Nirbhay Sharma (B19CSE114)
Cryptography
January 29, 2022

Cryptography Assignment

Question 1. *Given* : $G(s) : \{0,1\}^m \to \{0,1\}^n : m < n$

for a system to be PRG it should satisfy these conditions:

1. it should be length expanding
2. it should be efficient
3. it should be indistinguishable from a random source

since $G(s)$ is a secure PRG it must satisfy above 3 properties.

1.1. part1

consider the PRG : $G_1(s_1 \parallel s_2) = G(s_1) \wedge G(s_2)$

s_1 and s_2 are m bit strings and $G(s)$ is an n bit string

consider the input and output of $G_1(s_1 \parallel s_2)$ which is $\{0,1\}^{2m} \to \{0,1\}^n$ and we only know that $m < n$ but $2m$ is not necessarily less than n so it does not satisfies the first criteria of PRG i.e length expanding criteria and hence it is not secure.

1.2. part2

consider the PRG : $G_2(s_1 \parallel s_2) = G(s_1) \oplus G(s_2)$

s_1 and s_2 are m bit strings and $G(s)$ is an n bit string

consider the input and output of $G_2(s_1 \parallel s_2)$ which is $\{0,1\}^{2m} \to \{0,1\}^n$ and we only know that $m < n$ but $2m$ is not necessarily less than n so it does not satisfies the first criteria of PRG i.e length expanding criteria and hence it is not secure.

1.3. part3

consider the PRG : $G_3(s) = G(s) \oplus 1^n$

1. It is indeed length expanding because its input and output follows: $\{0,1\}^m \to \{0,1\}^n$ and from $G(s)$ we know that $m < n$ so it is length expanding

2. It is also efficient because $G(s)$ is efficient (since it is a PRG) and xor operation is also efficient so $G_3(s)$ is efficient

3. we can use contradiction to prove $G_3(s)$ is a PRG

towards the contradictions assume that $G_3(s)$ is not a secure PRG so $\exists$ a polynomial time distinguisher D which can distinguish between output of PRG and Uniform random distribution i.e

$|Pr[D(G_3(s)) = 1] - Pr[D(r) = 1]| \geq \epsilon$

but also consider that from $G_3(s)$ we can easily determine $G(s)$ by only doing xor with 1^n

$G_3(s) \oplus 1^n = G(s) \oplus 1^n \oplus 1^n = G(s)$

so it is equivalent to say that if distinguisher D can distinguish $G_3(s)$ from a random source it can also distinguish $G(s)$ from the random source and hence

$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| \geq \epsilon$

which is a contradicting statement that $G(s)$ is a secure PRG, which is not true

hence by contradiction we can say that $G_3(s)$ is a secure PRG.

1.4. part4

consider the PRG : $G_4(s) = G(s) \parallel G(s)$

1. It is indeed length expanding because its input and output follows: $\{0,1\}^m \to \{0,1\}^{2n}$ and from $G(s)$ we know that $m < n$ so $m < 2n$, hence it is length expanding

2. It is also efficient because $G(s)$ is efficient (since it is a PRG) and concatenation operation is also efficient so $G_4(s)$ is efficient

3. Let's try to construct a distinguisher D such that:
$D(x) = 1$ if $x[: n] == x[n :]$ i.e first n bit is same as last n bits
$D(x) = 0$ otherwise
so if the distinguisher D sees the first n bit and last n bit of an output, same then it will give me 1
so we can write the security equation as follows
$|Pr[D(G_4(s)) = 1] - Pr[D(r) = 1]|$
$Pr[D(G_4(s)) = 1] = 1$
$Pr[D(r) = 1] = \frac{2^n}{2^{2n}} = \frac{1}{2^n}$
so $|Pr[D(G_4(s)) = 1] - Pr[D(r) = 1]| = |1 - \frac{1}{2^n}| > \epsilon$
hence it is not a secure PRG.

Question 2. code provided in the zip file

Question 3.

3.1. CSS

CSS (Content scramble system) is a stream cipher which is used to encrypt DVD-videos, the stream cipher is based on two linear feedback shift registers (LFSR) one LFSR is LFSR-17 and another is LFSR-25 so they are basically of 17 and 25 bits and are components of the 42 bit state machine, it uses a key of 40 bits length and hence the internal state machine is initialized with 40bits key. a simple attack is brute force attack since key length is only 40bit but Due to its poor design its effective key length is left only to 16 bits and it can be breaked in 2^16 steps using correlation attacks. the correlation attacks are types of attacks for breaking stream cipher whose key system uses LFSR, it arises due to the poor choice of boolean function and it tries to brute force the smallest bit lfsr and tries to get the another LFSR which in turn get us the decrypted text.

3.2. Random

We know that random numbers are important and has applications in various fields, and they are important because of security reasons since most of the cryptographic algorithms uses random numbers to even generate keys and so on so it is highly important to generate truly random numbers. But there is one thing to wonder that if the computers are deterministic in nature then how the randomness in calculated in computers / systems. One simple example is when we write random.randint() in Python we get a random number and also each time we run this command we get random numbers. In this para we are going to talk about how these randomness comes in Unix operating system. One way to generate random numbers is to use Pseudo random number generators which can simulates random looking numbers, but given enough runs one can find out some patterns in the generation of random numbers but we want randomness to be completely unpredictable. One solution which is widely used is using Hardware random number generators. the simple flow is Hardware random number generators gathers random noise from environmental sources like mouse , keyboard some sample noise etc and store them in entropy pool, later when random numbers are needed linux's dev/urandom pulls random bytes from entropy pool and provide it when we call random.random() in python. there is another file called /dev/random which also pulls bytes from entropy pool but difference is that it can also blocks the data if sufficient randomness is not generated while /dev/urandom never blocks bytes. so if we want continuous random numbers /dev/urandom is recommended and if we are more concerned with security purposes then we should use /dev/random as it is more safer.