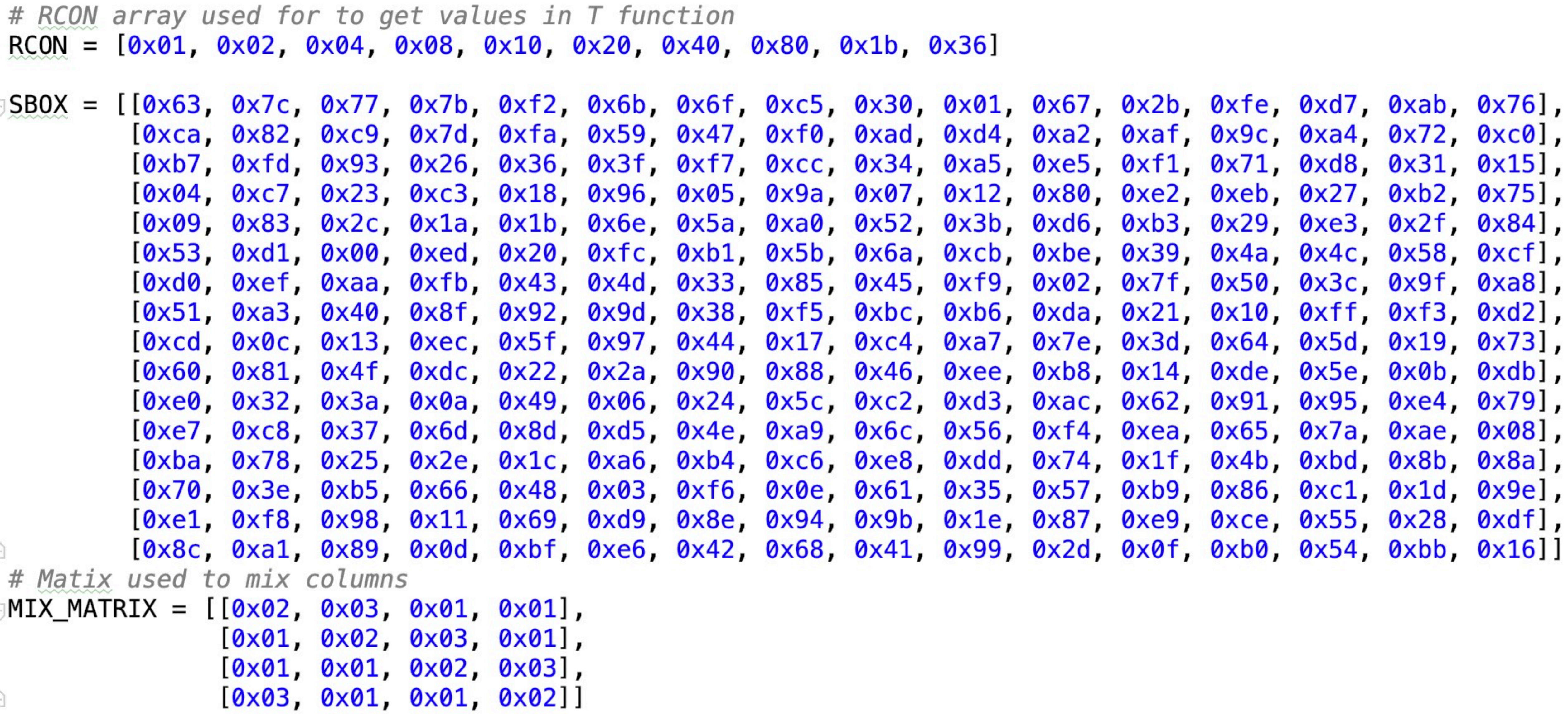
AES Algorithm Implementation in Python

Erikas Eigėlis

Vinius University Kaunas Faculty

Information Systems and Cyber Security (ISKS6)

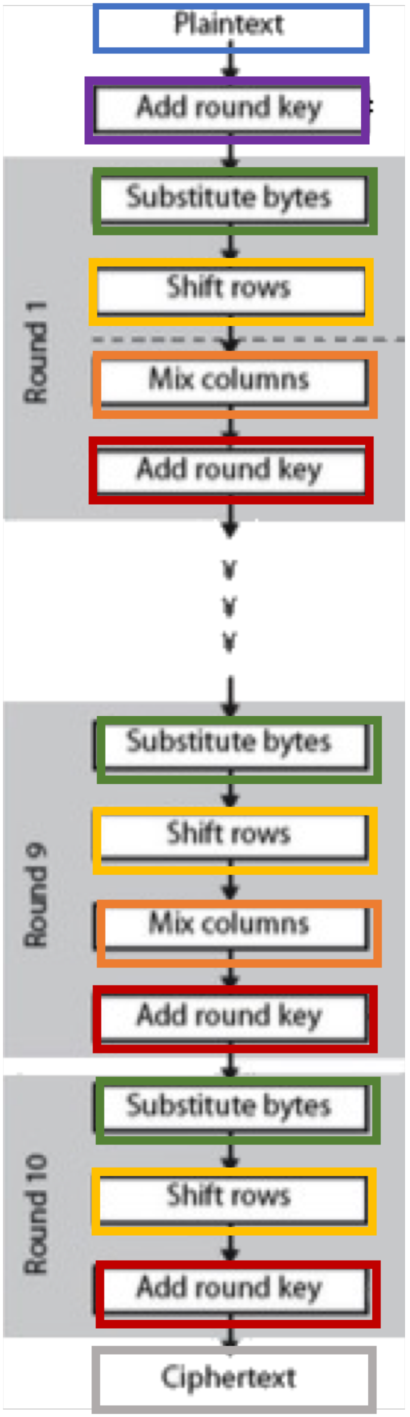
Hardcoded lists



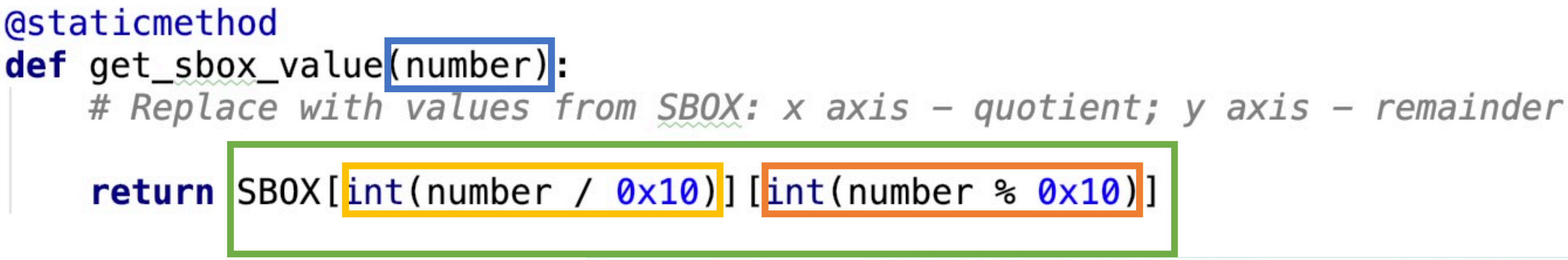
```
# RCON array used for to get values in T function
RCON = [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36]

SBOX = [[0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],
         [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],
         [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],
         [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],
         [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],
         [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],
         [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],
         [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],
         [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],
         [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],
         [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],
         [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],
         [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],
         [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],
         [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],
         [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]]
# Matix used to mix columns
MIX_MATRIX = [[0x02, 0x03, 0x01, 0x01],
              [0x01, 0x02, 0x03, 0x01],
              [0x01, 0x01, 0x02, 0x03],
              [0x03, 0x01, 0x01, 0x02]]
```

AES Round

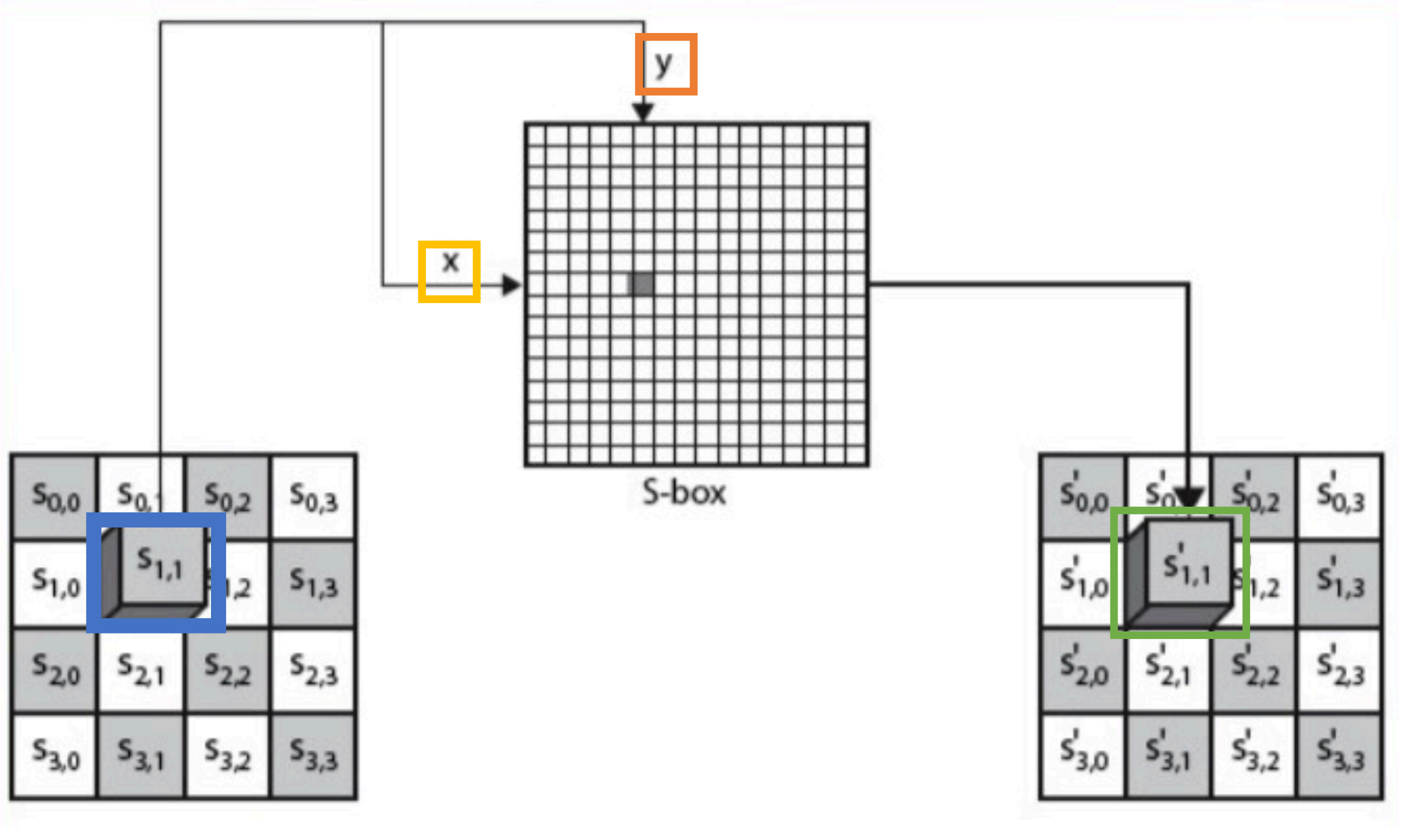
```
def do_rounds(self, message, key):
    # generate round keys:
    keys = self.generate_round_keys(key)
    # ...

    message_blocks = self.flip_matrix(self.split_array(message, 4))

    block = self.apply_round_key(message_blocks, keys[0])


    for round_no in range(1, len(keys)):


        # Substitutes values with corresponding ones from SBOX:
        block = [[self.get_sbox_value(foo) for foo in bar] for bar in block]


        # Shifts rows cyclically to the left by offsets of 0, 1, 2 and 3:
        block = [[int(foo) for foo in numpy.roll(block[i], -1 * i)] for i in range(len(block))]

        # Perform mix columns operation for rounds 1 - 9:
        if round_no < 10:
            block = self.mix_columns(block)

        # Apply round key:
        block = self.apply_round_key(block, keys[round_no])
output['cipher'] = self.format_output(self.flip_matrix(block), output_type='line')
```



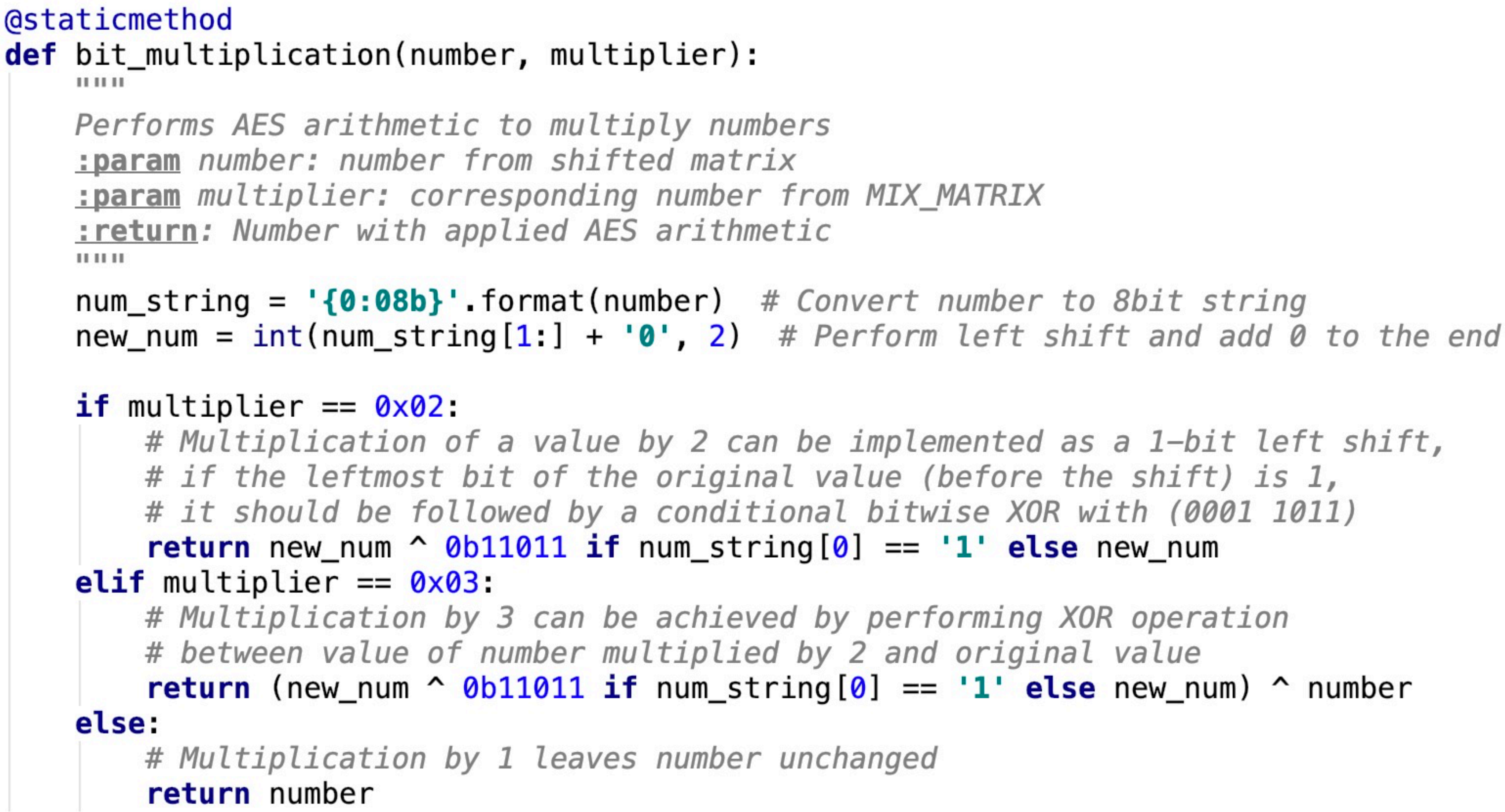
Substitute bytes



```
@staticmethod
def get_sbox_value(number):
    # Replace with values from SBOX: x axis - quotient; y axis - remainder

    return SBOX[int(number / 0x10)][int(number % 0x10)]
```

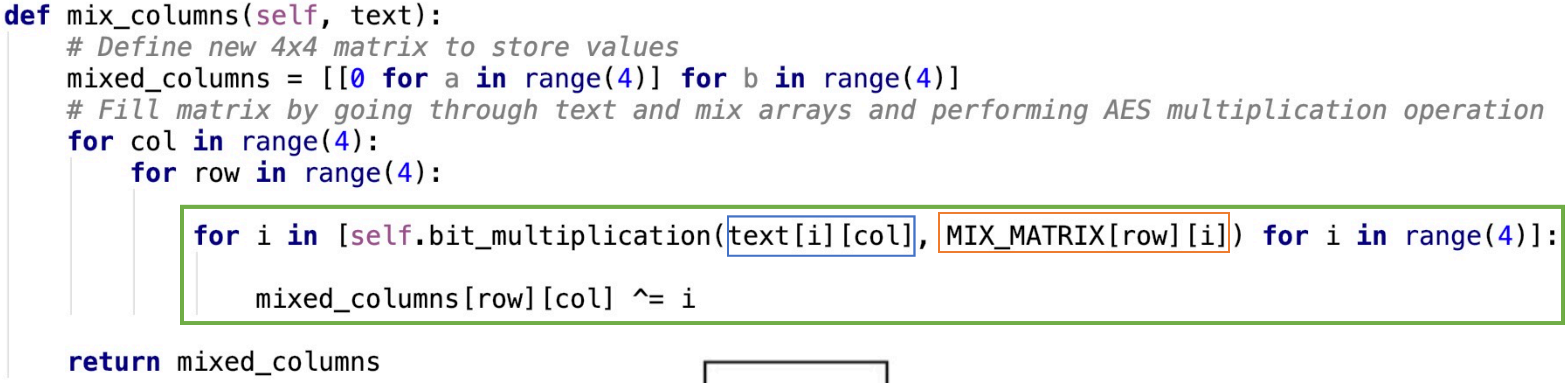


AES Arithmetic

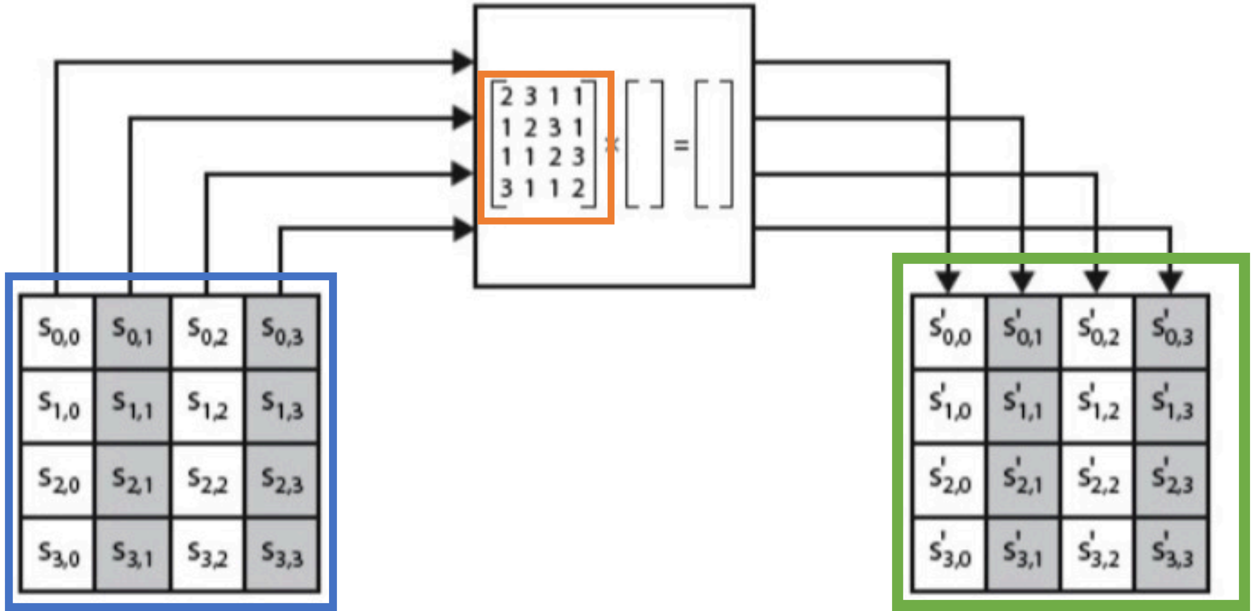


```python
@staticmethod
def bit_multiplication(number, multiplier):
    """
    Performs AES arithmetic to multiply numbers
    :param number: number from shifted matrix
    :param multiplier: corresponding number from MIX_MATRIX
    :return: Number with applied AES arithmetic
    """
    num_string = '{0:08b}'.format(number)  # Convert number to 8bit string
    new_num = int(num_string[1:] + '0', 2)  # Perform left shift and add 0 to the end

    if multiplier == 0x02:
        # Multiplication of a value by 2 can be implemented as a 1-bit left shift,
        # if the leftmost bit of the original value (before the shift) is 1,
        # it should be followed by a conditional bitwise XOR with (0001 1011)
        return new_num ^ 0b11011 if num_string[0] == '1' else new_num
    elif multiplier == 0x03:
        # Multiplication by 3 can be achieved by performing XOR operation
        # between value of number multiplied by 2 and original value
        return (new_num ^ 0b11011 if num_string[0] == '1' else new_num) ^ number
    else:
        # Multiplication by 1 leaves number unchanged
        return number
```

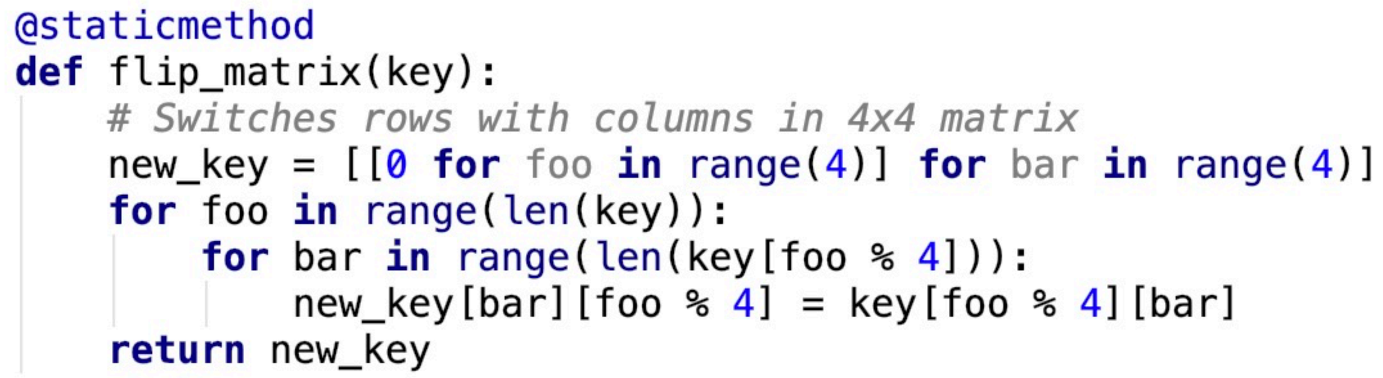
Mix Columns



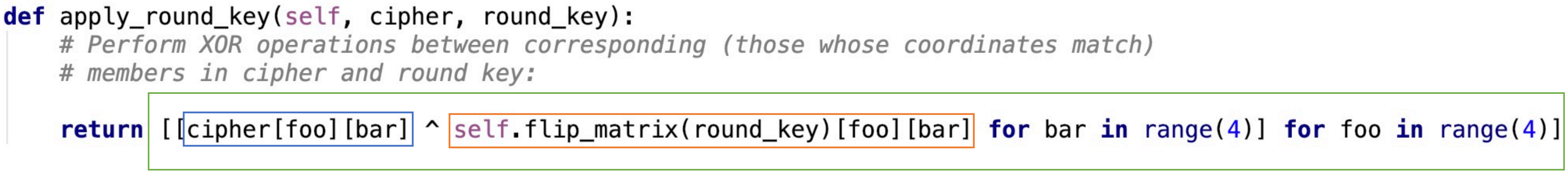
```python
def mix_columns(self, text):
    # Define new 4x4 matrix to store values
    mixed_columns = [[0 for a in range(4)] for b in range(4)]
    # Fill matrix by going through text and mix arrays and performing AES multiplication operation
    for col in range(4):
        for row in range(4):

            for i in [self.bit_multiplication(text[i][col], MIX_MATRIX[row][i]) for i in range(4)]:

                mixed_columns[row][col] ^= i

    return mixed_columns
```


2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

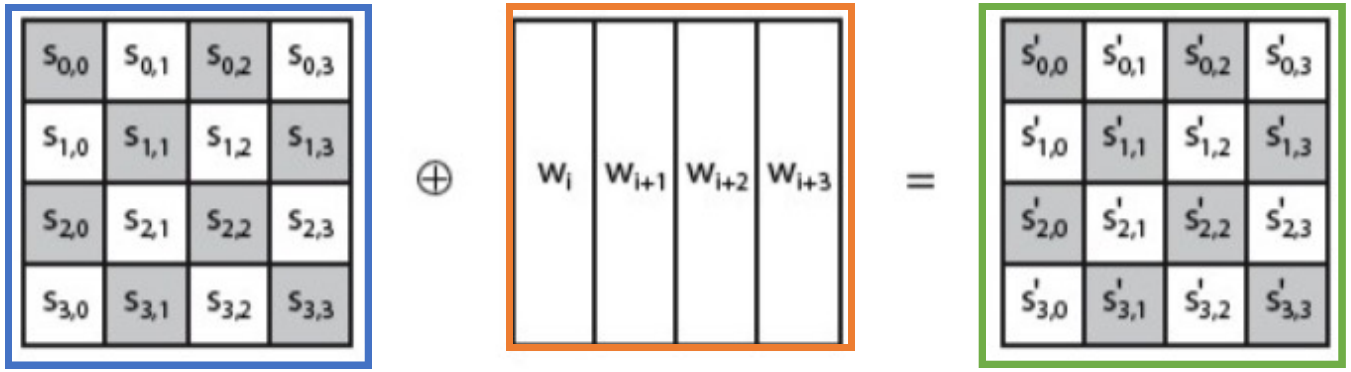
Add Round Key



```python
@staticmethod
def flip_matrix(key):
    # Switches rows with columns in 4x4 matrix
    new_key = [[0 for foo in range(4)] for bar in range(4)]
    for foo in range(len(key)):
        for bar in range(len(key[foo % 4])):
            new_key[bar][foo % 4] = key[foo % 4][bar]
    return new_key
```

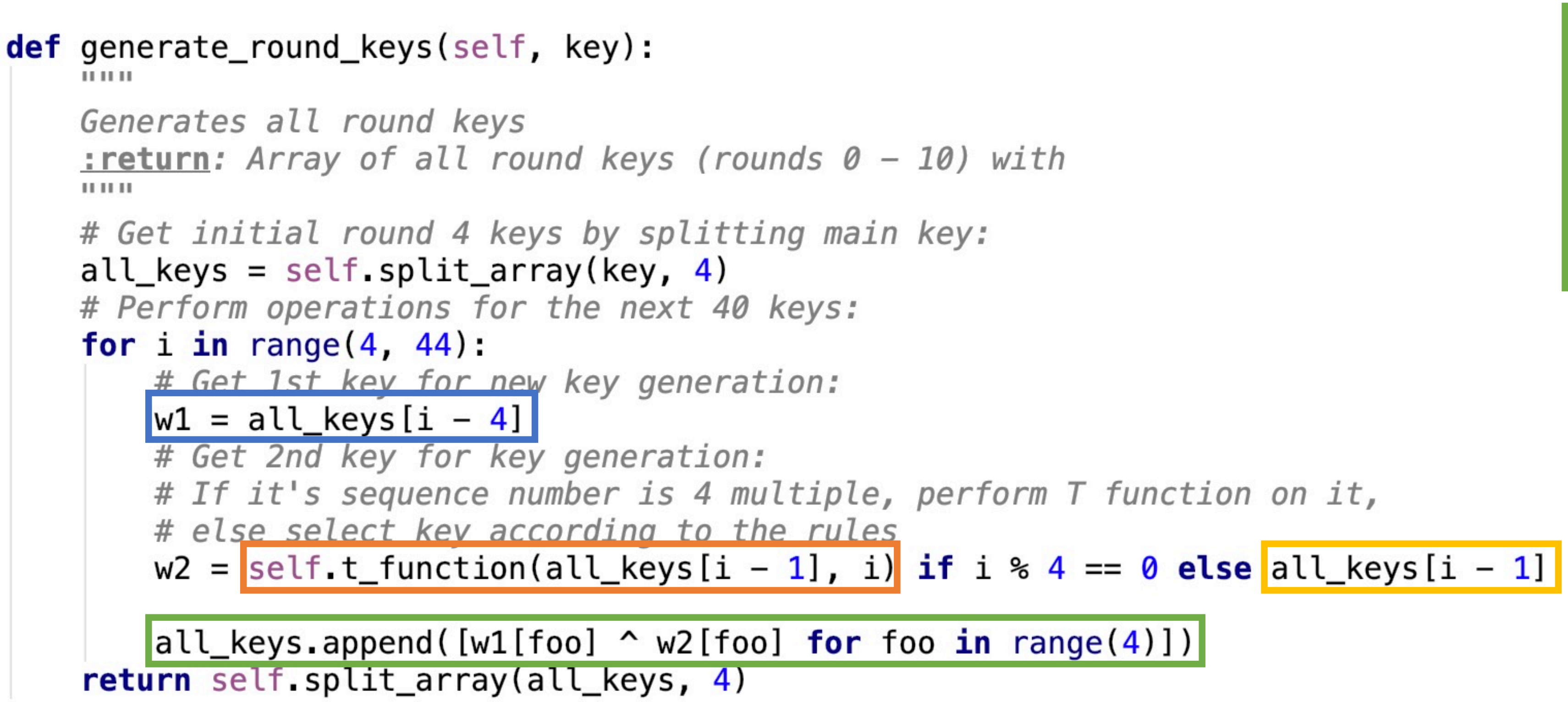


```python
def apply_round_key(self, cipher, round_key):
    # Perform XOR operations between corresponding (those whose coordinates match)
    # members in cipher and round key:

    return [[cipher[foo][bar] ^ self.flip_matrix(round_key)[foo][bar] for bar in range(4)] for foo in range(4)]
```



$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Key Generation



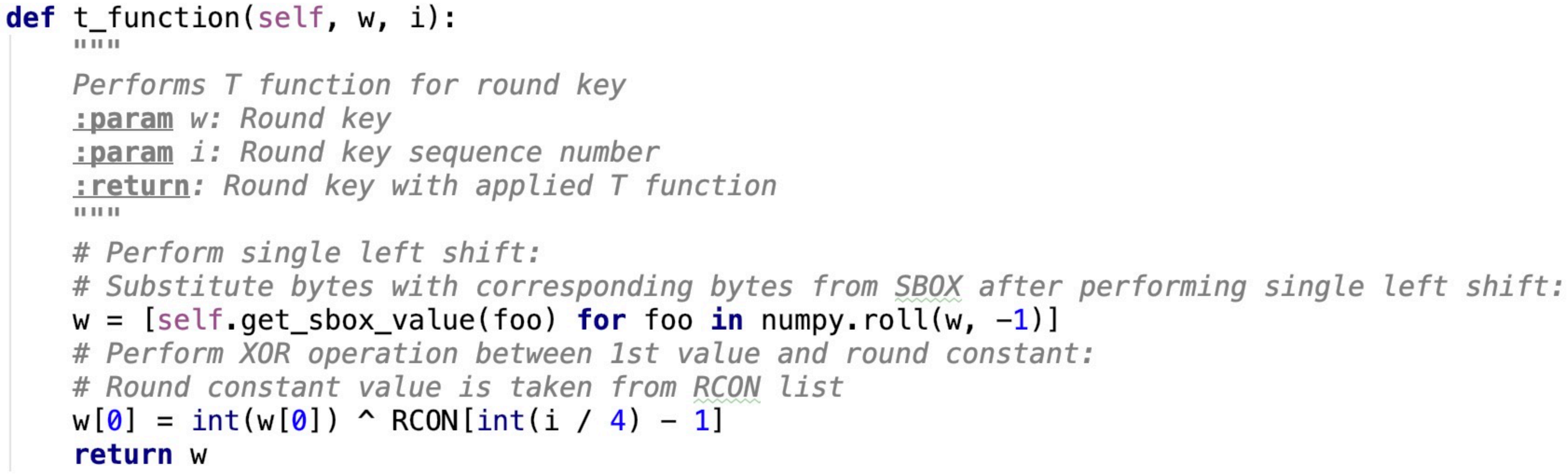
```
def generate_round_keys(self, key):
    """
    Generates all round keys
    :return: Array of all round keys (rounds 0 - 10) with
    """
    # Get initial round 4 keys by splitting main key:
    all_keys = self.split_array(key, 4)
    # Perform operations for the next 40 keys:
    for i in range(4, 44):
        # Get 1st key for new key generation:
        w1 = all_keys[i - 4]
        # Get 2nd key for key generation:
        # If it's sequence number is 4 multiple, perform T function on it,
        # else select key according to the rules
        w2 = self.t_function(all_keys[i - 1], i) if i % 4 == 0 else all_keys[i - 1]

        all_keys.append([w1[foo] ^ w2[foo] for foo in range(4)])
    return self.split_array(all_keys, 4)
```

$W(i) = \boxed{W(i-4)} \oplus \boxed{W(i-1)}$

<u>If i is a multiple of 4:</u>

$W(i) = \boxed{W(i-4)} \oplus \boxed{T(W(i-1))}$

T - function



```python
def t_function(self, w, i):
    """
    Performs T function for round key
    :param w: Round key
    :param i: Round key sequence number
    :return: Round key with applied T function
    """
    # Perform single left shift:
    # Substitute bytes with corresponding bytes from SBOX after performing single left shift:
    w = [self.get_sbox_value(foo) for foo in numpy.roll(w, -1)]
    # Perform XOR operation between 1st value and round constant:
    # Round constant value is taken from RCON list
    w[0] = int(w[0]) ^ RCON[int(i / 4) - 1]
    return w
```

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

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