Erikson Sodergren CS462-01

1. After PC-1 the swapped bit is at position 8
   a. Round 1: position 7 PC-2: position 20 S4
   b. Round 2: position 6 PC-2: position 10 S2
   c. Round 3: position 4 PC-2: position 16 S3
   d. Round 4: position 2 PC-2: position 24 S4
   e. Round 5: position 28 PC-2: position 8 S2
   f. Round 6: position 26 PC-2: position 17 S3
   g. Round 7: position 24 PC-2: position 4 S1
   h. Round 8: position 22 PC-2: position unused Snone
   i. Round 9: position 21 PC-2: position 11 S2
   j. Round 10: position 19 PC-2: position 14 S3
   k. Round 11: position 17 PC-2: position 2 S1
   l. Round 12: position 15 PC-2: position 9 S2
   m. Round 13: position 13 PC-2: position 23 S4
   n. Round 14: position 11 PC-2: position 3 S1
   o. Round 15: position 9 PC-2: position unused Snone
   p. Round 16: position 8 PC-2: position 18 S3
   q. For decryption the order would be reversed, so decryption round1 would have S3 affected, round 2 none, round 3 S1 and so on.
2. Given that decryption uses the same key schedule except in reverse, and that L^d_0/R^d_0 during decryption are equal to R_{16}/L_{16} : the final encryption creating $R_{16} = L_{15}$ * key16 and $L_{16} = R_{15}$ would clearly be the same as the first step in decryption, where key1^d = key16, so since the key is used in a reversible operation, $R^d_1 = L^d_0$ * key1^d = R_{16} * key16 = L_{15}, and $L^d_1 = R^d_0 = R_{15}$

3.
   a. Inverse of 25mod1033 is 124mod1033

i	q	r	s	t
0		1033	1	0
1	41	25	0	1
2	3	8	1	-41
3	8	**1**	**-3**	**124**
4		0	24	-1033

b. Inverse of 25mod1034 is 455mod1034

i	q	r	s	t
0		1034	1	0
1	41	25	0	1
2	2	9	1	-41
3	1	7	-2	83
4	3	2	3	-124
5		**1**	**-11**	**455**

c. 25mod1035 has no inverse

i	q	r	s	t
0		1035	1	0
1	41	25	0	1
2	2	10	1	-41
3	2	**5**	**-2**	**83**
4		0	5	-207

d. Inverse of 25mod1036 is 373mod1036

i	q	r	s	t
0		1036	1	0
1	41	25	0	1
2	2	11	1	-41
3	3	3	-2	83
4	1	2	7	-290
5		**1**	**-9**	**373**

4. Python code to use the EEA to find the multiplicative inverse of 123456 mod 7111111:

```
s = [1, 0]
t = [0, 1]
i = 1
r = [7111111, 123456]
q = [None]
while(True):
```

```
i+=1
r.append(r[i-2]%r[i-1])
q.append((r[i-2] - r[i])/r[i-1])
s.append(s[i-2] - q[i-1]*s[i-1])
t.append(t[i-2] - q[i-1]*t[i-1])
if(r[-1] is 0):
    if r[-2] is 1:
        print("multiplicative inverse of %s mod %s is %s"%(r[1], r[0], t[-2]))
    else:
        print("no multiplicative inverse exists for %s mod %s"%(r[1], r[0]))
    break
```