In [1]:
```
import numpy
import random
from timeit import default_timer as timer
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [2]:
```
######## Showing processing procedures
PRINT_FLAG = False
```

In [3]:
```
########XOR operation#####################################################################################
"""
    function:do XOR operation on bits string s1, s2
    condition: len (s1) == len(s2)
    return: xorResult -- the xor result and itstype is list
"""
def XOROperation(s1,s2):
    XOR_Operation_start = timer()

    length = len(s1)
    xorResult = []
    for i in range(0, length):
        # Convert int type to binary bits, then convert it to string type after the XOR operation
        xorResult.extend(str(int(s1[i]) ^ int(s2[i])))

    XOR_Operation_end = timer()

    XOR_Operation_.append(XOR_Operation_end - XOR_Operation_start)
    return xorResult
##########################################################################################################
```

In [4]:
```
########## int to Binary with fixed bits representation#################################################
def int2bin(n, count=24):
    Int_2_bin_start = timer()

    """returns the binary of integer n, using count number of digits"""
    result = "".join([str((n >> y) & 1) for y in range(count-1, -1, -1)])

    Int_2_bin_end = timer()
    Int_2_bin_.append(Int_2_bin_end-Int_2_bin_start)
```

```
    return result
##########################################################################################################
```

In [5]:

```
######binaryStr by given permutation table#########################################################
"""
    function: transfrom the binaryStr with the given permutation table
    condition: len(binaryStr) == len(PermutationTable)
    return: the permutated binary List.
"""

def Permutation(binaryStr, PermutationTable):
    Permutation_table_start = timer()

    length = len(PermutationTable)
    PermutatedList = []
    for i in range(0, length):
        PermutatedList.extend(binaryStr[PermutationTable[i] - 1])

    Permutation_table_end = timer()
    Permutation_table_.append(Permutation_table_end-Permutation_table_start)

    return PermutatedList
##########################################################################################################
```

In [6]:

```
##cycle left shift n bits#########################################################################
"""
    function: to achieve cycle shift n bits.
    return: the shifted result.
"""
def shiftLeft(binaryStr, nBits):
    Cycle_shift_left_start = timer()

    length = len(binaryStr)
    nBits = nBits % nBits
    shiftedList = list(binaryStr)
    for i in range(0, length):
        if i < nBits:
            shiftedList.extend(shiftedList[0])
            del shiftedList[0]
        else:
            break
```

```
    Cycle_shift_left_end = timer()

    Cycle_shift_left_.append(Cycle_shift_left_end - Cycle_shift_left_start)

    return shiftedList
##############################################################################################
```

In [7]:

```
##Byte 2 bit ##################################################################################
def ByteToBit(ByteString):
    Byte_2_Bit_start = timer()

    bitList = []
    for i in range(0,4):
        bitList.insert(0, str(ByteString%2))
        ByteString = int(ByteString / 2)
    bitResult = "".join(bitList)

    Byte_2_Bit_end = timer()

    Byte_2_Bit_.append(Byte_2_Bit_end - Byte_2_Bit_start)

    return bitResult
##############################################################################################
```

In [8]:

```
#########Initial P Permutation#################################################################################
InitialPermutationTable=[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]

"""
    function: Initial permutation function
    input: M_0--64bit plain text block
    return: L_0--the front 32 bits of M_0 , R0--the back 32 bits of M_0
"""
def InitialPermutation(M_0):
    Initial_Permutation_start = timer()

    if PRINT_FLAG == True:
```

```
        print("> processing initial IP permutation")

    InitialPermutationResult = Permutation(M_0, InitialPermutationTable)
    L_0 = InitialPermutationResult[0:int((len(InitialPermutationResult)/2))]
    R_0 = InitialPermutationResult[int((len(InitialPermutationResult)/2)):int(len(InitialPermutationResult))]

    Initial_Permutation_end = timer()
    Initial_Permutation_.append(Initial_Permutation_end - Initial_Permutation_start)
    return L_0, R_0 # List type
#############################################################################################
```

In [9]:

```
#####PC-1 permutation#########################################################################
PC_1Table = [57,49,41,33,25,17,9,
             1,58,50,42,34,26,18,
             10,2,59,51,43,35,27,
             19,11,3,60,52,44,36,
             63,55,47,39,31,23,15,
             7,62,54,46,38,30,22,
             14,6,61,53,45,37,29,
             21,13,5,28,20,12,4]

"""
    function: PC-1 permutation
    input: 56 not checked bits of secret ley
    return: C_0, D_0
"""
def PC_1_Permutation(SecretKey):
    PC_1_Permutation_start = timer()

    if PRINT_FLAG == True:
        print("> processing PC-1 permutation")
    PC_1_PermutationResult = Permutation(SecretKey, PC_1Table)
    C_0 = PC_1_PermutationResult[0: int(len(PC_1_PermutationResult)/2)]
    D_0 = PC_1_PermutationResult[int(len(PC_1_PermutationResult)/2): int(len(PC_1_PermutationResult))]

    PC_1_Permutation_end = timer()
    PC_1_Permutation_.append(PC_1_Permutation_end - PC_1_Permutation_start)
    return C_0, D_0
#############################################################################################
```

In [10]:

```
########cycle left shift n bits#################################################################################################
"""
    function: do ring shift left on a str_28_bits
```

```
    input: str_28_bits -- a 28 bits string; ShiftFlag -- when it is 1,2,9,16, shift 2 bits
    return: shift_result
"""
def RingShiftLeft(str_28_bits, ShiftFlag):
    Ring_Shift_Left_start = timer()

    shiftResult = ""
    if ShiftFlag == 1 or ShiftFlag == 2 or ShiftFlag == 9 or ShiftFlag == 16:
        shiftResult = shiftLeft(str_28_bits, 2)
    else:
        shiftResult = shiftLeft(str_28_bits, 1)

    Ring_Shift_Left_end = timer()
    Ring_Shift_Left_.append(Ring_Shift_Left_end - Ring_Shift_Left_start)

    return shiftResult
##############################################################################################
```

In [11]:

```
##########PC-2 permutation#############################################################################################
PC_2Table = [14,17,11,24,1,5,
             3,28,15,6,21,10,
             23,19,12,4,26,8,
             16,7,27,20,13,2,
             41,52,31,37,47,55,
             30,40,51,45,33,48,
             44,49,39,56,34,53,
             46,42,50,36,29,32]
"""
    function: PC-2 compressed permutation
    input:  str_56_bits
    return: str_48_bits
"""
def PC_2_Permutation(str_56_bits):
    PC_2_Permutation_start = timer()

    if PRINT_FLAG == True:
        print("> processing PC-2 permutation")
    #  get rid off 9,  18,  22,  25,  35,  38, 43,  54 th digit
    str_48_bits = Permutation(str_56_bits, PC_2Table)

    PC_2_Permutation_end = timer()

    PC_2_Permutation_.append(PC_2_Permutation_end-PC_2_Permutation_start)
```

```
    return str_48_bits
##############################################################################################
```

In [12]:

```
####Create sub-key#############################################################################
"""
    function: create the 16 son keys with the given key
    return: sonKeysList: 16 son keys list
"""
def createSonKey(SecretKey):
    # extract the non-validation bits in secret key
    Sub_key_creation_start = timer()

    if PRINT_FLAG == True:
        print("> Createing 16 bits sub-key")
    str_56_bits_List = list(SecretKey)
    sonKeyList = []
    # creating sub-key
    Temp_PC_1_PermutationResult_C_i_1, Temp_PC_1_PermutationResult_D_i_1 = PC_1_Permutation(str_56_bits_List)
    C_i = []
    D_i = []
    for i in range(1, 17):
        # C_i-1 D_i-1
        # Calculate C_i D_i
        if i == 1 or i == 2 or i == 9 or i == 16:
            C_i = shiftLeft(Temp_PC_1_PermutationResult_C_i_1, 1)
            D_i = shiftLeft(Temp_PC_1_PermutationResult_D_i_1, 1)
        else:
            C_i = shiftLeft(Temp_PC_1_PermutationResult_C_i_1, 2)
            D_i = shiftLeft(Temp_PC_1_PermutationResult_D_i_1, 2)
        CD = C_i + D_i
        sonKey_i = PC_2_Permutation(CD)
        sonKeyList.append(sonKey_i)
        Temp_PC_1_PermutationResult_C_i_1 = C_i
        Temp_PC_1_PermutationResult_D_i_1 = D_i
        if i == 16:
            break

    Sub_key_creation_end = timer()
    Sub_key_creation_.append(Sub_key_creation_end - Sub_key_creation_start)

    return sonKeyList
##############################################################################################
```

In [13]:
```
######E Expansion permutation#######################################################################################
E_ExpandTable = [32,1,2,3,4,5,
                 4,5,6,7,8,9,
                 8,9,10,11,12,13,
                 12,13,14,15,16,17,
                 16,17,18,19,20,21,
                 20,21,22,23,24,25,
                 24,25,26,27,28,29,
                 28,29,30,31,32,1]
"""
    function: E_Expand on the 32 bits R(i-1) string
    input: R_i_1 -- the (i-1)th back 32 bits string
    return: E_R_i_1 -- the 48 bits expanded string
"""
def E_Expand(R_i_1):
    E_Expansion_start = timer()

    if PRINT_FLAG == True:
        print("> processing E Expansion permutation")
    E_R_i_1 = Permutation(R_i_1, E_ExpandTable)

    E_Expansion_end = timer()
    E_Expansion_.append(E_Expansion_end - E_Expansion_start)

    return E_R_i_1
#####################################################################################################
```

In [14]:
```
#######S Box permutation#############################################################################################
eight_S_Boxes=[[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
                0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
                4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
                15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,],
                [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
                3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
                0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
                13,8,10,1,3,15,4,2,11,6,7,12,10,5,14,9,],
                [10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
                13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
                13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
                1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12],
                [7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
                13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
                10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
                3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,],
```

```
                [2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
                14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
                4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
                11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3],
                [12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
                10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
                9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
                4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,],
                [4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
                13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
                1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
                6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12],
                [13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
                1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
                7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
                2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]]
"""
    function: to transfrom a 6-bits string to a 4-bits string with 8 S-Boxes
    input: six_bits_str -- 6-bits string; S_Box_Num -- indicate the number of the S-Box [1, 8]
    return: four_bits_str -- 4 bits string group
"""
def S_Box_Transformation(six_bits_str, S_Box_Num):
    S_Box_permutation_start = timer()

    if PRINT_FLAG == True:
        print("> processing S Box permutation with 6-4 transform")
    row = int(six_bits_str[0]) * 2 + int(six_bits_str[5])
    col = int(six_bits_str[1]) * 8 + int(six_bits_str[2]) * 4 + int(six_bits_str[3]) * 2 + int(six_bits_str[4])
    value = eight_S_Boxes[int(S_Box_Num - 1)][int(row * 15 + col)]
    four_bits_str = list(int2bin(value,4))

    S_Box_permutation_end = timer()
    S_Box_permutation_.append(S_Box_permutation_end - S_Box_permutation_start)
    return four_bits_str
#################################################################################################
```

In [15]:

```
########P Expansion permutation#################################################################################
P_Table=[16,7,20,21,
   29,12,28,17,
   1,15,23,26,
   5,18,31,10,
   2,8,24,14,
   32,27,3,9,
   19,13,30,6,
```

```
   22,11,4,25]
"""
    function: P_Permutation on the 32 bits string
    input: str_32bits -- the 32 bits string List
    return: FeistelResult -- the output of the feistel function
"""
def P_Permutation(str_32bits):
    P_Expansion_start = timer()

    if PRINT_FLAG == True:
        print("> processing P Expansion permutation")
    FeistelResult = Permutation(str_32bits, P_Table)

    P_Expansion_end = timer()
    P_Expansion_.append(P_Expansion_end - P_Expansion_start)

    return FeistelResult
#############################################################################################
```

In [16]:

```
#####Feistel function#########################################################################
"""
    function: Feistel function to create bit-stR_ing to permute with R_i -- a 32-bit stR_ing
    input: R_i_1--the (i-1)th back 32 bits string, K_i--the son secret key
    return: Feistel result (string type)
"""
def Feistel(R_i_1, K_i):
    Feistel_network_start = timer()

    if PRINT_FLAG == True:
        print("> processing Feistel function")
    E_ExpandResult = E_Expand(R_i_1)
    xorResult = XOROperation(E_ExpandResult, K_i)
    str_32_bits = []
    for i in range(8):
        str_6_bits = xorResult[i * 6: i * 6 + 6]
        str_32_bits += S_Box_Transformation(str_6_bits, i + 1)

    result = "".join(P_Permutation(str_32_bits))

    Feistel_network_end = timer()
    Feistel_network_.append(Feistel_network_end - Feistel_network_start)
    return result
#############################################################################################
```

In [17]:
```
#########cross iteration in cryption#############################################################################
"""
    function: make cross iteration on L0, R0 for 16 times
    input: L0--the front 32 bits of 64-bits plain text , R0--the back 32 bits of plain text
    return: R16--the back iterated 32-bits result, L16--the front iterated 32-bits result
"""
def CrossIterationInEncryption(L_0, R_0, SecretKey):
    Cross_Iteration_Encryption_start = timer()

    if PRINT_FLAG == True:
        print("> processing cross iteration in cryption")
    R = ""
    L = ""
    tmp_R = R_0
    tmp_L = L_0
    sonKeyList = createSonKey(SecretKey)
    for i in range(1,17):
        L = tmp_R
        R = XOROperation(tmp_L,Feistel(tmp_R,sonKeyList[i - 1]))
        tmp_R = R
        tmp_L = L
    RL = R + L

    Cross_Iteration_Encryption_end = timer()
    Cross_Iteration_Encryption_.append(Cross_Iteration_Encryption_end - Cross_Iteration_Encryption_start)
    return RL
#################################################################################################################
```

In [18]:
```
#########cross iteration in decryption#############################################################################################
"""
    function: make cross iteration on L0, R0 for 16 times
    input: L0--the front 32 bits of 64-bits cipher text , R0--the back 32 bits of cipher text
    return: R16--the back iterated 32-bits result, L16--the front iterated 32-bits result
"""
def CrossIterationInDecryption(L_0, R_0, SecretKey):
    Cross_Iteration_Decryption_start = timer()

    if PRINT_FLAG == True:
        print("> processing the cross iteration in decryption")
    R = []
    L = []
    tmp_R = R_0
    tmp_L = L_0
    sonKeyList = createSonKey(SecretKey)
```

```
    for i in range(1,17):
        L = tmp_R
        R = XOROperation(tmp_L,Feistel(tmp_R,sonKeyList[16 - i]))
        tmp_R = R
        tmp_L = L
    RL = R + L

    Cross_Iteration_Decryption_end = timer()
    Cross_Iteration_Decryption_.append(Cross_Iteration_Decryption_end - Cross_Iteration_Decryption_start)

    return RL
#############################################################################################
```

In [19]:

```
######P inverse Permutation#############################################################################################
InversePermutationTable=[40,8,48,16,56,24,64,32,
                         39,7,47,15,55,23,63,31,
                         38,6,46,14,54,22,62,30,
                         37,5,45,13,53,21,61,29,
                         36,4,44,12,52,20,60,28,
                         35,3,43,11,51,19,59,27,
                         34,2,42,10,50,18,58,26,
                         33,1,41,9,49,17,57,25]

"""
    function: inverse permutation on the R16L16 bit-stR_ing
    input: R16--the back iterated 32-bits result, L16--the front iterated 32-bits result
    return: ciphterText--64bits
"""
def InversePermutation(R_16_L_16):
    P_inverse_Permutation_start = timer()

    if PRINT_FLAG == True:
        print("> processing P inverse Permutation")
    cipherText = ""
    cipherText = Permutation(R_16_L_16, InversePermutationTable)

    P_inverse_Permutation_end = timer()
    P_inverse_Permutation_.append(P_inverse_Permutation_end - P_inverse_Permutation_start)

    return cipherText
#############################################################################################
```

In [20]:

```
#####Encryption function#################################################################################
```

```
def Encryption(plainText, secretKey):
    DES_Encryption_start = timer()

    if PRINT_FLAG == True:
        print("> start Encrypt 64 bits plain text")
    M = list(plainText)
    L0, R0 = InitialPermutation(M)
    RL = CrossIterationInEncryption(L0, R0, secretKey)
    cipherText = "".join(InversePermutation(RL))

    DES_Encryption_end = timer()
    DES_Encryption_.append(DES_Encryption_end - DES_Encryption_start)
    return cipherText
#############################################################################################
```

In [21]:

```
######decryption function#####################################################################################
def Decryption(cipherText, secretKey):
    DES_Decryption_start = timer()

    if PRINT_FLAG == True:
        print("> start Decrypt 64 bits cipher text")
    M = list(cipherText)
    L0, R0 = InitialPermutation(M)
    RL = CrossIterationInDecryption(L0, R0, secretKey)
    decryptedText = "".join(InversePermutation(RL))

    DES_Decryption_end = timer()
    DES_Decryption_.append(DES_Decryption_end - DES_Decryption_start)
    return decryptedText
#############################################################################################
```

In [22]:

```
####randomly generate 64 bit secret key AKA 8 bytes#########################################################################
"""
    return: a 64-bits (8 bytes) string as a secret key
"""
def createSecrteKey():
    Create_Secret_Key_start = timer()

    seed = "1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()_+=-"
    key = []
    for i in range(8):
        key.append(random.choice(seed))
    randomSecretKey = ''.join(key)
```

```
    Create_Secret_Key_end = timer()
    Create_Secret_Key_.append(Create_Secret_Key_end - Create_Secret_Key_start)

    return randomSecretKey
#############################################################################################
```

In [23]:

```
########## transform 8 bytes string variable to ascii, then covert from decimal to binary string #################
def ToBitString(string_8_char):
    To_Bit_String_start = timer()

    strList = []
    for i in range(8):
        strList.append(str(int2bin(ord(string_8_char[i]), 8)))

    result = "".join(strList)

    To_Bit_String_end = timer()
    To_Bit_String_.append(To_Bit_String_end - To_Bit_String_start)

    return result
#############################################################################################
```

In [33]:

```
########convert 64 bits to 8 ascci characters#################################################
def ToAsciiChar(string_64_bits):
    To_Ascii_Char_start = timer()

    strList = []
    bitList = list(string_64_bits)
    for i in range(8):
        if int("".join(bitList[i * 8: i * 8 + 8]), 2) < 8:
            continue
        # we take 8 bits as a processing unit, we turn it to decimal first then covert into ascii charater form. fianlly
        # we put those charaters together and save it to a list.
        strList.append(chr(int("".join(bitList[i * 8: i * 8 + 8]), 2)))

    result = "".join(strList)

    To_Ascii_Char_end = timer()
    To_Ascii_Char_.append(To_Ascii_Char_end - To_Ascii_Char_start)

    return result
#############################################################################################
```