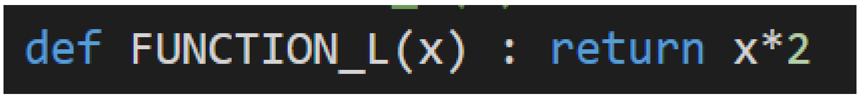
Pseudorandom Generators

Assumptions: I've designed a PRG which will take a seed (in binary) as input and return a pseudorandom number (in binary) of length = 2 * length of input seed. Also, for calculating the PRG it will also ask for prime and generator in integer form. Currently the function is just returning a length doubling PRG. We can change the function `FUNCTION_L(x)` to change the output length.



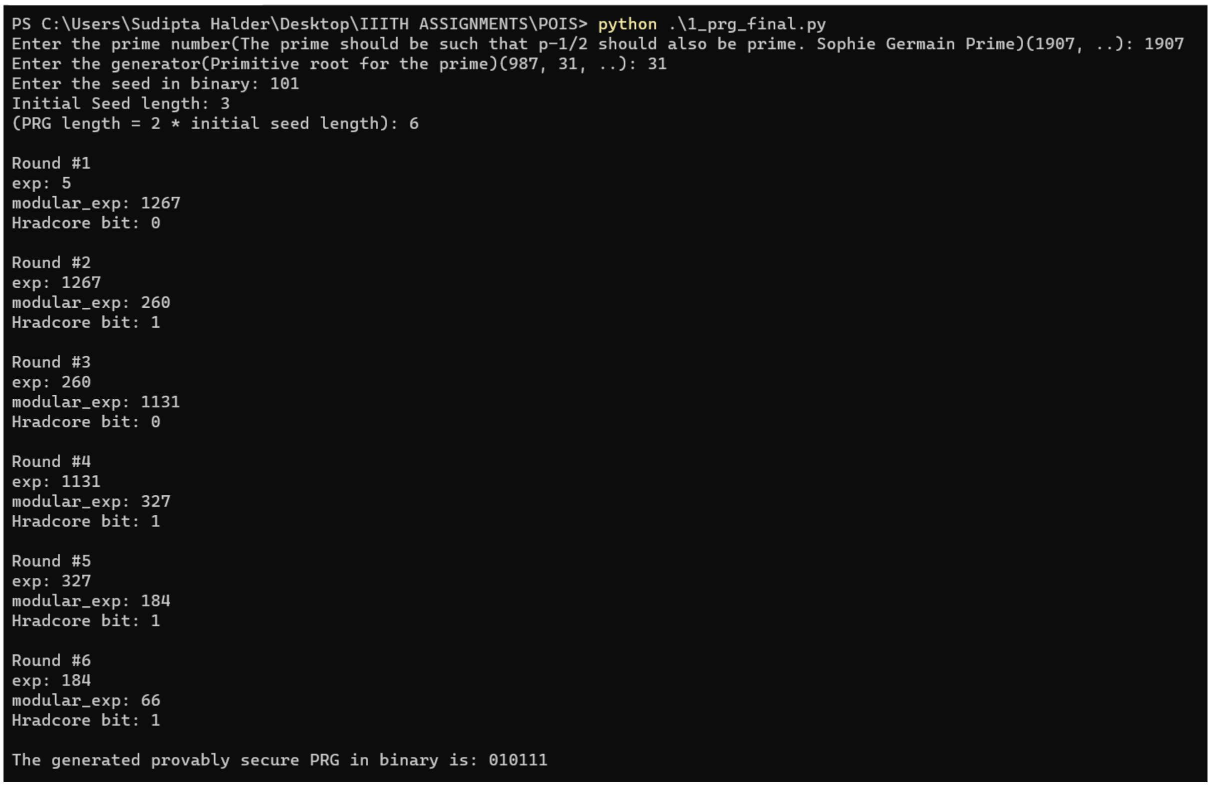
```
def FUNCTION_L(x) : return x*2
```

Input format:

1. It will ask for a safe prime number p in integer format.
2. It will ask for a generator (primitive root) of that prime in integer format.
3. It will ask to enter the seed in binary form.

Output Format:

1. It will return the length doubling pseudorandom number in binary.



```
PS C:\Users\Sudipta Halder\Desktop\IIITH ASSIGNMENTS\POIS> python .\1_prg_final.py
Enter the prime number(The prime should be such that p-1/2 should also be prime. Sophie Germain Prime)(1907, ..): 1907
Enter the generator(Primitive root for the prime)(987, 31, ..): 31
Enter the seed in binary: 101
Initial Seed length: 3
(PRG length = 2 * initial seed length): 6

Round #1
exp: 5
modular_exp: 1267
Hradcore bit: 0

Round #2
exp: 1267
modular_exp: 260
Hradcore bit: 1

Round #3
exp: 260
modular_exp: 1131
Hradcore bit: 0

Round #4
exp: 1131
modular_exp: 327
Hradcore bit: 1

Round #5
exp: 327
modular_exp: 184
Hradcore bit: 1

Round #6
exp: 184
modular_exp: 66
Hradcore bit: 1

The generated provably secure PRG in binary is: 010111
```

Working Flow:

1. Upon getting the input (prime, generator, seed), it will call the `generate_prg(prime, generator, initial_seed)` function.
2. Inside generate_prg() function, it'll calculate the length of the seed and determine the length of the output random number calling the function `FUNCTION_L(X)`.
3. It would convery the seed to integer format for binary.
4. Now, it'll run a for loop for the output length number of times. In each iteration, it would calculate: x_{i+1} = generator^{xi} mod prime, where x_0 is the initial seed. Now, we need to find the hardcore bit. In DLP, hardcore bit is the MSB. Please refer to the next two figures for MSB. So, for the next iteration, the calculated x_{i+1} will become the exponent. In this way, I've concatenated the hardcore bits one by one till the resultant pseudorandom number reaches output length. At the end, the psedorandom number is given as output in binary form.

Hardcore bit of discrete log [Blum-Micali '81]: Let p be an n-bit prime, $g \in \mathbb{Z}_p^*$. Define $B : \mathbb{Z}_p^* \to \{0, 1\}$ as

$$B(x) = msb(x) = \begin{cases} 0 & \text{if } x < p/2 \\ 1 & \text{if } x > p/2 \end{cases}$$

[1]

$$x_{i+1} = g^{x_i} \mod p.$$

The ith output of the algorithm is 1 if $x_i \leq \dfrac{p-1}{2}$. Otherwise the output is 0.

[2]

References

[1] https://crypto.stanford.edu/pbc/notes/crypto/hardcore.html

[2] https://en.wikipedia.org/wiki/Blum%E2%80%93Micali_algorithm