Block_Ciphers_treatment_1

ECB Mode Illustration

```
# Block ciphers operate on an n-bit block of plaintext (the length
of which is a function of the cipher)
M = matrix(Zmod(26),[[22,11,19],[15,20,24],[25,21,16]])
```

```
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]
```

```
M
```

```
[22 11 19]
[15 20 24]
[25 21 16]
```

```
pl = "WITHDRAWONEHUNDREDDOLLARSXX"
```

```
P = transpose(matrix(Zmod(26),9,3,str2lst(pl))).columns() # make
this into a matrix # transform integer value for plain text into
matrix P
```

```
P
```

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

```
nc = len(P) # length of p
nc
```

```
9
```

```
# The actual encryption will be done in a loop.
# The matrix product of M with the current column is the encryption.
# The result is appended to the current ciphertext matrix C as in
the ECB mode.

Cipher_text = []

for i in range(nc):
    Cipher_text += [M*P[i]]
```

```
Cipher_text
```

```
    [(23, 10, 8),
     (16, 1, 16),
     (14, 22, 10),
     (21, 1, 1),
     (16, 8, 15),
     (7, 17, 11),
     (13, 17, 25),
     (19, 1, 1),
     (20, 8, 1)]
```

```
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])
```

```
map(lst2str,Cipher_text)
```

```
    ['XKI', 'QBQ', 'OWK', 'VBB', 'QIP', 'HRL', 'NRZ', 'TBB', 'UIB']
```

Why ECB is not Recommended

```
# To see why this is not a good scheme, suppose a single value in
the plaintext is changed
pl2 = "WITHDREWONEHUNDREDDOLLARSXX" # now it is WITHDREW
```

```
P2 = transpose(matrix(Zmod(26),9,3,str2lst(pl2))).columns() #
Generate matrix for plain text
```

```
P2
```

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

```
# Encrypt the new plain text using ECB
cipher_text_faulty = []

for i in range(nc):
       cipher_text_faulty += [M*P2[i]]
```

```
# the only block in the ciphertext that changes
# is the single block where the plaintext change was made.
map(lst2str,cipher_text_faulty)
```

```
['XKI', 'QBQ', 'YEG', 'VBB', 'QIP', 'HRL', 'NRZ', 'TBB', 'UIB']
```

CBC MODE

```
IV = vector(Zmod(26),[7,13,19]) # declare initialisation vector
```

```
cipher_text_0 = [IV] # make initial cipher equals IV
```

```
# In order to keep the indices in the Sage loop the
# same as those in the definition, also add a block at the front P:
P = [vector([0,0,0])]+P
```

```
len(P)
```

```
10
```

```
# Practical example showing need for the vector notation
q= [(1,2,3),(4,5,6)]
vector([0,0,0])
q=[vector([0,0,0])] +q
q
```

```
[(0, 0, 0), (1, 2, 3), (4, 5, 6)]
```

```
# CBC encryption using loop
for i in range(1,nc+1):
        cipher_text_0 += [M*(P[i]+cipher_text_0[i-1]) ]
```

```
map(lst2str,cipher_text_0[1:nc+1])
```

```
['TRP', 'WYW', 'KIM', 'LBN', 'WLZ', 'VXW', 'CUH', 'ABL', 'GGQ']
```

```
# To check when a single text is changed ... this will affect the
whole cipher text in CBC. We will use P2 as the plain text as before
P2 = [vector([0,0,0])]+P2
cipher_text_1 = [IV]  # initialisation vector
for i in range(1,nc+1):
        cipher_text_1 += [M*(P2[i]+cipher_text_0[i-1]) ]
```

```
map(lst2str,cipher_text_1[1:nc+1]) # this generates a completely
different text
```

```
['WHH', 'DHO', 'MNS', 'OES', 'BEL', 'EOA', 'WUT', 'URJ', 'FZQ']
```

DECRYPTION

```
Decryption = []
MI = M.inverse()
for i in range(nc,0,-1):
    Decryption = [MI* cipher_text_0[i] - cipher_text_0[i-1]]
+Decryption
```

```
Decryption
```

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

```
map(lst2str,Decryption)
''.join(_) # join the output
```

```
'WITHDRAWONEHUNDREDDOLLARSXX'
```

Output Feedback Mode

```
IV = vector(Zmod(26),[7,13,19]) # Initialisation vector
K = [IV]
IV
```

```
(7, 13, 19)
```

```
# generate the keys to be used
for i in range(nc):
    K+= [M*K[i]]
```

```
# OFB mode. encrypt by saying
ofb_encrypt = [x+y for (x,y) in zip(P,K)]
```

```
ofb_encrypt
```

```
[(7, 13, 19),
 (4, 23, 17),
 (24, 11, 6),
 (19, 17, 15),
 (5, 5, 3),
 (13, 25, 20),
 (6, 1, 14),
 (15, 1, 5),
 (18, 10, 0),
 (11, 24, 6)]
```

```
# convert intergers to string and output
map(lst2str,ofb_encrypt[1:nc+1])
```

```
['EXR', 'YLG', 'TRP', 'FFD', 'NZU', 'GBO', 'PBF', 'SKA', 'LYG']
```

```
# To decrypt
ofb_decrypt = [x-y for (x,y) in zip(ofb_encrypt,K)]
map(lst2str,ofb_decrypt[1:nc+1])
```

```
['WIT', 'HDR', 'AWO', 'NEH', 'UND', 'RED', 'DOL', 'LAR', 'SXX']
```

```
''.join(_) # join the output
```

```
'WITHDRAWONEHUNDREDDOLLARSXX'
```

CTR mode

```
# Choose a vector
nonce= [11,21]
```

```
# Generate a counter stream using the nonce
CTR = [vector(v+[i+1]) for i in range(nc)]
```

```
CTR
```

```
[(11, 21, 1),
 (11, 21, 2),
 (11, 21, 3),
 (11, 21, 4),
 (11, 21, 5),
 (11, 21, 6),
 (11, 21, 7),
 (11, 21, 8),
 (11, 21, 9)]
```

```
# For encryption
crypt_ctr = []
```

```
for i in range(nc):
    crypt_ctr += [M*CTR[i]+P[i]]
```

```
crypt_ctr
```

```
[(24, 11, 4),
 (13, 17, 13),
 (17, 10, 1),
 (3, 1, 14),
 (9, 7, 23),
 (9, 14, 9),
 (25, 3, 25),
 (4, 11, 23),
 (5, 21, 19)]
```

```
map(lst2str,crypt_ctr)
```

```
['YLE', 'NRN', 'RKB', 'DBO', 'JHX', 'JOJ', 'ZDZ', 'ELX', 'FVT']
```

```
```