Comp443 - Project 1

Mehmet Üstek - 64782

Q1)

Proof by Reduction:
Theorem: If the key is uniformly picked from the space defined by the length L, there exists no program that can give the correct output with probability more than ½.

Proof: If ∃ program A that can give the correct output with probability more than ½ , when the key is uniformly picked from the space defined by the length L, then we construct a program B that can give the correct output with probability more than ½, when the key is picked from the space defined by the length L. (for N=2)

The adversary will play the next game:

1. The adversary A outputs a pair of messages $m_0, m_1 \in M$, $m_0 \neq m_1$
2. A key k is generated using Gen, a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $\leftarrow Enc_k(m_b)$ is computed and given to A.
3. A outputs a bit b'.
4. The output of the experiment is defined to be 1 if b' = b, and 0 otherwise.

The adversary will have at most ½ probability to guess the corresponding output, since the ciphertext does not contain any information about the plaintext. Only thing that the adversary knows is he is given a ciphertext from a message either from m_0 or m_1. Since there are no other information given and furthermore the ciphertext does not leak any information, the probability that the adversary wins this game is ½.

Regarding the problem, if we run the program with these two pairs:

- (Plaintext 1, Ciphertext 3), (Plaintext 2, Ciphertext 1)

The program gives these two possible keys:

- ['9a98d0918837464bc288738738271294865278efa7837838934662178ab2', '8088c18b932b4f5af993758c2e2f128f945c6de0a282652b854b610f99bb']

Meaning that one of these keys is the correct key.
Normally the correct key is the former one. However, with the proof done above, we know that the program cannot give the correct answer, since there is not a single matching key. Instead it gives two possible keys.

Q2)

The problem with the given scheme is that the key is used over and over in the scheme. This is the first requirement of the OTP that the key should not be used more than once.
For N>2:
Let's say we have three messages, m_0, m_1, m_2 with property $m_0 = m_2$ are the same message.
The ciphertext for the first message m_0 will be $m_0 \oplus k = c_0$, the second ciphertext will be $m_1 \oplus k = c_1$, the third and last ciphertext will be $m_2 \oplus k = c_2$ which will be equal to c_0 since neither k nor message is changed. Thus,
$M_0 = m_2$, $c_0 = c_2$
The adversary will play the next game:

1. A key k is generated using Gen
2. The adversary A outputs a pair of messages $m_0, m_1 \in$ M.
3. A uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $\leftarrow Enc_k(m_b)$ is computed and given to A.
4. A outputs a bit b'.
5. The output of the experiment is defined to be 1 if b' = b, and 0 otherwise.

Probability that A succeeds must be ½ by outputting a random guess.

1. The adversary plays the game with m_0 and m_1.
2. The adversary notes the given ciphertexts.
3. Be aware that the key does not change between the next iteration, since the scheme provided uses the same key.
4. The adversary plays the same game with m_1 and m_2 which $m_2 = m_0$.
5. Now since the key does not change between iterations, the first step (key generation) is skipped.
6. Adversary will observe that if the outputted ciphertext c_i is the same as before, then $c_0 = c_1$. If the adversary does not observe that the first c_0 is not the same as before, then $c_0 != c_1$.
7. If the ciphertexts are not the same, the adversary will be left out with c_0 and c_1, thus the attacker can xor these two to get $m_0 \oplus m_1$, since $m_0 \oplus k = c_0$, $m_1 \oplus k = c_1$ and;
   $m_0 \oplus m_1 = c_0 \oplus c_1$
   If the ciphertexts are same, the attacker will know that either the first encrypted message was m_0
and the second message was m_2, or m_1 was the one that was encrypted in both messages.
8. Next, the attacker can apply a **frequency analysis** at this point to get the key, with that attacker already having two ciphertexts and two messages. The attacker can easily get the key from here.
9. In the next step, the attacker will guess the bit b'. Since the attacker performed an analysis to these ciphertexts with this equation $m_0 \oplus m_1 = c_0 \oplus c_1$, the attacker now knows which ciphertext corresponds to which message. Moreover, the attacker even knows key k now. With xoring the ciphertext with the corresponding message.
10. Thus the attacker guesses the correct bit b' with probability 1.

Regarding the problem, now if we run the program with all 3 given (plaintext, ciphertext) pairs, the output will be:

- *Pairs:*
- *[('Plaintext1', 'Ciphertext3'), ('Plaintext2', 'Ciphertext1'), ('Plaintext3', 'Ciphertext2')]*
- *key:*

- *9a98d0918837464bc288738738271294865278efa7837838934662178ab2*

Now the program knows exactly which key is the correct key, with the given proof above. And for all cases, our program will find the correct key and pairs with probability 1.
That is why the One Time Pad uses a key only once, since in other cases it is trivial for our program to find the key when n > 2.

Q3)

I used python for designing and programming the given program.
My code runs in O(N^2L) time giving the correct output.
The pseudo code for the running time analysis is as follows:

```
    For p1 in range of N:
        For p2 in range of N:
            For a in range of L:
                Get plaintext hexadecimal value
                Get ciphertext hexadecimal value
                Xor the two values.
                If it is the first iteration, get the value of first plaintext and
                append it
                to the first keys list.
                If it is second, append it to the 2nd list.
                If it third, append it to the third list.
```

< This part of the code takes O(N^2L) time.>
< Now if n> 2: we have three separate keys, which we will use to evaluate and get the correct key.>
< In the case n <= 2, we may have more than one key that have the chance of being the key for the encryption>
< Because of the proof in Q1 and Q2, we now know that if we have n>2, it is trivial to find the key in O(N^2L) time.

Second part: find the identical keys on arbitrary number of sets.

<Below code takes O(N^2)>
<Purpose is to get an identical key between the first two sets.>

```
    For n times:
        For n times:
            If two keys are identical in first and second set, get the value of key.
```

<Then compare the identical sets to find the correct key which has to be in the third set>

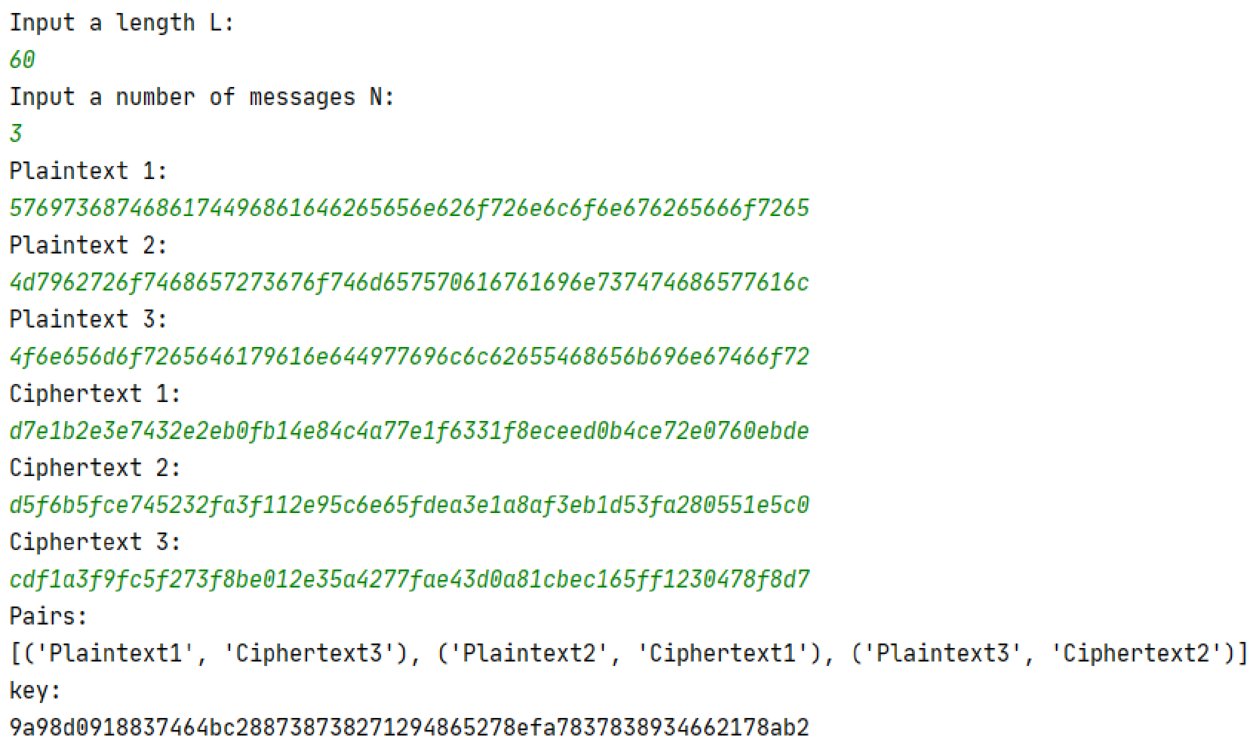
< Now we know the key, we will output the pairs>
< Below code will take O(N^2L) time considering the for loops>

```
For n1 in N:
```

```
**For n2 in N:**
    **For l in L:**
        **Get the plaintext - ciphertext values and xor them. Assign the output as key k.**
    **If key is identical the real key, get the pair (n1, n2).**
    **Else continue iterating all over the sets until the real key is observed.**
```

Thus the code takes $O(N^2L)$ running time with for loops being the sole computational factor.

The visual from the code output with the given plaintext,ciphertext pairs in project description.



```
Input a length L:
60
Input a number of messages N:
3
Plaintext 1:
5769736874686174496861646265656e626f726e6c6f6e676265666f7265
Plaintext 2:
4d7962726f7468657273676f746d657570616761696e737474686577616c
Plaintext 3:
4f6e656d6f7265646179616e644977696c6c62655468656b696e67466f72
Ciphertext 1:
d7e1b2e3e7432e2eb0fb14e84c4a77e1f6331f8eceed0b4ce72e0760ebde
Ciphertext 2:
d5f6b5fce745232fa3f112e95c6e65fdea3e1a8af3eb1d53fa280551e5c0
Ciphertext 3:
cdf1a3f9fc5f273f8be012e35a4277fae43d0a81cbec165ff1230478f8d7
Pairs:
[('Plaintext1', 'Ciphertext3'), ('Plaintext2', 'Ciphertext1'), ('Plaintext3', 'Ciphertext2')]
key:
9a98d0918837464bc28873873827129486527 8efa7837838934662178ab2
```

The second shared test pairs with 6 (plaintext, ciphertext) pairs have given the below key:
afcdfe54d37224fb6a13e08777aeca6da8d2a7d2e8ebed9034d08f74f3bda5ff0483a22a38f4357c
Which is the correct result.

Acknowledgements:
I understand the university rules for plagiarism and I have never shared or used any code or slice of code while doing this project. Thus, the effort belongs only to me.