EE720: Home Paper

Kaishva Chinta Shah, 200020066
Karrthik Arya, 200020068
Priyanshi Gupta, 200070061
Raavi Gupta, 200070064
Ronil Mandavia, 20D180020

November 25, 2022

1 Implementation of Algorithms

We first implemented the algorithm (explained in section 2.3) to find the minimum polynomial of a given vector sequence and its linear complexity in a field $\mathscr{F}_q$.

1.1 Miminum Polynomial using Berlekamp Massey

```
#getting the sequence
m = 4
y = 2
f = x + 13
seq = []
y = f(y)
maxlen = 0
for i in range(0, m):
    fy = bin(y).replace("0b","")
    maxlen = max(len(fy), maxlen)
    seq.append(fy)
    y = f(y)
seq = [seq[i].zfill(maxlen) for i in range(0, len(seq))]
print(seq)
A = [x for x in seq[0]]
S = np.array(A)
for i in range(1,m):
```

```
    A = np.array([x for x in seq[i]])
    S = np.vstack((S,A))
S = matrix(GF(2), S)
S = np.array(S)
def get_minimal_poly(S):
    f = berlekamp_massey(list(S[:,0]))
    l = S.shape[1]
    for i in range(1,l):
        g = berlekamp_massey(list(S[:,i]))
        f = lcm(f,g)
    return f, f.degree()
min_poly, lc = get_minimal_poly(S)
print("Minimal Polynomial is", min_poly)
print("Linear Complexity = ", lc)
```

Listing 1: Minimal polynomial using Berlekamp Massey

We use the dummy function f = x + 13 to create a sequence [seq] of numbers in their binary form. We then separate the bits of each binary number and convert the sequence of numbers into a numpy array for further computation.
The function get_minimal_poly(S) takes this numpy array as its input. We first find the minimum polynomial of the sequence formed by the 0^{th} bit of all the elements of the array using the berlekampmassey algorithm of sage math. We then find the minimum polynomial of the sequence formed by the i^{th} bit of all the elements of the array and update the minimum polynomial by taking its lcm with the minimum polynomial from the previous computation. We finally return the minimum polynomial of the vector sequence and the linear complexity, given by the degree of the minimum polynomial.

1.2 Finding the local inverse x

Here we use equation 6 provided in the home paper to estimate the local inverse x.

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

```
import random
random.seed(42)

def e_inv(p , q, e, y_r):
    m = int((log(p*q,2)+1)**2)
    f = lambda x: mod(x^e,p*q)
```

```
 corrects = 0
 corrects2 = 0
 evals = 0
 for i in range(y_r):
     seq = []
     y = random.randint(0, p*q)
     y = f(y)
     maxlen = 0
     for i in range(0, m):
         fy = bin(y).replace("0b","")
         maxlen = max(len(fy), maxlen)
         seq.append(fy)
         y = f(y)
     seq = [seq[i].zfill(maxlen) for i in range(0, len(seq))]
     A = [x for x in seq[0]]
     S = np.array(A)
     for i in range(1,m):
         A = np.array([x for x in seq[i]])
         S = np.vstack((S,A))
     S = matrix(GF(2), S)
     S = np.array(S)
     min_poly, lc = get_minimal_poly(S)
     if(lc>m):
     evals+=1
     X = S[lc-1]/min_poly[0]
     for i in range(1,lc):
         X-=min_poly[i]*S[i-1]/min_poly[0]
     if(f(to_int(X.astype("uint8"))) == to_int(S[0].astype("uint8"))):
         corrects+=1
     elif(f(f(to_int(X.astype("uint8")))) == to_int(S[1].astype("uint8"))):
         corrects2+=1
 return (evals, corrects, corrects2)

 def d_inv(p , q, d, y_r):
 m = int((log(p*q,2)+1)**2)
 f = lambda x: mod(d^x,p*q)
 corrects = 0
 corrects2 = 0
 evals = 0
 for i in range(y_r):
     seq = []
     y = random.randint(1, p*q)
```

```
        y = f(y)
        maxlen = 0
        for i in range(0, m):
            fy = bin(y).replace("0b","")
            maxlen = max(len(fy), maxlen)
            seq.append(fy)
            y = f(y)
        seq = [seq[i].zfill(maxlen) for i in range(0, len(seq))]
        A = [x for x in seq[0]]
        S = np.array(A)
        for i in range(1,m):
            A = np.array([x for x in seq[i]])
            S = np.vstack((S,A))
        S = matrix(GF(2), S)
        S = np.array(S)
        min_poly, lc = get_minimal_poly(S)
        if(lc>m):
            continue
        evals+=1
        X = S[lc-1]/min_poly[0]
        for i in range(1,lc):
            X-=min_poly[i]*S[i-1]/min_poly[0]
        if(f(to_int(X.astype("uint8"))) == to_int(S[0].astype("uint8"))):
            corrects+=1
        elif(f(f(to_int(X.astype("uint8")))) == to_int(S[1].astype("uint8"))):
            corrects2+=1
    return (evals, corrects, corrects2)
```

Listing 2: Calculating the local inverse

The following table illustrates the results obtained by us for the home paper.

- eval: The number of cases that follow the rule linear complexity of the minimum polynomial is less than n.
- C1: Number of cases where we exactly get y = F(x)
- C2: There are certain cases where the first term, F(x), does not match y but the second term, F(y), and F(F(x)) turn out to be equal. C2 takes into account these cases.
- E: m^e mod n

- D: c^d mod n

where e and d satisfy ed = 1 mod $\phi(n)$

p	q	e	n	l	Sample size	eval	C1	C2	$v(E)$	Time taken
397	5	5	1985	11	15000	15000	15000	0	1	226.543
397	5	23	1985	11	15000	15000	15000	0	1	209.835
59	5	5	295	9	10000	10000	10000	0	1	94.619
59	5	7	295	9	10000	10000	10000	0	1	83.492
523	7	5	3661	12	20000	20000	20000	0	1	416.423
523	89	5	46547	16	20000	8430	8430	0	0.4215	846.186

Table 1: Results of encryption function E

Here we can observe that for small values of p, q and e we get 100 percent accuracy and as the value of e increases keeping the rest of the parameters same the time taken for computations decreases.

p	q	d	n	l	Sample size	eval	C1	C2	$v_1(D)$	$v_2(D)$	Time taken
397	5	317	1985	11	15000	15000	1307	991	0.087	0.153	218.766
397	5	551	1985	11	15000	15000	1458	766	0.097	0.148	238.428
59	5	93	295	9	10000	10000	1624	1138	0.1624	0.2762	86.976
59	5	199	295	9	10000	10000	2428	964	0.2428	0.3192	80.454
523	7	1253	3661	12	20000	20000	1547	815	0.077	0.1181	404.909
523	89	36749	46547	16	20000	18620	100	180	0.009	0.014	4019.32

Table 2: Results of decryption function D

$v_1(D)$ is calculated considering only C1 and $v_2(D)$ is calculated using C1 and C2 for the decryption table. It can be seen that the frequency for decryption is much less than for encryption.