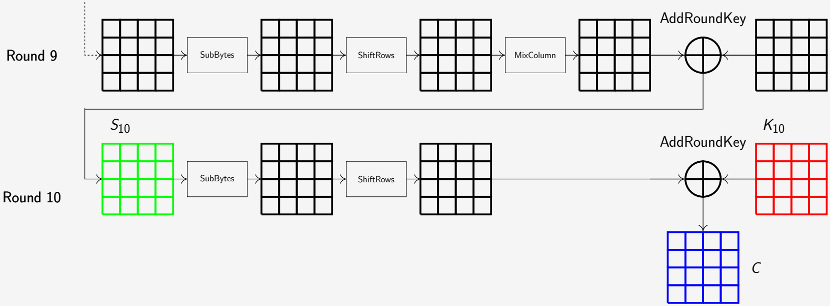
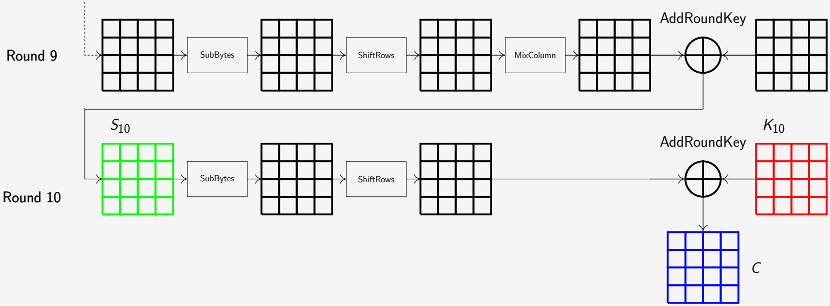
AES-128 Assignment 1



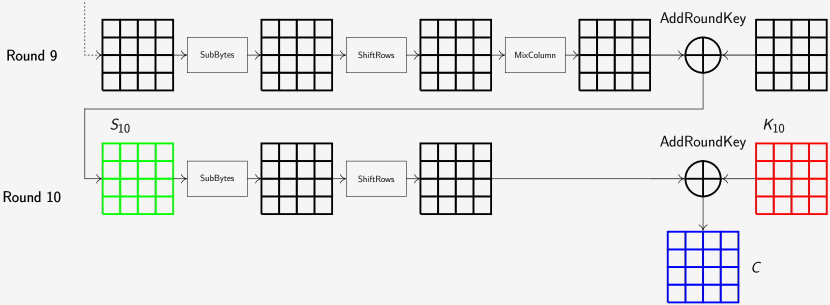
- A developer left a debug option that prints the state 10 (S_{10} shown in green) and a ciphertext C shown in blue. What can be wrong with that?

AES-128 Assignment 1



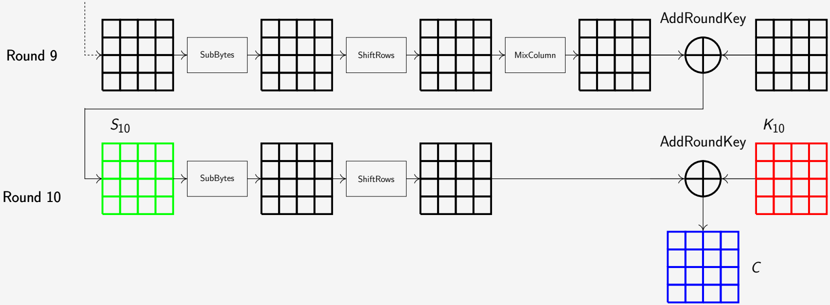
- A developer left a debug option that prints the state 10 (S_{10} shown in green) and a ciphertext C shown in blue. What can be wrong with that?
- Find K_{10} and then get the master key using the provided code

AES-128 Assignment 1



- A developer left a debug option that prints the state 10 (S_{10} shown in green) and a ciphertext C shown in blue. What can be wrong with that?
- Find K_{10} and then get the master key using the provided code
- $C = ShiftRows(SubBytes[S_{10}]) \oplus K_{10}$

AES-128 Assignment 1



- A developer left a debug option that prints the state 10 (S_{10} shown in green) and a ciphertext C shown in blue. What can be wrong with that?
- Find K_{10} and then get the master key using the provided code
- $C = ShiftRows(SubBytes[S_{10}]) \oplus K_{10}$
- $K_{10} = C \oplus ShiftRows(SubBytes[S_{10}])$

What to do: tip 1

- Read the description of the task
- There is a file called sca_training.py - the file contains all AES operations required for the assignment (you don't need to implement anything)
- The first thing you need to do is to get a ciphertext and a trace in two separate variables: just run the code in a proper cell
- Compute the equation: $K_{10} = C \oplus ShiftRows(SubBytes[S_{10}])$
- Run the code to get the entire AES key-schedule: key_schedule = sca_training.inverse_key_expansion(key)
- Get the ASCII of the 0 round key (master key)

What to do: tip 2

```
output, ctext, trace = binary_aes128_encrypt("0102030405060708090000a0b0c0d0e1f", verbose=False)

#Define numpy structure to keep the recovered key
key = np.zeros((16), dtype=np.uint8)

key_cand = np.arange(256).astype(np.uint8)

#The global idea of side-channel attacks is to find a model, i.e., that takes known data, such as ciphertext bytes, and unknown key data (one key b
#predicts a leakage, which is measured by other means.
#In this example the leakage is a raw 10th round input, so the model is equal to invShiftRows[invSbox[k ^ c[i,j]]], where k is a key byte
# c is a ciphertext byte, i is an encrytion index, j is a byte index.
#In this example the model shall be strictly equal to the leakage when the key is correct, i.e. trace[v] = invShiftRows[invSbox[k ^ c[i,j]]].
#Since the invShiftRows operation is involved the indexes v and j are not necessary eqaul.
#This expression can be reqritten as: ShiftRows[trace][j] = invSbox[k ^ c[i,j]]
#When indexes v and j correspond to each other (due to the ShiftRows operation they are slightly shuffled), this expression for the correct key is
#but for the wrong key this expression is strictly wrong (in this specific example).
#Therefore, the idea of the attack is to compute the model invSbox[k ^ c[i,j]] or invShiftRows[invSbox[k ^ c[i,j]]] and compare the model with the
#leakage value.
#When the idexes v and j are selected correctly then in this example one encryption is enough to get the key.
#Shift the leakage samples so that they are aligned with the ciphertext bytes: ShiftRows[trace]
shifted_trace = sca_training.shift_rows(trace)

#Find all the key bytes one by one
for iByte in range(16):

    #Xor all the key candidates with the ciphertext byte: k ^ c[i,j]
    #k is a matrix of 256 elements (key candidate from 0 to 255)
    #i is equal to 0 (only one encrytion is needed)
    #j is iByte
    sbox_out = np.bitwise_xor(key_cand, ctext[iByte])

    #Apply inverse Sbox: invSbox[k ^ c[i, j]]
    sbox_in  = sca_training.invSbox[sbox_out]

    #Find the key (position) where the trace is equal to the sbox_in: ShiftRows[trace][j] == invSbox[k ^ c[i, j]]
    #There is only one key for which the model 'sbox_in' is equal to the byte of the correctly suffled trace in this particular example
    key[iByte] = np.where(sbox_in==shifted_trace[iByte])[0]

#Get master key using the provided binary
key_schedule = sca_training.inverse_key_expansion(key)
print('Master key in hex:', key_schedule[0,0,:])
print('Master key in ASCII:', binascii.unhexlify(''.join('{:02x}'.format(c) for c in key_schedule[0,0,:])))
```

Thank you!

Roman Korkikian, Nicolas Oberli

Side-channels and Fault Attacks
February 23rd, 2023 - June 29th, 2023

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch