popular crypto systems

```
## My cryptography practice sheet
```

Practice of the Caesar cipher for encryption and decryption

```
## encrypt a given word using the caesar cipher
## Computation is done modulo 3
## Note use of ord keyword to return ascii values. ASCII valus for
"A" to "Z" are 65 to 90
## To get values between 0 and 65 subtract 65. Letters of alphabet
are encoded as 0 to 25
##
pl = "WITHDRAWONETHOUSANDDOLLARS" # plain text

#pln = [ord(x)-65 for x in pl ]
```

```
def str2lst(s):
    '''
    return integer representation of the letters
    The values should be between 0 and 25

    input: string of characters.
    output: integers representation (0 to 25) of the string input.
    '''
    return [ord(x)-65 for x in s]
```

```
def lst2str(lst):
    '''
    convert values after encryption to letters.
    Note 65 is added to the character representation.

    input: integer values between 0 and 25. Type is list
    output : string.

    '''
    return ''.join([chr(int(x)+65) for x in lst])
```

```
def caesar(pl):
    '''
    encryption function
    input: string to be encrypted

    '''
    pln = str2lst(pl);
    pln2 = [(x+3)%26 for x in pln];
    return (lst2str(pln2))
```

```
encrypt = caesar(pl)
print(encrypt)
```

```
    ZLWKGUDZRQHWKRXVDQGGROODUV
```

```
def decrypt_caesar(encrypt):
    '''
    decryption function
    input : string to be decrypted
    '''
    pln = str2lst(encrypt);
    pln2 = [(x-3)%26 for x in pln];
    return (lst2str(pln2))
```

```
decrypt_caesar(encrypt)
```

```
    'WITHDRAWONETHOUSANDDOLLARS'
```

Practice of the Translation Cipher

```
# change n =3 to n=13 to get the ROT13 cipher
# These are simple generalization of the Caesar cipher, in which
encryption involves shifting
# letters along the alphabet by a prearranged value n
```

```
def trans(pl,n):
    '''
    same as that of encryption above
    But now instead of value of n=3 we can provide any value
    '''
    pln = str2lst(pl)
    pln2 = [(x+n)%26 for x in pln]
    return lst2str(pln2)
```

Use of brute force to break the systems

```
## Let us show the vulnerability of this systems by using brute
force to break the system
encrypt # show cipher text
```

```
    'ZLWKGUDZRQHWKRXVDQGGROODUV'
```

```
for i in range(25): print i , trans(encrypt,i)
```

```
    0 ZLWKGUDZRQHWKRXVDQGGROODUV
    1 AMXLHVEASRIXLSYWERHHSPPEVW
    2 BNYMIWFBTSJYMTZXFSIITQQFWX
    3 COZNJXGCUTKZNUAYGTJJURRGXY
    4 DPAOKYHDVULAOVBZHUKKVSSHYZ
    5 EQBPLZIEWVMBPWCAIVLLWTTIZA
    6 FRCQMAJFXWNCQXDBJWMMXUUJAB
    7 GSDRNBKGYXODRYECKXNNYVVKBC
    8 HTESOCLHZYPESZFDLYOOZWWLCD
    9 IUFTPDMIAZQFTAGEMZPPAXXMDE
    10 JVGUQENJBARGUBHFNAQQBYYNEF
    11 KWHVRFOKCBSHVCIGOBRRCZZOFG
    12 LXIWSGPLDCTIWDJHPCSSDAAPGH
    13 MYJXTHQMEDUJXEKIQDTTEBBQHI
    14 NZKYUIRNFEVKYFLJREUUFCCRIJ
    15 OALZVJSOGFWLZGMKSFVVGDDSJK
    16 PBMAWKTPHGXMAHNLTGWWHEETKL
    17 QCNBXLUQIHYNBIOMUHXXIFFULM
    18 RDOCYMVRJIZOCJPNVIYYJGGVMN
    19 SEPDZNWSKJAPDKQOWJZZKHHWNO
    20 TFQEAOXTLKBQELRPXKAALIIXOP
    21 UGRFBPYUMLCRFMSQYLBBMJJYPQ
    22 VHSGCQZVNMDSGNTRZMCCNKKZQR
    23 WITHDRAWONETHOUSANDDOLLARS
    24 XJUIESBXPOFUIPVTBOEEPMMBST
```

```
# Observe the answer is in the 23rd option
```

Transposition cipher

```
## Change the orderings. As opposed to using wraparound mod in abpve
2 cases
## Generally this permutation of the positions of the letters
```

```
# we encrypt the plain text pl defined earlier
p = 'ZAYBXCWDVEUFTGSHRIQJPKOLMN' # permutation text
perm = [ord(x)-64 for x in p]
```

```
perm
```

```
    [26,
     1,
     25,
      2,
      24,
      3,
      23,
      4,
      22,
      5,
      21,
      6,
      20,
      7,
      19,
      8,
      18,
      9,
      17,
      10,
      16,
      11,
      15,
      12,
      13,
      14]
```

```
# pl is the plain text. Convert from string to integers mod 26
pln = str2lst(pl)
```

```
pltr = [perm[pln[i]]-1 for i in range(len(pl))] # apply perm as the
permutation on the pln above
```

```
lst2str(pltr) # obtain the string output
```

```
    'OVJDBIZOSGXJDSPQZGBBSFFZIQ'
```

```
# Inverse permutation
```

```
len(perm)
```

```
    26
```

```
inv = Permutation(perm).inverse()
pls = [inv[pltr[i]]-1 for i in range(len(pl))]
lst2str(pls)
```

```
    'WITHDRAWONETHOUSANDDOLLARS'
```

MATRIX CIPHER

```
#let M be an n × n matrix whose determinant is relatively prime
#to 26
# Divide plain text into blocks of n symbols
# c = Mp (mod 26)  [where p = plain text and c = cipher text]
```

```
n=3 # matrix size
lp = str2lst(pl) # convert letters to numbers mod 26
```

```
ln = len(lp) # get lenth of lp
```

```
for i in range((-ln)%n):
    print(lp.append(23)) # ensure padding of the plain text  to be
multiple of 3
```

```
None
None
```

```
Mpl = transpose(matrix(len(lp)/n,n,lp)); Mpl # Matrix of plain text
```

```
[22  7  0 13  7 18  3 11 17 23]
[ 8  3 22  4 14  0  3 11 18 23]
[19 17 14 19 20 13 14  0 23 23]
```

```
M = matrix(Zmod(26),[[22,11,19],[15,20,24],[25,21,16]]) # n * n
matrix
```

Encryption

```
C = M*Mpl; C
```

```
[23 16 14 15 12 19  1 25 21  0]
[10  1 22  3  7 10 25 21 23  5]
[ 8 16 10 11  9  8 24 12  1 22]
```

```
# Convert to  cipher text string
lst2str(transpose(C).list())
```

```
'XKIQBQOWKPDLMHJTKIBZYZVMVXBAFW'
```