David Herel / KBE Published at Dec 15, 2021 Unlisted

Lab 4

David Herel

This is my notebook for lab4 - communication security class.⚾

Lab 4

> Today we will attack a very popular >PRNG: the Mersenne Twister. It is a >very good PRNG for applications such as physics >simulations, statistics, etc. But in >its original version, it is not designed for cryptographic >applications, and we are going to see >why.

> All integers below (except indices) >are unsigned 32-bit integers. We use >them in big-endian format: for instance, decimal 1234567890 is >the same as 0×499602d2.

Exercise 1: Write the Mersenne Twister

a., b., c. done

```
import copy

class MT:

    def __init__(self, seed):
        self.v = [0]*624
        self.index = 0
        self.w,self.n,self.m,self.r = 32,624,397,31
        self.a = 0x9908B0DF
        self.u,self.d = 11, 0xFFFFFFFF
        self.s,self.b = 7,  0x9D2C5680
        self.t,self.c = 15, 0xEFC60000
        self.l = 18
        self.f = 1812433253
        self.seed(seed)

    def seed(self, seed):
        self.v[0] = seed
        for i in range(1, 624):
            self.v[i] = (self.f * (self.v[i-1] ^ (self.v[i-1] >> 30)) + i) & 0xffffffff
        self.index = self.n


    def twist(self, v_prev):
        v = v_prev
        for i in range(0, 623):
            x = (self.v[i] & 0x80000000) ^ (self.v[(i+1)%self.n] & 0x7fffffff)
            xA = x >> 1
            if (x & 0x00000001):
                xA = xA ^ self.a
            self.v[i] = self.v[(i+self.m) % self.n] ^ xA
        return v

    def temper(self, x):
        y1 = x  ^ ((x  >> self.u) & self.d)
        y2 = y1 ^ ((y1 << self.s) & self.b)
        y3 = y2 ^ ((y2 << self.t) & self.c)
        z  = y3 ^ (y3 >> self.l)
        return z

    def next(self):
        if self.index == self.n:
            self.v = self.twist(self.v)
            self.index = 0
        x = self.v[self.index]
        self.index+=1

        return self.temper(x)


mt = MT(123456789)
print(mt.v)
```

```
[123456789, 2139825738, 2037464729, 1515522555, 3820599718, 3961686814, 2950588279, 2555342384, 3031251906, 2677166793, 714493217, 1148851984, 670708417, 2049
```

Task is done till point c. We got the correct vector state: `[123456789, 2139825738, 2037464729, 1515522555, ..., 3075821708]`

```
for i in range(0, 4):
    print(mt.next())
```

```
2288500408
4254805660
2294099250
56498137
```

All tasks from exercise 1. are done. I got correct sequence of random numbers: `2288500408, 4254805660, 2294099250, 56498137` .

Exercise 2. and 3.

Unfortunately I did not have a time to complete it.