Department of Electrical Engineering, IIT Bombay
EE 720 - An Introduction to Number Theory and Cryptography

Term Paper Report

Prepared by:
Shivang Tiwari (190040112)
Deven Bansal (190070019)
Arpit Sahu (190070010)
Akash Yadav (190070004)
Shivam Modi (190070059)
Instructor: Prof. Virendra Sule
Date: November 25, 2022

1 Description of the work done

We have followed the term paper to find the local inverse of a sequence using Berlekamp Massey algorithm. We will use the encryption map to demonstrate our work done in the project. Decryption map is similar.

These are the steps taken by us to get to the results:

1. Code the formula for encryption map in SAGE. The code for this is shows in the results section.
2. After generating the sequence $\{y, F(y), ..., F^{(2M-1)}(y)\}$, the sequence is split into l binary sequences where each sequence is the sequence corresponding to the bits at that position.

```
def get_bit_sequence(sequence,bit):
  res = []
  for x in sequence:
    b = 0
    if (int(x) & (1 << bit)) != 0:
      b = 1
    res.append(b)
  return res

```

   Now Berlekamp Massey is used to find the minimal polynomial of all sequences and their LCM is taken.

```
def get_minimum_polynomial(sequence,bit_length):
  res = None
  for j in range(bit_length):
    bit_seq = []
    for i in range(len(sequence)):
      bit = 0
      if (int(sequence[i]) & (1 << j)):
        bit = 1
      bit_seq.append(Mod(bit, 2))
    min_poly = berlekamp_massey(bit_seq)
    if res == None:
      res = min_poly
    else:
      res = lcm(res,min_poly)
  return res

```

3. Now the local inverse is calculated using the formula provided in the term paper.

$$x = \frac{1}{a_0}\left[F^{(m-1)}(y) - \left(\sum_{i=1}^{m-1} a_i F^{(i-1)}(y)\right)\right]$$

   Code for the same is as follows

```
def local_inverse(F,poly):
  a = poly.list()
  m = len(a) - 1
  if m > len(F):
    return None
  x = F[m-1]
  if(a[0] == 0):
    return None
  for i in range(1,m):
    x += a[i] * F[i-1]
  return x
```

The local inverse is calculated for each of the bit sequences and the inverse gives the bit at that position for the inverse of the whole encryption function.

4. So the entire run is as follows

```
def run_encryption(p,q,e,sample_size):
  n = p * q
  l = 1 + math.floor(math.log(n,2))
  M = l * l
  x_list = random.sample(range(0,n-1), sample_size)
  correct = 0
  total = 0
  for i in range(len(x_list)):
    x = x_list[i]
    x = Mod(x,n)
    c = x^e
    F = generate_E(c,e,n,2*M)
    fail = False
    poly = get_minimum_polynomial(F,l)
    ans = 0
    for bit in range(l):
      bit_sequence = get_bit_sequence(F,bit)
      inv = local_inverse(bit_sequence,poly)
      if inv == None:
        fail = True
        break
      if int(inv) == 1:
        ans += 1 << bit
    if fail:
      ans = 0
    ans = Mod(ans,n)
    if c == ans^e:
      correct += 1
    total += 1
  print("Got ",correct,"/",total," correct")
```

2 Results

1. Code for the formula of E and D as written in SAGE is as follows.

```
def generate_E(c,e,n,seq_len):
  sequence = [Mod(c,n)]
  for i in range(seq_len - 1):
    sequence.append(sequence[-1]^e);
  return sequence

def generate_D(m,c,n,seq_len):
  sequence = [Mod(m,n)]
  c = Mod(c,n)
  for i in range(seq_len - 1):
    sequence.append(c^sequence[-1]);
  return sequence
```

2. The results are as follows

Sr no	(p, q, e)	n	l	$\|S_y\|$	$\nu(E)$	Time(seconds)
1	(613, 449, 5)	275237	19	10000	1.0	442.46
2	(907, 503, 23)	456221	19	12000	0.02	709.1
3	(9421, 10369, 13)	97686349	27	20000	0.68	3043.44
4	(13171, 12421, 23)	163596991	28	25000	0.01	4900.46
5	(41737, 39769, 6)	1659838753	31	30000	0.05	8018.37
6	(63409, 59011, 5)	3741828499	32	50000	0.002	15078.12

Table 1: Results for Encryption map

Sr no	(p, q)	n	l	$\|S_y\|$	$\nu(D)$	Time(seconds)
1	(601, 521)	313121	19	10000	0.03	465.28
2	(569, 683)	388627	19	10000	0.02	471.2
3	(2857, 3739)	10682323	24	12000	0.001	1396.07
4	(15803, 10369)	163861307	28	20000	0.005	3977.4
5	(33581, 28961)	972539341	30	30000	0.01	7497.81
6	(63409, 59011)	3741828499	32	50000	0.003	15483.77

Table 2: Results for Decryption map

3 Conclusion

The encryption function is invertible using this method is certain cases. We even got a case of 100% inversion which is because the order of e is less that M. It is to be noted that we only chose values which actually had an inverse. In one case we also got 68% inversion. However usually the results are lower than 10%.

The decryption function inversion is very poor. Top result which we got was 3% success and even that for small n. For larger values of p and q, we didn't get results better than 2%.