SSN LAB ASSIGNMENT: SYMMETRICAL ENCRYPTION

1 NOV. 2016

ALI ABDULMADZHIDOV

1.DES

```
setKey(0101010101010101)
encryptDES(416c690000000000)
  IP:   L0=07000205, R0=00060600
  Rnd1  f(R0=00060600, SK1=00 00 00 00 00 00 00 00 ) = c8b8cbfc
  Rnd2  f(R1=cfb8c9f9, SK2=00 00 00 00 00 00 00 00 ) = ac9062f5
  Rnd3  f(R2=ac9664f5, SK3=00 00 00 00 00 00 00 00 ) = ef63339a
  Rnd4  f(R3=20dbfa63, SK4=00 00 00 00 00 00 00 00 ) = e4c0c3ed
  Rnd5  f(R4=4856a718, SK5=00 00 00 00 00 00 00 00 ) = b688d6c5
  Rnd6  f(R5=96532ca6, SK6=00 00 00 00 00 00 00 00 ) = 078e4a35
  Rnd7  f(R6=4fd8ed2d, SK7=00 00 00 00 00 00 00 00 ) = 8731e059
  Rnd8  f(R7=1162ccff, SK8=00 00 00 00 00 00 00 00 ) = ec5804d6
  Rnd9  f(R8=a380e9fb, SK9=00 00 00 00 00 00 00 00 ) = c5907157
  Rnd10 f(R9=d4f2bda8, SK10=00 00 00 00 00 00 00 00 ) = 498041a2
  Rnd11 f(R10=ea00a859, SK11=00 00 00 00 00 00 00 00 ) = eb0493e5
  Rnd12 f(R11=3ff62e4d, SK12=00 00 00 00 00 00 00 00 ) = 8459728c
  Rnd13 f(R12=6e59dad5, SK13=00 00 00 00 00 00 00 00 ) = 08057433
  Rnd14 f(R13=37f35a7e, SK14=00 00 00 00 00 00 00 00 ) = 45050caf
  Rnd15 f(R14=2b5cd67a, SK15=00 00 00 00 00 00 00 00 ) = 1c0ad75b
  Rnd16 f(R15=2bf98d25, SK16=00 00 00 00 00 00 00 00 ) = 4aaf3da9
  FP:   L=ff950aac, R=31f6753d
returns ff950aac31f6753d
```

Addition

At start we have 64 bits of input (416c690000000000 hex = "Ali" ASCII) and 64 bit key (0101010101010101).

We convert our text to binary:

```
0b1000001011011000110100100000000000000000000000000000000000000000 =
0x416c690000000000
```

Then we convert our hex key to binary:

```
   0b1000000010000000100000001000000010000000100000001 =
0x0101010101010101
```

Next step will be IP of our input text:

```
![sbox](/home/mrzizik/screenshots/sbox.png)
```

I wrote a small script for doing this in python:

```
1     ip = (58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62,
   54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33,
   25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5,
   63, 55, 47, 39, 31, 23, 15, 7)
2     input = [0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
   0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3     output = []
4     for i in xrange(64):
5      output.append(input[ip[i]-1])
6     print output
```

After that we have got such permuted list

```
   0000011100000000000000010000001010000000000000110000001100000000
```

After that we do IP and divide key to two parts 28 bits each:

```
![keypermut](/home/mrzizik/screenshots/keypermut.png)
```

```
   left = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0]
```

```
   right = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0]
```

Then we shift our key, but in our situation (all are zeros) shifting wouldn't change anything.

For some rounds we make shift 2, not one.

Then we make second permutation of key, that's also useless for us.

We have round 1 key. In that manner we generate keys for all rounds.

Then we divide our input to 2 parts and go to first round of encryption

```
![cipher](/home/mrzizik/screenshots/cipher.png)
```

```
   Rnd8 f(R7=1162ccff, SK8=00 00 00 00 00 00 00 00 ) = ec5804d6
```

2. Firstly we make internal permutation of key, while we pemute it by special box and make it 56 bit instead of 64.

```
57      49      41      33      25      17      09      01
58      50      42      34      26      18      10      02
59      51      43      35      27      19      11      03
60      52      44      36      63      55      47      39
31      23      15      07      62      54      46      37
30      22      14      06      61      53      45      37
29      21      13      05      28      20      12      04
```

   then we spit it in two parts each 28 bit.

   Then we shift both of them left (1,2,9,16 rounds - 1 bit, else - 2 bits) and send to compressing P box

```
14      17      11      24      01      05      03      28
15      06      21      10      23      19      12      04
26      08      16      07      27      20      13      02
41      52      31      37      47      55      30      40
51      45      33      48      44      49      39      56
34      53      46      42      50      36      29      32
```

   that makes it 48 bit from 56 for each round of encryption.

   Wrote with explantaions in 1 answer

3. It's because the first step of key generation, when we make 56bit key from 64bit one.

   In our example we have in binary:

```
0000 0001 0000 0001
0000 0001 0000 0001
0000 0001 0000 0001
0000 0001 0000 0001
```

   And the first step cuts out 8, 16, 24, 32, 40, 48, 56, 64, and left us with all zero's in key.

2.AES

1. Diffusion is reached by 16 rounds of SubBytes, ShiftRows and Mix Collumns with applying round key. Also the last permutation grant more diffusion
2. Confusion elements are rcon and sbox of key scheduling, as all key schedulling process. It makes one bit modification in input key become huge change in key, that we are going to use and in output ciphertext.

   Many rounds of crypting when output of round becomes input of next, and diffusion elements also grants more confusion.

3.Bonus: RC4

After bruteforce attack found that keys are "adwtg" and "495706". There is book of Darwin encrypted in them.

```
The Project Gutenberg EBook of On the Origin of Species, by Charles Darwin

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: On the Origin of Species
       1st Edition

Author: Charles Darwin

Release Date: Release Date: March, 1998 [EBook #1228]
Posting Date: November 23, 2009

```

1. (a) How did you identify the encrypted files ?

   Firstly i just bruteforced all of them by lowercase alphabet, and after crackin one, tried other with digits, but then i opened them in python and saw that 2 encrypted files have different pattern of content

```
ŵE"ŵIov ŵC ŵb ŵus ŵet ŵŵŵnl
</S
%ŵŵŵŵŵ*ŵW1ŵŵŵ\ŵŵ,ŵ7ŵŵ6ŵWŵŵŵl=Fc2ŵcgŵŵjŵ*bŵŵŵŵ/c)ŵŵE)ŵŵ
ŵŵ
   ŵŵŵ>ŵŵ3ʒI4        v<ŵBhŵvA
                      ,Mŵ
Eŵŵ)ŵ}n]LAŵs
ŵ:ŵŵŵŵeŵ;d/wŵ!MPwŵMŵŵ?Pŵŵŵŵ Fŵ4"ŵ□ŵ2>Jᵲ V1`ndŵXR6ŵŵyQŵŵf覺
ŵ8ŵŵ | ŵŵŵrvŵŵŵRŵQ)ŵɒT#xŵŵtŵŵŵŵŵŵ^D
ŵ5ŵŵŵŵŵkꚋ{*ŵŵŵŵŵŵ
ŵŵ1vŵ4ŵŵŵŵŵh3فUŵŵkŵ*ŵŵ^ŵ:ŵŵ!6>ŵŵX<Jŵ=ŵDN

ŵKŵŵŵ&gŵ0ŵYŵeIŵcŵ3(ŵXŵŵ[ŵo:ŵ2ŵMŵŵŵGŵŵ$ŵŵ
          /Jòŵ Iŵŵ|ŵ0`Rŵŵŵŵhŵ)□ŵŵ0P]Ŭŵ/
ŵŵŵŵŵŵŵoŵŵQŵ9ŵVŵŵŵdŵŵŵŵŵ6}ŵŵ^6-
ŵ7ŵŵŵŵŵ7CŵTVŵŵVŵŵŵŵ+ŵŵ^=,ŵ
ŵŵ8Fŵŵŵ Mŵŵŵ}"vŵ'T9~ŵŵŵ#ŵŵQ,ŵŵ{O
9ŵŵŵŵŵŵ                                  ŵZŵŵŵ]
       eMŵŵŵ@ŵmvŵŵŵ!ŵŵŵaŵŵŵIzŵŵŵŵŵŵ̦ŵŵŵ
                                          ŵKtŵŵŵŵAŵŵ″
4,ŵ+ŵŵ[ŕ;"ŵaŵŵŵxŵHAŵŵ?ŵŵŵ~ŵΓ2ŵaŵ
+GuBtŵI,ŵ*ŵ,ŵeŵҜ8ŵs]ŵbŵ"lNŵ8[ŵDŵ;]Cŵ&ŵ
                                   ŵŵŵŵTŵŵ~J̌t
T:ŵ2ŵzŵt}/Zdŵŵvŵ['ŵ(o|ŵŵxmŵŵŵŵŵglŵYŵ/ŵŵŵŵc!ŵŵ*ŵ$ŵGŵŵ
ŵжŵŵŵŵ(
```

But those with trash inside are like this:

```
ŵcPPdB^YR[AsEEVYZPKStqXW^[V|YAQQ~A^_\W_WdHPZ]UBZLwXPA[]FpQCD^V839:e[^K\v_^
XQFR_CCPPACTX^XZI^]RTWMGYVE]X@_\[ZJ@P]SBP@X<9VTXVGD]XG\GDCZTL\VZCD_YAJ[UGVE
m_DZYLW_AJQAWXER\MQFRNZK9:CVMF\YEBVQ\FE[RA\F]BX^M\UcEW_\WDtBLPWVUCTt\ZQ^BVQ
[ZXEUVS5?N]DYCP\JUs\XSVF^][Q[\QE@OBSEEVYZPKS^AP5?
4>=;g^LY\~]L]\CZPQ[[V`G]VPQC<9DL|PYEZXV839:pFCPZKr[VJY\GuREO\W9:
<9e]Y\UCTsYA\cV[l}wV[[yQCP_
i=;cXKAPZWwVLP~^ERUW\F
<9:2yXZWDRP]q^V_^K]4>=;>=cerelvre{~kif{vtl~adt}u}g~us|xsvze{rzk}wx}wsg`tp~}
f=5?4>=;>=hGVPERVSW@cDVyFJWXTA:2839:
<9:283{~g}vfyvzyzcavtqpj=;>=wg`xtgjpjqbgrcqzwwqycvabtwjtzqczyaqqbgemr~xuuxj
u}vt:2839:sJ{]XF\T@|TKCY_ux<9:2s\X\^DwS`XTeWLXXtRWYVSYRR[u]^_RRY[uEPj[SXVCQ
PJ=;>=y@M\_Cx^~_DAYYY{VaRKPXFSYVDqLFY_Tptcu]T^XU@nZ@UWTeW@WPe[R5?n[B]W839:
<9{w{}{~
```

2. (b) What is the effective key strength for each of the keys?

   Effective key strength for key containing 6 digits is 10^6

   Effective key strength for lower case character key with length 5 is 26^5

3. (c)

```
def worker(base):
        #read 64KB from the file
        data = open(FILE_NAME, 'rb').read(2**16)
        #generate all the strings of KEY_LENGTH length  and check them
        #We know prior that the key starts with a. Remove the next two
lines for generic behavior
        if string.ascii_lowercase in ALPHABET:
    base = tuple(['a']) + base
        startms = time.time()*1000.0
        coll = len(ALPHABET)**(KEY_LENGTH-len(base))
        print coll
        for i in itertools.product(ALPHABET,    repeat=KEY_LENGTH-
len(base)):
    check(''.join(base + i), data)
        endms = time.time()*1000.0
        print (1000/((endms-startms)/coll))
```

2771 attempts

4. (d)

   I set up CPU_COUNT to 3 and changed serial method to

   ```
   worker(parallel())
   ```

   It gave me 2771*3 attempts in sec.

5. (e)

   $(100^6)/2771 = 360880548$ seconds or something like 11 years in one thread