Segurança Computacional - Trabalho 2

Gabriel Cruz Vaz Santos - 200049038

Descrição do código - Transformações e Funções auxiliares

O programa lê um arquivo texto para ser cifrado, no entanto deve-se garantir que o conteúdo desse arquivo seja um múltiplo de 16 bytes. Caso não seja, a função generatePadMessage preenche os dados com zero.

```
def generatePadMessage(message):
  messageOriginalSize = len(message)
  padMessageSize = messageOriginalSize
  if(messageOriginalSize % 16 != 0):
    padMessageSize = (messageOriginalSize // 16 + 1)*16
  padMessage = [None for i in range(padMessageSize)]
  for i in range(padMessageSize):
    if(i >= messageOriginalSize):
      padMessage[i] = 0
    else:
      padMessage[i] = message[i]
  return padMessage
```

A função keyExpansion recebe uma chave de 16 bytes e retorna uma de 176, no qual a cada 16 bytes temos uma *chave de round* designada a transformação **add round key**.

```
def keyExpansion(key):
    keySchedule = [None for i in range(176)]
    rConIteration = 1
    for i in range(16):
      keySchedule[i] = key[i]
    byteGenerated = 16
    while(byteGenerated < 176):
      temp = [keySchedule[byteGenerated - 4], keySchedule[byteGenerated - 3], keySchedule[byteGenerated - 2], keySchedule[byteGenerated - 1]]
      if (byteGenerated % 16 == 0):
        temp =  Utils.xorLists(Transformation.substituteBytes( Utils.shift( temp ) ), [rCon[rConIteration], 0, 0, 0])
        rConIteration += 1
      keySchedule[byteGenerated] = keySchedule[byteGenerated - 16] ^ temp[0]
      keySchedule[byteGenerated + 1] = keySchedule[byteGenerated + 1 - 16] ^ temp[1]
      keySchedule[byteGenerated + 2] = keySchedule[byteGenerated + 2 - 16] ^ temp[2]
      keySchedule[byteGenerated + 3] = keySchedule[byteGenerated + 3 - 16] ^ temp[3]
      byteGenerated += 4
    return keySchedule
```

O deslocamento de bytes e a multiplicação de galouis no modelo exponencial também possuem funções auxiliares.

```
def shift(word, n=1):
  return word[n:]+ word[0:n]

def galoisMulti(a, b):
  p = 0
  hiBitSet = 0
  for i in range(8):
      if b & 1 == 1:
          p ^= a
      hiBitSet = a & 0x80
      a <<= 1
      if hiBitSet == 0x80:
          a ^= 0x1b
      b >>= 1
  return p % 256
```

Para selecionar a chave round dentre resultante da keyExpansion e também fazer o a operação xor entre duas listas. Nesse caso, é usada na tranformação addRoundKey para fazer o xor entre o estado e a chave de round.

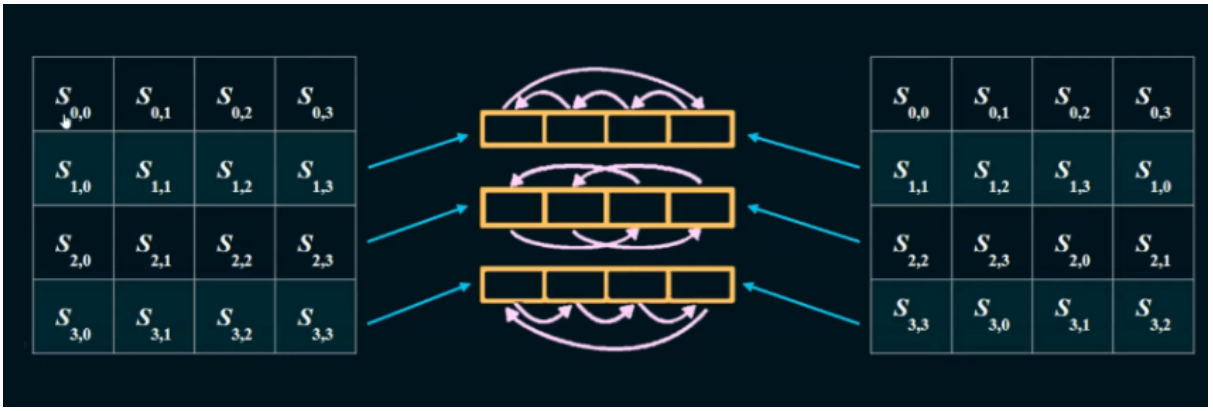
```
def xorLists(list1, list2):
  newList = [None for i in range(len(list1))]
  for i in range(len(list1)):
    newList[i] = list1[i] ^ list2[i]
  return newList

def extractRoundKey(expanded_key, round):
  return expanded_key[round*16:round*16+16]
```

```
def addRoundKey(state, roundKey):
  for i in range(len(state)):
    state[i] = state[i] ^ roundKey[i]
  return state
```

A transformação substituiteBytes substituir os bytes do nosso estado. Temos uma tabela de substituição de 16x16 bytes chamada s-box, que é uma constante. Sua inversa substitui com a matriz inversa de s-box.

A transformação shiftRows faz o deslocamentos das linhas do estado de 16 bytes:



A transformação Mix Column é a mais complexa das transformações, no qual cada coluna é tratada como um polinômio de quatro termos e que operam sobre multiplicação e operação xor em uma matriz constante já determinada.

```
class Transformation:

  def substituteBytes(state):
    for i in range(len(state)):
      state[i] = sBox[state[i]]
    return state

  def shiftRows(state):
    return [
        state[0], state[5], state[10], state[15],
        state[4], state[9], state[14], state[3],
        state[8], state[13], state[2], state[7],
        state[12], state[1], state[6], state[11]
    ]


  def mixColumn(column):
    temp = copy(column)
    mixedColumn = [
    Utils.galoisMulti(temp[0],2) ^ Utils.galoisMulti(temp[3],1) ^ \
                Utils.galoisMulti(temp[2],1) ^ Utils.galoisMulti(temp[1],3),
    Utils.galoisMulti(temp[1],2) ^ Utils.galoisMulti(temp[0],1) ^ \
                Utils.galoisMulti(temp[3],1) ^ Utils.galoisMulti(temp[2],3),
    Utils.galoisMulti(temp[2],2) ^ Utils.galoisMulti(temp[1],1) ^ \
                Utils.galoisMulti(temp[0],1) ^ Utils.galoisMulti(temp[3],3),
    Utils.galoisMulti(temp[3],2) ^ Utils.galoisMulti(temp[2],1) ^ \
        Utils.galoisMulti(temp[1],1) ^ Utils.galoisMulti(temp[0],3)
    ]
    return mixedColumn
```

```python
def mixColumns(state):
  mixedColumns = [None for i in range(len(state))]
  for i in range(4):
    column = state[i * 4: i * 4 + 4]
    mixedColumn = Transformation.mixColumn(column)
    mixedColumns[i * 4: i * 4 + 4] = mixedColumn
  return mixedColumns

def addRoundKey(state, roundKey):
  for i in range(len(state)):
    state[i] = state[i] ^ roundKey[i]
  return state


def invSubstituteBytes(state):
  for i in range(len(state)):
    state[i] = sBoxInv[state[i]]
  return state

def invShiftRows(state):
  return [
      state[0], state[13], state[10], state[7],
      state[4], state[1], state[14], state[11],
      state[8], state[5], state[2], state[15],
      state[12], state[9], state[6], state[3]
  ]

def invMixColumn(mixedColumn):
  temp = copy(mixedColumn)
  column = [
  Utils.galoisMulti(temp[0],14) ^ Utils.galoisMulti(temp[3],9) ^ \
             Utils.galoisMulti(temp[2],13) ^ Utils.galoisMulti(temp[1],11),
  Utils.galoisMulti(temp[1],14) ^ Utils.galoisMulti(temp[0],9) ^ \
             Utils.galoisMulti(temp[3],13) ^ Utils.galoisMulti(temp[2],11),
  Utils.galoisMulti(temp[2],14) ^ Utils.galoisMulti(temp[1],9) ^ \
             Utils.galoisMulti(temp[0],13) ^ Utils.galoisMulti(temp[3],11),
  Utils.galoisMulti(temp[3],14) ^ Utils.galoisMulti(temp[2],9) ^ \
      Utils.galoisMulti(temp[1],13) ^ Utils.galoisMulti(temp[0],11)
  ]
```

Descrição do código - Cifração e Decifração

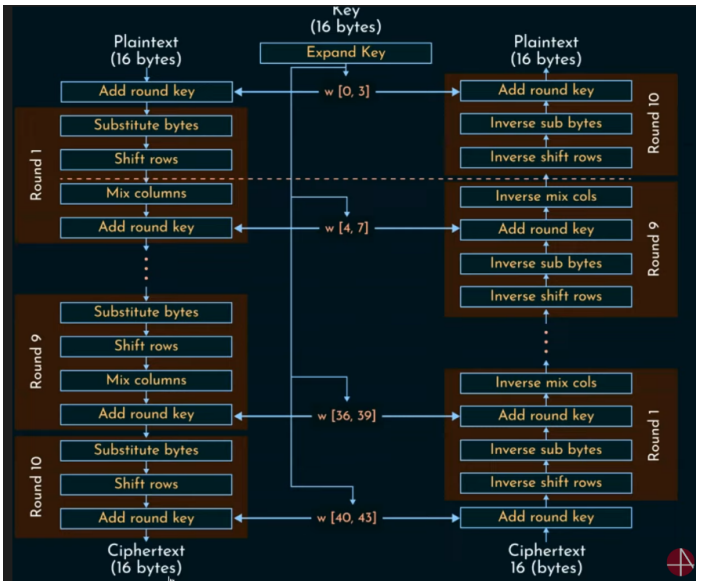


Diagrama de cifração e decifração

```
class AesCipher:

  def encrypt(message, expandableKeys, rounds=10):
    states = [message[i:i+16] for i in range(0, len(message), 16)]
    cipherStates = []
    for x in range(len(states)):
      state = Transformation.addRoundKey(states[x], Utils.extractRoundKey(expandableKeys, 0))

      for i in range(rounds - 1):
        state = Transformation.substituteBytes(state)
        state = Transformation.shiftRows(state)
        state = Transformation.mixColumns(state)
        state = Transformation.addRoundKey(state, Utils.extractRoundKey(expandableKeys, i + 1))


      state = Transformation.substituteBytes(state)
      state = Transformation.shiftRows(state)
      state = Transformation.addRoundKey(state, Utils.extractRoundKey(expandableKeys, 10))

      for i in range(len(state)):
        cipherStates.append(state[i])

    return cipherStates

  def decrypt(state, expandableKeys, rounds=10):
    states = [state[i:i+16] for i in range(0, len(state), 16)]
    decryptStates = []
    for x in range(len(states)):
      state = Transformation.addRoundKey(states[x], Utils.extractRoundKey(expandableKeys, 10))

      for i in range(rounds - 1):
        state = Transformation.invShiftRows(state)
        state = Transformation.invSubstituteBytes(state)
        state = Transformation.addRoundKey(state, Utils.extractRoundKey(expandableKeys, 10 - i -1))
        state = Transformation.invMixColumns(state)
      state = Transformation.invShiftRows(state)
      state = Transformation.invSubstituteBytes(state)
      state = Transformation.addRoundKey(state, Utils.extractRoundKey(expandableKeys, 0))

      for i in range(len(state)):
        decryptStates.append(state[i])

    return decryptStates
```