attempts
```

Read the password file

```
15  def load_password_list(file_path):
16      with open(file_path, 'r') as file:
17          password_list = file.read().splitlines()
18      return password_list
```

```
25  password_list_file = "password.txt"
26  password_list = load_password_list(password_list_file)
```

Output the results for easy and medium hash

```
28  for hash_name, target_hash in hashes_to_break.items():
29      clear_text_password, attempts = break_sha1_hash(target_hash, password_list, 0)
30      if clear_text_password:
31          print(f"Hash: {target_hash}")
32          print(f"Password: {clear_text_password}")
33          print(f"Took {attempts} attempts to crack message.")
```

Use the salt to break and output the result for leet hacker hash

```
38  salt_answer, attempt1 = break_sha1_hash(salt, password_list, 0)
39  clear_text_password, attempt2 = break_sha1_hash(leet_hacker_hash, password_list, 1)
40  clear_text_password = clear_text_password.replace(salt_answer,"")
41
42  if clear_text_password:
43      print(f"Hash: {leet_hacker_hash}")
44      print(f"Password: {clear_text_password}")
45      print(f"Took {attempt2} attempts to crack message.")
```

Problem 3

Result in logger.log

Finally, it found out 7 bits of my student ID but failed to find the 8th and 9th bit

```
2025/03/27 17:46:27 [INFO] [preImage] f8b0b5c4d8e0ed75e5c69b6a8f8b7162dcfcc8feddd9ac2c697b6027921c9311
2025/03/27 17:46:27 [INFO] [Round 1 with nonce 00000002] 10d93a07de6d24fb6cb30260096b85177af7ca9580a713bcbedcd96e33e314ea
2025/03/27 17:46:27 [INFO] [Round 2 with nonce 00000072] 118f678b3dfa013072444983a5b1cf805757d0a908bc340706bdfb272099a81f
2025/03/27 17:46:27 [INFO] [Round 3 with nonce 00000e99] 11302cc34693315f9be6c7e39ff2a03cd830d30a096186ba16649a49cb534b11
2025/03/27 17:46:27 [INFO] [Round 4 with nonce 00005873] 113550f882b17262635d8c48f7a77534a1a62244fc118bee4e3557a2408b8ec0
2025/03/27 17:46:27 [INFO] [Round 5 without nonce] 113550f882b17262635d8c48f7a77534a1a62244fc118bee4e3557a2408b8ec0
2025/03/27 17:46:27 [INFO] [Round 6 without nonce] 113550f882b17262635d8c48f7a77534a1a62244fc118bee4e3557a2408b8ec0
2025/03/27 17:49:44 [INFO] [Round 7 with nonce 166db968] 1135500021ed198ec50ff1c613a4b043bc4c18bd4d8a7cb68194d337436beeb3
2025/03/27 18:26:57 [EROR] [Round 8] not found with running out of nonce
```

Description for the program

Configure logging file and change the ERROR into EROR

```
logging.basicConfig(filename='logger.log', level=logging.INFO,
                    format='%(asctime)s [%(levelname)s] %(message)s', datefmt='%Y/%m/%d %H:%M:%S')
logging._levelToName[logging.ERROR] = "EROR"
```

Use sha256 in hashlib in Python to get the preImage

```
def sha256(data: str) -> str:
    return hashlib.sha256(data.encode()).hexdigest()
```

```
    pre_image = sha256(student_id)
    logging.info(f'[preImage] {pre_image}')
```

Mining the student ID

```
def mine_block(previous_hash: str, target_prefix: str) -> (str, str):
    nonce = 0
    while nonce <= 0xffffffff:
        candidate = sha256(previous_hash + format(nonce, '08x'))
        if candidate.startswith(target_prefix):
            return candidate, format(nonce, '08x')
        nonce += 1
    return None, None
```

```
    start_block = 1
    for i, digit in enumerate(student_id):
        if pre_image[i] != digit:
            start_block = i + 1
            break

    previous_hash = pre_image

    for round_num in range(start_block, len(student_id) + 1):
        target_prefix = student_id[:round_num]

        if previous_hash.startswith(target_prefix):
            logging.info(f'[Round {round_num} without nonce] {previous_hash}')
            continue

        new_hash, nonce = mine_block(previous_hash, target_prefix)
        if new_hash:
            logging.info(f'[Round {round_num} with nonce {nonce}] {new_hash}')
            previous_hash = new_hash
        else:
            logging.log('[EROR]' + f'[Round {round_num}] not found with running out of nonce')
            break
```