CS 406: Cryptography and Network Security
Hands On - 1

Chaitanya Garg, 210050039

January 21, 2024

Challenge - 1

The key component used in this Challenge was that both the flag and message were meaningful English Sentences.

Taking the xor of the 2 cipher-texts (= cip) gives us the xor of the flag and the message since the same key was used. Now we know that the flag starts with 'cs406{'. Its xor with cip gives 'Crypta', so the message must start with that. So a logical guess for message can be 'Cryptanalysis', which gives (on xor with cip) the flag as 'cs406{one_tim'. Continuing this process, we can guess some more characters in flag, which will reveal some more characters in message and then we can guess in message and get flag. If stuck, we can use the hint that flag contains the word 'compromises' and try it in all available places, and fix it in the place that gives meaningful message.

With this, we get the whole message ('Cryptanalysis frequently involves statistical attacks') and flag ('cs406{one_time_pad_key_reuse_compromises_security!!!}').

```python
from Crypto.Util.strxor import strxor

with open('ciphertext1.enc', 'rb') as f1:
    cip1 = f1.readline()


with open('ciphertext2.enc', 'rb') as f2:
    cip2 = f2.readline()

cip = strxor(cip1, cip2)

flag = b'cs406{one_time_pad_key_reuse_compromises_security!!!}'
msg =  b'Cryptanalysis frequently involves statistical attacks'

print(strxor(cip, msg))
print(strxor(cip, flag))
```

Challenge - 2

In this challenge, if we look at the encryptor code given, it is very clear that instead of generating all the characters of the key randomly and independently, only 1 character is chosen at random (from 0 to 127) and then the whole key is generated deterministically from that character. Hence, since we also know the length of cipher-text and hence the length of key, there are only 128 possible keys (which can be generated easily using the same formula as in encryptor). The encryption algorithm is also very simple using the mod 128 group, hence the decryption is also straight forward. Hence we can try to decrypt the cipher-text using all 128 keys and only 1 of those make sense, which must be our flag ('cs406{algebra_enters_the_picture!}').

```python
import hashlib
import random


with open('ciphertext.enc', 'rb') as file:
    msg = file.readline()

for i in range(128):
    key = chr(i).encode()
    for _ in range(1, len(msg)):
        key += chr(hashlib.sha256(key).digest()[0]%128).encode()

    flag = b""

    for j in range(len(msg)):
        flag += chr((msg[j] + 128 - key[j])%128).encode()

    print(i, flag)
```

Challenge - 3

The distinction between the key and the random string is just that the key does not have the character 0 and the random string may have the character 0. The key not having the character 0 means all the characters in the message/payload will definitely be changed. Hence if the payload/message itself has all the characters as 0, then we can say that the encrypted text will not have the character 0, and the random string might have character 0.

If the length of the payload if n then the probability of random string 'not' containing character 0 is $= (\frac{255}{256})^n$. If we take n large enough (as I have taken $n = 30,000$), we will have the probability of random string not containing 0 to be very low ($1.0149e - 51$).

Hence it is guaranteed that the encrypted text will not have the character 0 and practically guaranteed that the random string will have the character 0. Hence we can just check for that and make the guess accordingly. The flag comes out as 'cs406{y0u_h4d_fu11_4dv4nt4g3}'.

```
# ===== YOUR CODE BELOW =====
payload = "0"*30000 # TODO: This variable should finally contain the hex-string you
    want to send
# ===== YOUR CODE ABOVE =====

#### GIVEN TEMPLATE CODE ......

    # ===== YOUR CODE BELOW =====
    # Write code here to decide whether to send c1 or c2
    # The variable c1 (which is of type str) contains the hex-encoded version of c1
    returned by the server
    # The variable c2 (which is of type str) contains the hex-encoded version of c2
    returned by the server

    l1 = list(bytes.fromhex(c1))
    guess = 1

    if 0 in l1:
        guess = 2

    # guess = 0 # TODO: Set guess to 1 or 2 accordingly if you think the correct
    answer is c1 vs. c2 respectively
    # ===== YOUR CODE ABOVE =====

#### GIVEN TEMPLATE CODE ......
```

Challenge - 4

In this Challenge we are very clearly given the encryption scheme used and hence we can easily revert the process to get the algorithm for decryption. Also the key is given to us, so that makes every thing straight-forward.

I interpreted a number in any base as a list of characters in that base (like byte array). So first we write a function, which given a number (list) and the base, converts it into required base. Then we read the cipher-text file, and from the keyfile, we take the required number of characters as key. We convert the cipher-text to base 255 from 256 (= c_list). Then character-wise from c_list and the key, we get p_i. We collect these in p_list. and then convert it from base 255 to base 256 (= m_list). This is the message m as a byte array. which we can covert back to a string and get the flag. The flag comes out to be 'cs406{r4d1x_ch4ng1ng_f0r_th3_w1n!}'.

```
def base_to_base(num: list, base_from: int, base_to: int):
    def decimal_to_base(num: int, base: int):
        ans = []
        while num > 0:
            ans.append(num % base)
            num = num // base
```

```python
        ans.reverse()
        return ans

    def base_to_decimal(num: list, base: int):
        ans = 0
        for i in num:
            ans = ans * base + i
        return ans

    tmp = base_to_decimal(num, base_from)
    return decimal_to_base(tmp, base_to)

with open('ciphertext.enc', 'rb') as file:
    cip = file.read()

with open('keyfile', 'rb') as file:
    key = list(file.read(len(cip)))

c_list = base_to_base(cip, 256, 255)
p_list = []

for ci, ki in zip(c_list, key):
    pi = (ci - ki + 1 + 255) % 255
    p_list.append(pi)

m_list = base_to_base(p_list, 255, 256)

result_string = ''.join([chr(hex_value) for hex_value in m_list])
print(result_string)
```