Assignment 2

Q.2

(a) $X_{n+1} = (aX_n) \bmod 2^4$

Maximum period = 2^{4-2}
= 4

7
$5X_n \bmod 2^4$,
5

(b) a = 5, 11

(c) Seed must be odd.

Q.4

$X_{n+1} = (6X_n) \bmod 13$

Sequence : 1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1, ...

$X_{n+1} = (7X_n) \bmod 13$

Seq : 1, 7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1, ...

Because of the patterns seen in 2nd half of the latter sequence, most people would consider it to be less random than the first sequence.

(8.6) Use a key of length 255 bytes The first two bytes are zero,
$K[0] = K[1] = 0$

Thereafter we have
$K[2] = 255$, $K[3] = 254$, . . . ,
$K[255] = 2$.

This key will leave S unchanged.

(8.7) (a) Let us ~~state~~ consider i, j, S which requires
$$8 + 8 + (256 * 8) = 2064 \text{ bits.}$$

(b) The number of states is
$$(256! \times 256^2) \times 2^{1700}$$

Therefore 1700 bits are required.

8.8 (a) By taking the first 80 bits of $v \| c$ we obtain the intialization vector v. Since v, c, k are known, the message can be recovered by computing $RC4(v \| k) * c$.

(b) If adversary observes the $v_i = v_j$ for distinct i, j than he/ she

knows that the same key stream was used to encrypt both mi and mj.
In this case, the messages mi and mj may be vulnerable to the type of cryptanalysis carried out in above part (a).

(c) Since the key is fixed, the key stream varies with the choice of the 80 bit V, which is selected randomly. Thus after approximately messages are sent, we expect the same V and hence the same key stream, to be used more than once.

(d) The key K should be changed sometime before 240 messages are sent.

8.5 Algorithm

1. Generate 'n' number of any random numbers

2. Finding GCD for pairs in random numbers using Euclids algorithm. (for 'n' numbers, we ~~have~~ will see n/2 pairs)

3. Find probability of GCD = 1 (wherever this occurs)

4. Estimate π = sqrt(6/(probability gcd as 1)) calculated in 3

Program

```
#include <random>
#include <iostream>
#include <chrono>
int ~~main()~~ number generator( )
{
    unsigned int num ;
  unsigned int seed = chrono::system_clock::now
                              time_since_epoch()


      minstd_rand0 generator (seed);
    // standard linear congruential engine

          num = generator();

        return num;

}


int euclids_algo( int n, int m)
{
        ~~return (m!=0) ? (gcd(m, n%m)) : n~~

        if (n == 0)
              return m.
        return euclid's_algo(n%m, n);

}
```

```
int  square root( int n)
{
   int sq = sqrt(n)
      return sq;
 }


int main( )
{
      float phi;        int val;
~~int n = number_generator();~~
   int probability;
   int n;
   int  random[100];  // array of random
   int   GCD [50];                  numbers.
        cin >> n;   // number of nos.
for (int i=0;  i<n;  i++)
    random[i] = number_generator();

for( int i=0;  i<n;  i+=2)
{
    GCD[i] = euclids_algo(random[i],
                          random[i+1]
}

for (int i=0;  i< n/2;  i++)
{
    if (GCD[i] == 1)
       num_1 ++;
 }
```

```
probability = $\frac{num-1}{(n/2)}$;

val = $\frac{6}{probability}$;

phi = squareroot( val);

~~if((phi <= 3)~~
cout << " π value is" << phi

return 0;
```