Pseudo Random Generator :-

→ It is a deterministic polynomial time algorithm G where the input is n bits turn into output $I(n)$.

$I(n) > n$ and Output (G) is computationally indistinguishable from uniform distribution.

PRG Definition :-

Let $l(\cdot)$ be a polynomial and let G be a deterministic polynomial time algorithm such that for any input $s \in \{0,1\}^n$, algorithm G outputs a string of length $l(n)$. We say that G is a pseudo random generator.

1) (Expansion:) For every n it holds that $l(n) > n$

2) (Pseudo randomness:) for all probablistic polynomial time distinguishers D, there exists a negligible function negl such that :-

$$|Pr[D(r)=1] - Pr[D(G(s))=1]| \leq negl(n)$$

r is chosen uniformly at random from $\{0,1\}^{l(n)}$

s " " " " " $\{0,1\}^n$

The function $l(\cdot)$ is called expansion factor of G.

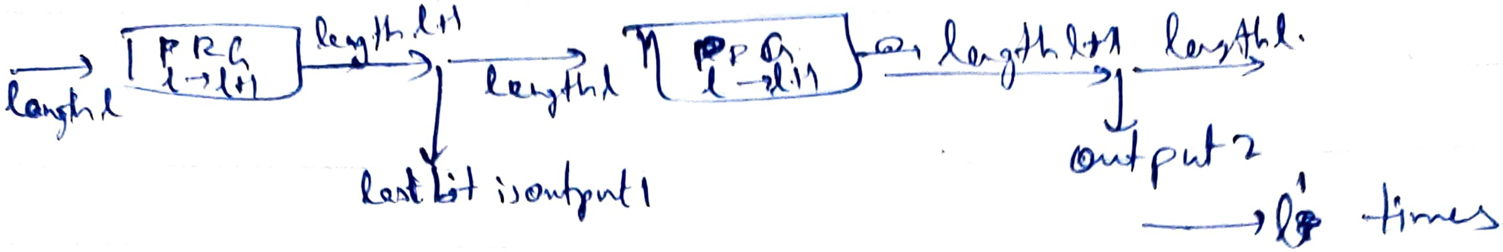
~~Designing~~ Using a PRG ~~according~~ designing ~~to a~~ Secure Encryption Scheme.

Pseudo randomizing the one-time pad.

- Generation :- Input $\Rightarrow 1^n$, Choose $k \leftarrow \{0,1\}^n$ uniformly at random and Output it as the key.
- Encryption :- Take the input $k \in \{0,1\}^n$ and message $m \in \{0,1\}^{l(n)}$ ~~output~~
  output $\Rightarrow c := G(k) \oplus m$
- Decryption :- input key $\Rightarrow k \in \{0,1\}^n$ and the cipher text $c \in \{0,1\}^{l(n)}$
  Output plain message $\Rightarrow m := G(k) \oplus c$.

Expanding the Expansion in PRG

→ Assume that there exists a pseudorandom generator with expansion factor $l(n) = n+1$. Then for any polynomial $p(\cdot)$ there exists a PRG with expansion factor $l(n) = p(n)$



Designing a single bit expansion PRG.

→ A function $f: \{0,1\}^* \to \{0,1\}^*$ is one-way if the 2 cond's holds

1) There exists a polynomial-time algorithm M_f computing f; that is, $M_f(x) = f(x)$ for all x.
2) For every probabilistic polynomial-time algorithm A, there exists a negligible function negl such that

$$\Pr[\text{Invert}_{A,f}(n) = 1] \leq \text{negl}(n)$$

~~Discr~~ Candidate One-way function:-

$$f_{p,g}(x) = g^x \bmod p.$$

→ we first define a string of length n by taking the input x of length n.
→ Now we convert the input into binary string

Now, we have to determine the discrete logarithm problem.

$$dlp = g^s \bmod$$

→ Then we write def for hardcore_prediate of dlp as MSB(x).

⇒ PRG = dlp + msb (hardcore predicate)

→ Then we encrypt to return a string of length n+1

```
for i in range (expfactor)
    t= self.prg (t)
    out = out + t [len(t) -1]
    t= t[:-1]
```

This is a simple PRG.