CSE350: Network Security
Programming Exercise 1 Report

Chetan (2020046), Pratham Koli (2020453)

The given system uses poly-alphabetic substitution for encryption and decryption over a character consisting of English lowercase alphabets and ensures that the plaintext is recognisable using the md5 hashing algorithm.

For the encryption using poly-alphabetic substitution, first its md5 hash hex digest is generated using hashlib library, which is converted to the limited character set using custom mapping and the same is appended to the final plaintext that is being encrypted.

```
def encrypt_text(plaintext, key):

    cipher_text = ""

    hash_val = hashlib.md5(bytes(plaintext, "utf-16")).hexdigest()
    hash_in_chars = hex_to_char_mapper(hash_val)

    final_plaintext = plaintext + hash_in_chars

    for i in range(len(final_plaintext)):
        if((charsToIndc[final_plaintext[i]] + (charsToIndc[key[i%(len(key))]] - 1)) > 26):
            cipher_text += indcToChar[(charsToIndc[final_plaintext[i]] + (charsToIndc[key[i%(len(key))]] - 1))%26]
        else:
            cipher_text += indcToChar[(charsToIndc[final_plaintext[i]] + (charsToIndc[key[i%(len(key))]] - 1))]

    return cipher_text
```

Decryption uses a relatively simpler mapping using the provided key in the following manner.

```
def decrypt_text(ciphertext, key):
    decrypt_text_str = ""

    for i in range(len(ciphertext)):
        try:
            decrypt_text_str += indcToChar[(charsToIndc[ciphertext[i]] - (charsToIndc[key[i%(len(key))]] - 1))%26]
        except KeyError:
            decrypt_text_str += ciphertext[i]

    return decrypt_text_str
```

A function is declared to check whether the decrypted text is the original plaintext in the following way

- Since the Md5 hash adds 32 extra characters to our original text, we first split the hash and the original plain text within these characters.
- The obtained original text is then hashed again and mapped to the given character set.
- The final hash is compared to the last 32 characters in our decrypted text.
- If the given holds, it satisfies the property, which is considered our original plaintext.

```
def check_property_text(deciphered_text):

    str_val = deciphered_text[:len(deciphered_text)-32]
    hash_val = deciphered_text[len(deciphered_text)-32:]

    final_hash_val = hashlib.md5(bytes(str_val, "utf-16")).hexdigest()
    hash_in_chars = hex_to_char_mapper(final_hash_val)

    if hash_val == hash_in_chars:
        return True
    else:
        return False
```

Finally, the brute force uses itertools to generate possible permutations of our character set with length four and tests each key for the first text in the list to find the possible key by decrypting with each one. The same is repeated for the rest of the keys if a match is found.

```python
def brute_force_key(cipher_text_list):

    character_set = "abcdefghijklmnopqrstuvwxyz"

    for i in itertools.permutations(character_set, 4):
        current_key = "".join(i)
        # print(current_key)
        deciphered_text = decrypt_text(cipher_text_list[0],current_key)
        # print(deciphered_text)

        if (check_property_text(deciphered_text) == True):
            for i in range(1, len(cipher_text_list)):
                deciphered_text_1 = decrypt_text(cipher_text_list[i],current_key)

                if(check_property_text(deciphered_text_1) == False):
                    break
                else:
                    return current_key
```

The inputs and outputs for the same can be seen below:

```python
key = "cbpa"

message_list = ["hello", "world", "makes", "stairway", "heaven"]
print("-"*32)
print("List of plain texts")
print(message_list)
print("-"*32)
```

```
CHETAN@LAPTOP-7BB1SI10 MINGW64 ~/Documents/IIITD/Winter 2022/cse350-network-security (main)
$ python -u "/c/Users/CHETAN/Documents/IIITD/Winter 2022/cse350-network-security/Assignment1/TextCipher.py"
--------------------------------
List of plain texts
['hello', 'world', 'makes', 'stairway', 'heaven']
--------------------------------
List of encrypted texts
['jfalqdukrieiglydmhbphnaackvhohpjdbbhp', 'ypglfptgcexjfmbhejebjgvldfvanlsglcqan', 'obzeugtornwmhqdoepqphbxpkexlpqdlrfsoe',
bphpfcgcnojpnokakpnunlq']
--------------------------------
Using bruteforce to find keys
--------------------------------
Key found:  cbpa
--------------------------------
```