Network Security Assignment - 01
Project Zero

Ketan Mohan Garg (2022248)
Keshav Bindlish (2022246)

Introduction

In this Assignment, we are tasked with applying a brute-force attack for decrypting a ciphered text based on a permutation of an encryption key. The encryption and decryption functions use simple substitution techniques, where each letter of a plaintext string is mapped to another letter using a key. We have to generate ciphertexts based on a provided key, and then use a brute-force method to find the key by decrypting the ciphertexts and comparing the plaintexts using hash values. We employ the MD5 hash function to ensure the secrecy of the recovered plaintext.

Code Explanation

1. **Imports**:
    - itertools: Used to generate all permutations of a given list (in this case, the alphabet).
    - hashlib: Provides the MD5 hashing algorithm for generating hashes.
    - random: Used to randomly shuffle the elements of the Omega list to simulate a key.
2. **Functions**:

generate_hash(string): This function takes a string as input, encodes it into bytes, and returns the first 6 characters of its MD5 hash value.

```
def generate_hash(string):
    return hashlib.md5(string.encode()).hexdigest()[:6]
```

encrypt_func(plaintext, key): This function encrypts a given plaintext string using the given key. The key is used to map each character in the original string to a new character. The mapping is done by doing an encryption table where the OMEGA characters are substituted by the characters from the key.

```
def encrypt_func(plaintext, key):
    string, hash_val = plaintext
    encryption_table = {OMEGA[i]: key[i] for i in range(len(OMEGA))}
    encrypted_string = ''
    for char in string:
        encrypted_string += encryption_table[char]
        return encrypted_string
```

decrypt_func(ciphertext, key): This function decrypts a given ciphertext using the same key. It makes a decryption table by reversing the encryption mapping. Each character in the ciphertext is matched back to its original character.

```
def decrypt_func(ciphertext, key):
    decryption_table = {key[i]: OMEGA[i] for i in range(len(OMEGA))}
    decrypted_string = ''
    for char in ciphertext:
        decrypted_string += decryption_table[char]
    return decrypted_string
```

brute_force_attack(ciphertexts, plaintexts): It generates all possible permutations of the `OMEGA` list and try to decrypt each ciphertext using the permutation constructed as the key. If the decrypted text on comparison matches the original plaintext (validated using the hash function), it returns the key that was passed in decrypting the ciphertext.

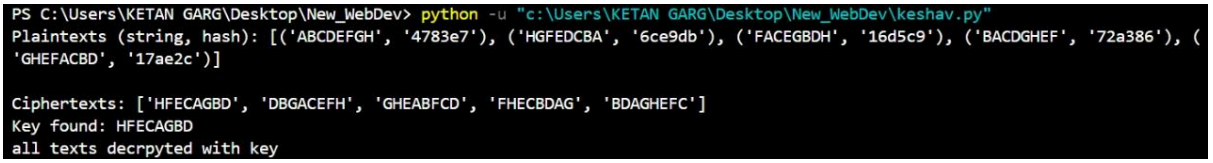
```
def brute_force_attack(ciphertexts, plaintexts):
    for combo in itertools.permutations(OMEGA):
        key = list(combo)
        found = 0
        for i, ciphertext in enumerate(ciphertexts):
            decrypted_text = decrypt_func(ciphertext, key)
            ostring, ohash = plaintexts[i]
            if generate_hash(decrypted_text) == ohash:
                found = 1
                break
        if found == 1:
            print(f"Key found: {''.join(key)}")
            return key
    return None
```

main(plaintexts: list, key: str): This function is responsible for generating ciphertexts from plaintexts and then running a brute-force attack to find the encryption key. It prints out the list of ciphertexts and calls the brute-force function to recover the key. If the key is found, it is success.

```
def main(plaintexts: list, ekey: str) -> int:
    ciphertexts = [encrypt_func(pt, ekey) for pt in plaintexts]
    print("\nCiphertexts:", ciphertexts)
    discovered_key = brute_force_attack(ciphertexts, plaintexts)
    return 1 if discovered_key else 0
```

3. **Main Execution Flow**:
    - An array of plain texts named original_strings is hard coded in the code.
    - A random encryption key is generated using random.sample(), which is one of the permutations of omega.
    - The function main() is called by passing the original_strings and the generated key which then tries to decrypt all ciphertexts using brute force attack and successfully discovers the key.

CODE OUTPUT



Random key generated in this Case is **HFECAGBD,** which is successfully found through the brute force attack.

Random key generated in this Case is **ABDCGHEF,** which is successfully found through the brute force attack.

Plain texts are hard coded and first plain text was kept ABCDEFGH, to check functionality by checking if the corresponding ciphertext is exactly the same as the key or not.