Sudipta Halder (2021202011)

Build a provably secure PRG

Let’s first understand the definition of PRG. The definition of PRG is as follows:

DEFINITION 3.14 *Let $\ell(\cdot)$ 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 $\ell(n)$. We say that G is a* pseudorandom generator *if the following two conditions hold:*

1. (Expansion:) *For every n it holds that $\ell(n) > n$.*
2. (Pseudorandomness:) *For all probabilistic polynomial-time distinguishers D, there exists a negligible function* negl *such that:*

$$\left|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]\right| \le \mathsf{negl}(n),$$

where r is chosen uniformly at random from $\{0,1\}^{\ell(n)}$, the seed *s is chosen uniformly at random from $\{0,1\}^n$, and the probabilities are taken over the random coins used by D and the choice of r and s.*

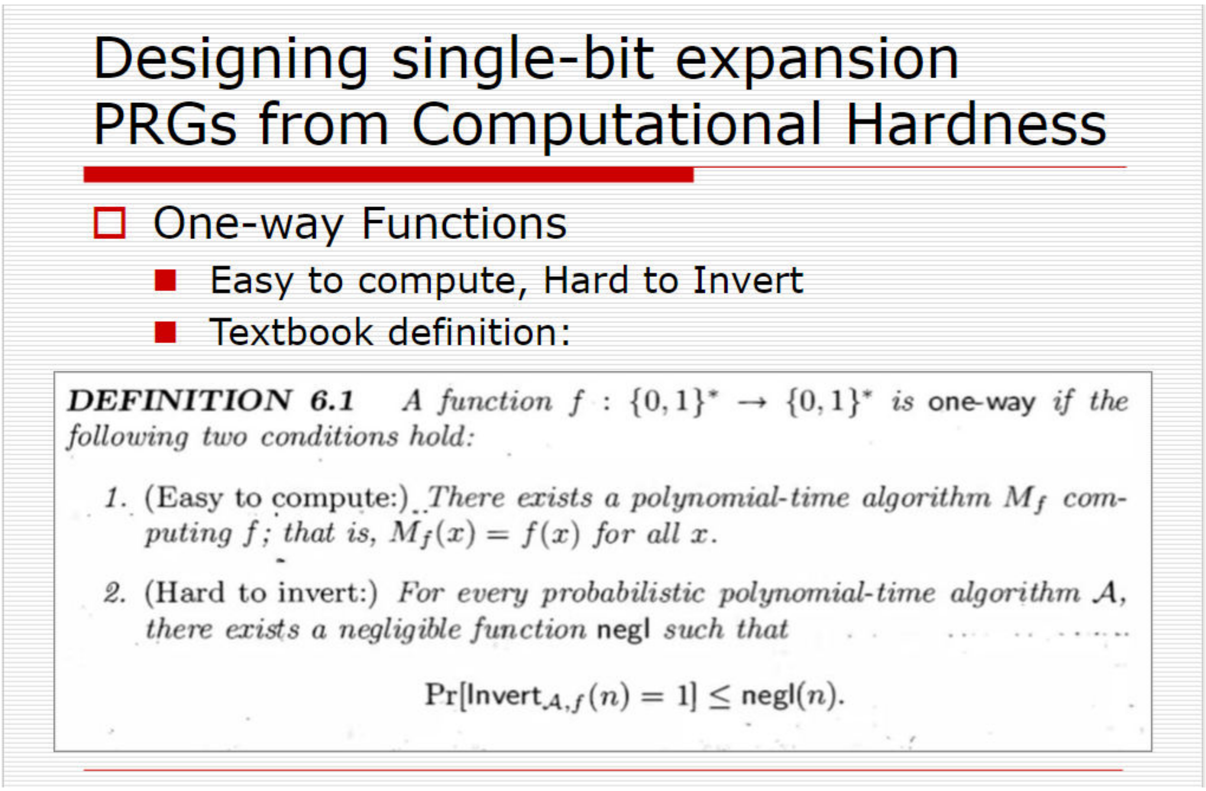
The function $\ell(\cdot)$ is called the expansion factor *of G.*

[1]

So, it is basically a deterministic polynomial time algorithm G, which takes input of n bits and outputs l(n) bits where:

1. l(n) > n and
2. Output of G is computationally indistinguishable from uniform distribution.

Now, let’s figure out how to design a single-bit expansion PRGS from computational hardness.



[1]

For that, we would use one-way functions. The idea is we can compute the one-way function easily but given the result it will be very to invert and get the arguments back.

I have used Discrete Logarithm Problem (DLP) as a one-way function here. Discrete Logarithm Problem (in Zp*): $f_{p,g}(x) = g^x \bmod p$.

So, given a prime(p), generator(g) and initial seed(x_0) we can compute the function and from that output we have to choose the hardcore bit.

HARDCORE PREDICATES

- Hardest bit of information about x to obtain from f(x)
- Textbook definition:

DEFINITION 6.5 *A function* hc : $\{0,1\}^* \rightarrow \{0,1\}$ *is a* hard-core predicate of a function f *if* **(1)** hc *can be computed in polynomial time, and* **(2)** *for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(f(x)) = \mathsf{hc}(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the uniform choice of x *in* $\{0,1\}^n$ *and the random coin tosses of* $\mathcal{A}$.

MSB(x) is a Hardcore predicate of Discrete Logarithm Problem

[1]

In DLP problem, msb bit is the hardcore bit. We will calculate the hardcore bit in the following way. If the result of the DLP function is <= (p-1)/2 then hardcore bit is 1, else hardcore bit is 0.

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

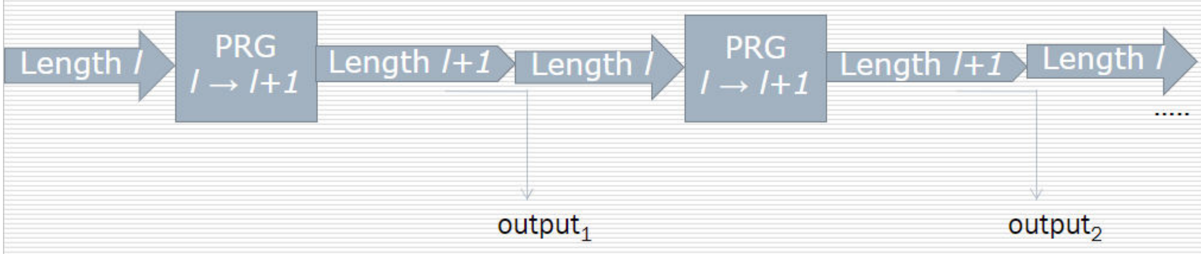
$$B(x) = msb(x) = \begin{cases} 0 & \text{if } x < p/2 \\ 1 & \text{if } x > p/2 \end{cases}$$

[2]

Now, we can see how we can generate how to create 1 bit for PRG randomly. Now, we have to figure out how to expand it n times.

The expansion is depicted in the diagram below. Basically, we keep on appending the hardcore bits till it reaches the desired output length.

THEOREM 6.8 *Assume that there exists a pseudorandom generator with expansion factor $\ell(n) = n+1$. Then for any polynomial $p(\cdot)$, there exists a pseudorandom generator with expansion factor $\ell(n) = p(n)$.*



1. Take the last bit from $l + 1$ length string for output
2. Apply l' times to get output of string l'

[1]

So, let's understand the full scenario once again.

Upon getting input for prime(p), generator(g) and initial seed(x0), we calculate $x_1 = g^{x0}$ mod p. Then calculate the hardcore bit dependent on (x_1, p). We use X_1 as exponent for the next iteration i.e., $x_2 = g^{x1}$ mod p. Again, calculating hardcore bit dependent on (x_2, p).

So, $x_i = g^{xi-1}$ mod p.

We keep on appending the hardcore bits till it reaches the desired output length.

In the end, we output this PRG of l(n) bits, where n = length of initial seed. Here l(n) = 2*n.

Now as DLP is a one-way function, our PRG is also secure because it is very hard to invert in polynomial time.

References

[1] J. K. a. Y. Lindell, Introduction to Modern Cryptography.

[2] B. Micali, "Hardcord bits," [Online]. Available:

https://crypto.stanford.edu/pbc/notes/crypto/hardcore.html