Pham Hoang Long Do
NetID: 2995060

Homework 1

CSE 3400
Ghada Almashaqbeh

Problem 1

Part a

Python 1: XORing with 8003

```
import time

def main():
    L = [2**(10),2**(20),2**(30)]
    for i in L:
        start = time.time()
        for counter in range(0,i+1,1):
            xor = counter ^ 8003
        end = time.time()
    print(end-start)
```

```
CPU: I7-1165G7 (4 cores - 8 threads - 2.8GHz Clock Speed)
RAM: 16GB
Language: Python
```

Use the for the CPU time formula:

$$\text{CPU time} = \text{Intructions count} \times CPI \times \text{Clock Cycle}$$

Since we use the same code for every counter $2^{10}, 2^{20}, 2^{30}, 2^{330}$ so the CPI will be the same. Lets consider the two cases 2^{30} and 2^{330} which is CPU Time$_A$ and CPU Time$_B$ respectively. There is only one intruction (XORing) in the body of the code. Therefore, the total instructions count for CPU Time$_A$ is 2^{30} instructions and for CPU Time$_B$ is 2^{330} instructions. Clock cycle will always be the same. In addition, this algorithm has the 0(n) run time.

$$38.3 = 2^{30} \times CPI \times \text{Clock Cyle}$$
$$\text{CPU Time}_B = 2^{330} \times CPI \times \text{Clock Cyle}$$

Divide both equations, we have:

$$\frac{38.3}{\text{CPU Time}_B} = \frac{2^{30}}{2^{330}}$$

Therefore

$$\text{CPU Time}_B = \frac{2^{330} \times 38.3}{2^{30}} = 7.8 \times 10^{91} seconds = 2.47 \times 10^{84} years$$

Part b

Python 2: Dividing with 4009

```
import time

def main():
    L = [2**(10),2**(20),2**(30)]
    for i in L:
        start = time.time()
        for counter in range(0,i+1,1):
            xor = counter / 4009
        end = time.time()
        print(end-start)
```

Use the same logic that we used to solve part a since this question, we have:

$$\frac{31.3 \times 2^{330}}{2^{30}} = 6.374 \times 10^{91} \text{ seconds} = 2.02 \times 10^{84} years$$

```
❯ ./Xor
Time to run 1024 is 0.0000302792
Time to run 1048576 is 0.0284295082
Time to run 1073741824 is 38.0975677967
❯ ./divide
Time to run 1024 is 0.0000255108
Time to run 1048576 is 0.0229318142
Time to run 1073741824 is 31.3022570610
```

Figure 1: Result when XORing and Dividing

Problem 2

1. $G_2(s \| t \| z) = G_1(s) \| (t \oplus z)$
   G_2 is not a secure PRG for the following reasons:
   (a) $G_1(s)$ is a secure PRG
   (b) $t \oplus z$ is not random since z is random but t is not necessarily random; therefore, half of the string could be not necessarily random.

   Therefore, the attacker could ignore the $G_1(s)$ and go after the $t \oplus z$ part. Hence, $G_2(s)$ is ***Not A Secure PRG***

2. $G_3(s) = G_0(\bar{s}) \oplus 1^{2n}$
   (a) $G_0(\bar{s})$ is a secure PRG since $G_0(s)$ is the PRG and $\bar{s}$ will not change the distribution of $0, 1$
   (b) $1^{2n} = 1^n$ which is a string of 1 that has length n. This is a fixed pattern.

   We basically flip the bit of $G_0(\bar{s})$. In orther words, $G_3(s) = \tilde{}G_0(\bar{s})$. We use arguement from 1 to say that $G_0(\bar{s})$ is a secure PRG. Therefore, $G_3(s)$ is a ***Secure PRG***
3. $G_4(s \parallel z) = ((s \parallel z) \oplus G_0(s)) \parallel G_0(s)$
   (a) $G_0(s)$ is a secure PRG
   (b) $s \oplus z$ will produce another random string since both s and z are random.
   (c) $((s \parallel z) \oplus G_0(s))$ is a secure PRG due to the above properties.
   (d) $((s \parallel z) \oplus G_0(s)) \parallel G_0(s)$ is a secure PRG since we concatenate property c with property a.

   Therefore, $G_4(s)$ is ***A Secure PRG***.
4. $G_5(s) = \mathsf{msb}(G_0(s)) \parallel G_0(s) \parallel G_1(z)$, where msb is the most significant bit.
   (a) $G_0(s)$ and $G_1(s)$ are a secure PRG
   (b) The attacker knows the algorithm that constructed G_0 but he doesn't know the seed. However, he still has a non-negligible chance of guessing $msb(G_0(s)$ which in this case is 50%

   Basically, the problem becomes,

   $$50\% \ \{0,1\} \parallel \text{PRG} \parallel \text{PRG}$$

   $G_5(s)$ is ***A Secure PRG*** since he only has 50% change of guessing the correct MSB of $G_5(s)$, and the rest of $G_5(s)$ are still psuedo random.

Problem 3

Part 1

1. In monoalphabetic substitution cipher, each letter is mapped with another letter to form the ciphertext. We have 26 letters in the alphabet. Therefore, the CPA attacker need to send 26 distinct letter to the oracle to obtain its correspond ciphertext.
2. No my answer won't change if we have a CCA attacker. Since each cipher text is mapped uniquely with a plaintext letter. Therefore, the attacker has to pick 26 distinct ciphertext letter and send them to the oracle to obtain the correspond plaintext.
3. Yes. The letter frequency attack will help since we can use the one-on-one fact that used in monoalphabetic substitution. Each plaintext letter is mapped uniquely with a ciphertext, so we can analyze the distribution of ciphertext letters and find the key.

Part 2

1. Since Alice and Sponge only exchange 100 messages, and the pad for those 100 messages are different. Therefore, it's **Secure** for Alice to do so since an attacker can't guess the key or any insight about Alice messages by Xoring two messages together.
2. No it's **not secure** to do so since each of message has length of 1 which mean the key also has the length of 1 in order for them to XOR. If a key has legth of 1 bit, the key space is extremely small in this case it only has two choices 0,1; therefore, an attacker could have a non-negligible chance to guess the key which is 50%.