DES Algorithm Implementation in Python

Erikas Eigėlis

Vinius University Kaunas Faculty

Information Systems and Cyber Security (ISKS6)

Hardcoded lists

```
# Initial permutation matrix for the data blocks
PI = [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]

# Initial permutation made on the key
KP_1 = [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]

# Permutation applied on shifted key to get Ki+1
KP_2 = [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]
```

```
# Expand 32bit matrix to get a 48bits in order to apply the xor with Ki
EXPANSION_DBOX = [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]

# Permutation made after each S-Box substitution for each round
STRAIGHT_DBOX = [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]

# Final permutation for block after 16 rounds
FINAL_PERMUTATION = [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]

# Array to determine the shift for each round of keys
SHIFT = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
```

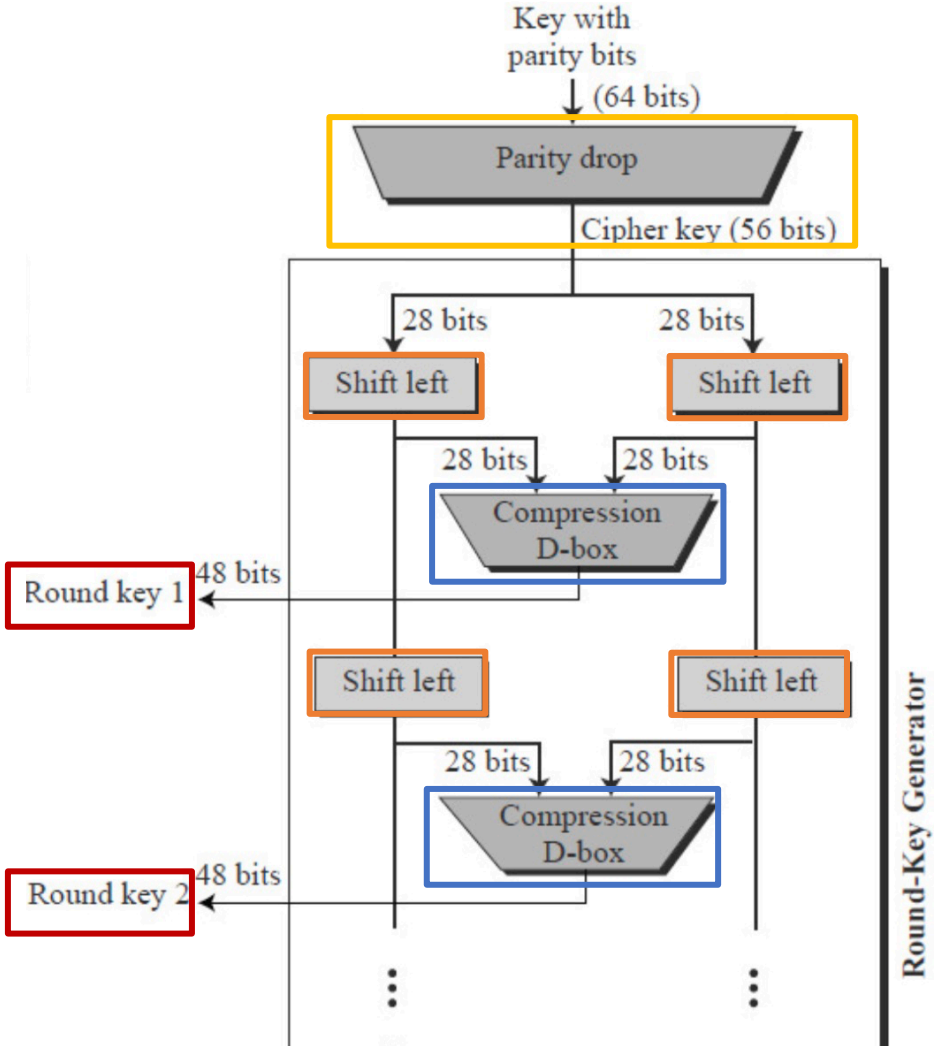
S-Boxes

- 8 S-Boxes are added into array *S_BOX*
- Each S-Box is two dimensional array so that row and column indexes could be accessed

```
# S-Boxes
S_BOX = [
    # 1:
    [[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],
     ],
    # 2:
    [[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, 0, 5, 14, 9],
     ],
    # 3:
    [[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],
     ],
    # 4:
    [[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],
     ],
    # 5:
    [[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],
     ],
    # 6:
    [[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],
     ],
    # 7:
    [[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],
     ],
    # 8:
    [[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],
     ]
]
```

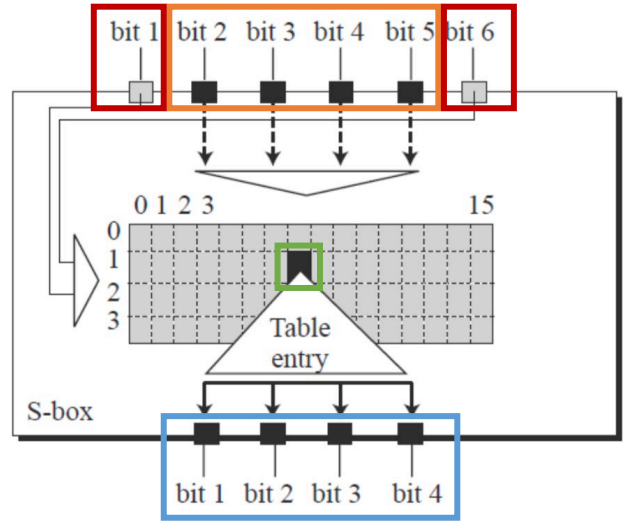
Key Generation

```
def generate_keys(self):
    self.keys = []
    # Convert hex password to binary:
    key = self.create_bit_array(self.password)
    # Perform initial permutation to a key (64bits -> 56 bits)
    key = self.permutation(key, KP_1)
    # Split key in half:
    left = key[:28]
    right = key[28:]
    # Generate 16 keys
    for i in range(16):
        # Shift left and right sides by set steps in shift list
        left, right = self.shift(left, right, SHIFT[i])
        temp = left + right  # merge two sides of a key
        # Perform final key permutation
        a = self.permutation(temp, KP_2)
        self.keys.append(a)  # add to key list
```



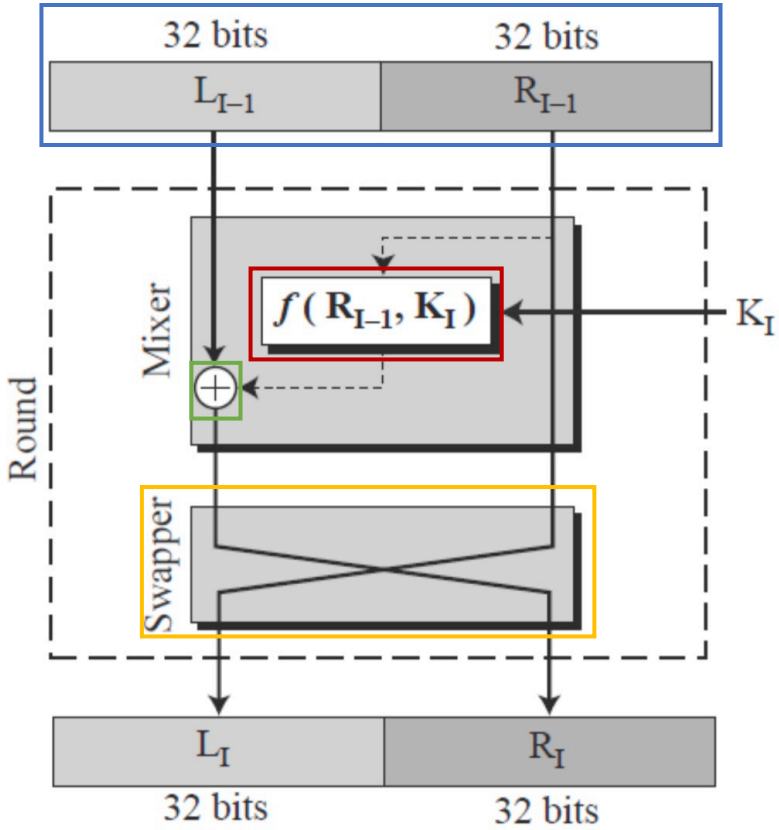
S-Box substitution

```
def substitute(self, right_expanded):  # Substitute bytes using S-Box
    subblocks = self.split_array(right_expanded, 6)  # Split bit array into sublist of 6 bits
    result = list()
    # For all the sub blocks:
    for i in range(len(subblocks)):
        block = subblocks[i]
        # Get decimal int values for S-Box row and column by converting numbers from binary:
        row = int(str(block[0]) + str(block[5]), 2)  # Row is the first and last bits
        column = int(''.join([str(x) for x in block[1:][:-1]]), 2)  # Column consists of middle 4 bits (2nd to 5th)
        val = S_BOX[i][row][column]  # Take the value in the S-Box appropriated for the round (i)

        bin_num = format(val, 'b').zfill(4)  # Convert the value to binary
        result += [int(x) for x in str(bin_num)]  # And append it to the result list
    return result
```



DES Round

```
# Splitting message into blocks of 64 bits and performing DES operations on them:

for text_bits in self.split_array(text_bit_array, 64):
    # Performing initial permutation on a block:
    text_bits = self.permutation(text_bits, PI)
    # Initial split to left and right:
    left = text_bits[:32]
    right = text_bits[32:]

    for i in range(16):
        right_expanded = self.permutation(right, EXPANSION_DBOX)  # Expand right to match Ki size (48bits)
        if action == 'encrypt':
            temp = self.xor(self.keys[i], right_expanded)  # If encrypt use Ki from first key...
        else:
            temp = self.xor(self.keys[15 - i], right_expanded)  # ... if decrypt use Ki from last key

        temp = self.substitute(temp)  # Method that will apply the S-Boxes
        temp = self.permutation(temp, STRAIGHT_DBOX)  # Perform permutation after S-Box
        temp = self.xor(left, temp)  # Perform XOR with left
        # Switch left and right sides:
        left = right
        right = temp

    # Performing final permutation for a block:
    # The last DES round does not switch left with right, but since it's done in all previous steps, after all rounds right is passed first and left
is passed second:
    final_permutation = self.permutation(right + left, FINAL_PERMUTATION)

    # result gets value of final permutaion if it's empty (the first block is used),
    # if it's another block, it's final permutation is added to a result:
    result = final_permutation if result is None else result + final_permutation

return result
```



Live showcase:
https://isks6.pythonanywhere.com/des/

View source code:
https://github.com/EErikas/PythonCryptoExamples