ECEN 759 Lab 7: Differential Fault Analysis Attack on AES

S M Farabi Mahmud

May 6, 2021

1 Preliminaries on AES

Since we are using a AES with 128 bit key length, there are 10 rounds of operation in total. First 9 rounds consists of the following operations -

- Shift Rows **SR**
- Sub Byte **SB**
- Mix Column **MC**
- Add Round Key **RK**

However, in the last round, we do not have the **MC** operation. We express the key of each i-th round as K^i The plaintext and the ciphertext is denoted by P and C respectively. M_j^i denotes the j-th byte at i-th round.

2 Reverse Key Scheduling

We know the key scheduling algorithm uses specific algorithm to generate all the round keys from the master key. We use the exact inverse order to generate the round keys from the later round keys. For this purpose, we use the following code -

```python
def gfunction(w,round_constant):
  return [sbox[w[1]] ^ round_constant,sbox[w[2]],sbox[w[3]],sbox[w[0]]]

def prev_roundkey(arr, round_constant):
  prev_arr = []
  w4 = [arr[0],arr[1],arr[2],arr[3]]
  w5 = [arr[4],arr[5],arr[6],arr[7]]
  w6 = [arr[8],arr[9],arr[10],arr[11]]
  w7 = [arr[12],arr[13],arr[14],arr[15]]

  w3 = [x^y for x,y in zip(w6,w7)]
  gw3 = gfunction(w3, round_constant)
  w2 = [x^y for x,y in zip(w5,w6)]
  w1 = [x^y for x,y in zip(w4,w5)]
  w0 = [x^y for x,y in zip(w4,gw3)]
  prev_arr.extend(w0)
  prev_arr.extend(w1)
  prev_arr.extend(w2)
  prev_arr.extend(w3)

  return prev_arr
```

3 Single Bit Fault Attack

In this section, we will describe how we have implemented the single bit fault attack on AES. We used the concept from Giraud's work [1]. In the first attack, we target the one bit of the input of the Round 10 that is denoted by M^9.

```python
def dfa_bit_fault(c, ds):
  locations = [0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80]
  recovered = []
  for j in range(16):
    count = [0]*255
    for x in range(255):
        for e in locations:
            for d in ds:
                lhs = c[0][shift_row_index(j)] ^ d[shift_row_index(j)]
                rhs = subbyte(x) ^ subbyte(x ^ e)
                if (lhs == rhs):
                    count[x] = count[x]+1
    recovered.append(np.argmax(count))
  return recovered
```

Listing 1: Main loop for the DFA Single Bit Fault Attack

For this reason, we can directly determine that the output of Round 10, the ciphertext C that follows the equation -

$$C_{SR(j)} = SB(M_j^9) \oplus K_{SR(j)}^{10} \tag{1}$$

Here, in this equation 1, we can get the output of the round 10, the ciphertext C from the input of the Round 10 M^9 and the XORing that with the round key K^{10}. If we apply a fault e_j on one bit of the j-th byte of the M^9 we can get the faulty ciphertext D following the equation 2

$$D_{SR(j)} = SB(M_j^9 \oplus e_j) \oplus K_{SR(j)}^{10} \tag{2}$$

XORing equation 1 and equation 2 we can get -

$$C_{SR(j)} \oplus D_{SR(j)} = SB(M_j^9) \oplus SB(M_j^9 \oplus e_j) \tag{3}$$

We have only one bit fault, so possible location $e_j \in \{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80\}$. Again, one bit fault at M^9 impacts only one byte of the faulty ciphertext D, we can generate all possible values for the M^9 in range $\{0, 1, \dots, 255\}$ and all possible bit locations and see if the equation 3 holds for these values. Then we determine the possible value of the recovered key from the maximum of the count.

3.1 Results

After running the code, first we get the round key for the 10th Round. From there using the Reverse Key Scheduling algorithm, we get the following results -

Round	**Round Key**
10	dfb15c7948b0a6f353222dec717e0ce7
9	a34c77ea9701fa8a1b928b1f225c210b
8	33e08df8344d8d608c93719539ceaa14
7	ff59812d07ad0098b8defcf5b55ddb81
6	539513faf8f481b5bf73fc6d0d832774
5	ff2cc7cdab61924f47877dd8b2f0db19
4	1a08bf2b544d5582ece6ef97f577a6c1
3	93330eff4e45eaa9b8abba1519914956
2	173e14cddd76e456f6ee50bca13af343
1	5d340296ca48f09b2b98b4ea57d4a3ff
Master Key	75c45b86977cf20de1d044717c4c1715

4 Byte Fault Attack

In this section we have followed the attack suggested in the paper by Michael Tunstall et al [2]. Since this attack induces faults at one of the 4 bytes of the top row of input to the eighth round, M^8. Since we will have two rounds of SR, SB and RK and one round of MC on this M^8 we can use to solve the following set of equations 4 to determine the possible values of key at Round 10 from the ciphertext c. Here SB^{-1} denotes the inverse of Shift Byte operations. x_i denotes the i-th byte of the correct ciphertext and x'_i denotes the i-th byte of the faulty ciphertext. k_i denotes the i-th byte of the Round 10 roundkey. f denotes the faulty byte location.

$$\begin{aligned}
2f &= SB^{-1}(x_0 \oplus k_0) \oplus SB^{-1}(x'_0 \oplus k_0) \\
2f &= SB^{-1}(x_1 \oplus k_1) \oplus SB^{-1}(x'_1 \oplus k_1) \\
2f &= SB^{-1}(x_2 \oplus k_2) \oplus SB^{-1}(x'_2 \oplus k_2) \\
2f &= SB^{-1}(x_3 \oplus k_3) \oplus SB^{-1}(x'_3 \oplus k_3) \\
3f &= SB^{-1}(x_4 \oplus k_4) \oplus SB^{-1}(x'_4 \oplus k_4) \\
3f &= SB^{-1}(x_5 \oplus k_5) \oplus SB^{-1}(x'_5 \oplus k_5) \\
3f &= SB^{-1}(x_6 \oplus k_6) \oplus SB^{-1}(x'_6 \oplus k_6) \\
3f &= SB^{-1}(x_7 \oplus k_7) \oplus SB^{-1}(x'_7 \oplus k_7) \\
f &= SB^{-1}(x_8 \oplus k_8) \oplus SB^{-1}(x'_8 \oplus k_8) \\
f &= SB^{-1}(x_9 \oplus k_9) \oplus SB^{-1}(x'_9 \oplus k_9) \\
f &= SB^{-1}(x_{10} \oplus k_{10}) \oplus SB^{-1}(x'_{10} \oplus k_{10}) \\
f &= SB^{-1}(x_{11} \oplus k_{11}) \oplus SB^{-1}(x'_{11} \oplus k_{11}) \\
f &= SB^{-1}(x_{12} \oplus k_{12}) \oplus SB^{-1}(x'_{12} \oplus k_{12}) \\
f &= SB^{-1}(x_{14} \oplus k_{14}) \oplus SB^{-1}(x'_{14} \oplus k_{14}) \\
f &= SB^{-1}(x_{13} \oplus k_{13}) \oplus SB^{-1}(x'_{13} \oplus k_{13}) \\
f &= SB^{-1}(x_{15} \oplus k_{15}) \oplus SB^{-1}(x'_{15} \oplus k_{15})
\end{aligned} \tag{4}$$

We iterate through all possible values of $f, k_i \in \{0, 1, \ldots, 255\}$ to determine for which value the equations of the systems of equations 4 is valid. Here for each faulty byte, we have to satisfy four equations containing four locations of the key. We can get that from the following figure 1. For example, byte 0 impacts the k_0, k_7, k_{10}, k_{13}. So we do not need to check beyond the equations related with these locations.

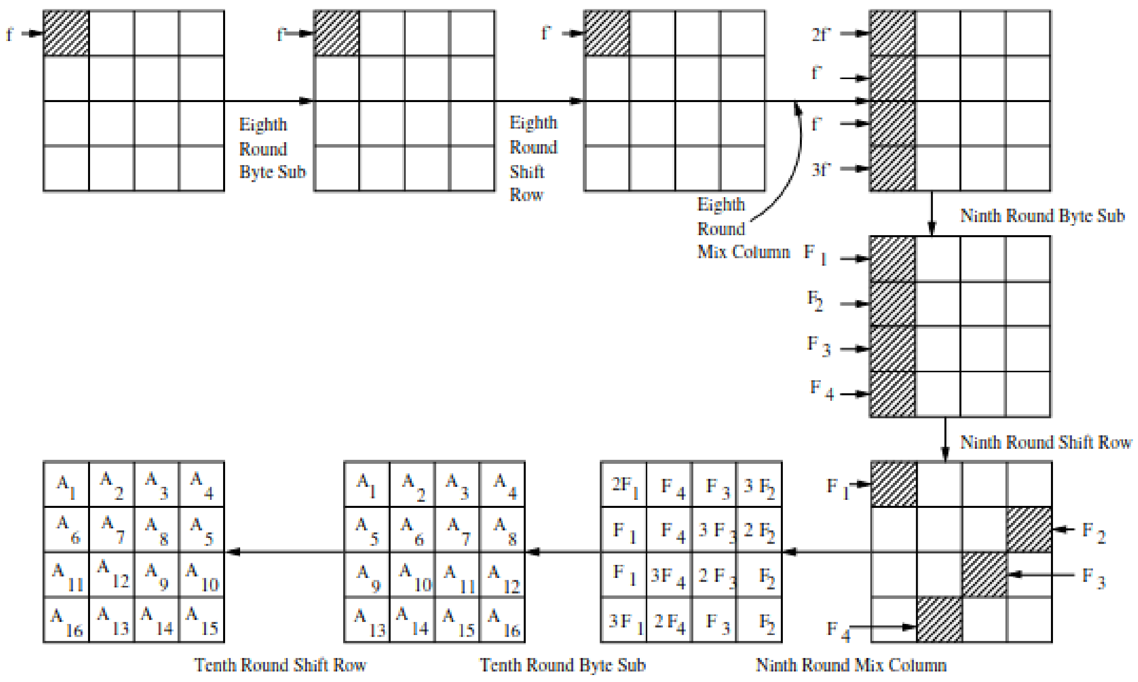

Figure 1: Change in one byte impacting 4 bytes in Round 10

To determine the round key of 10th Round, we have used the following code given at Listing 4. We have used the counting method to determine the values that appear for most of the time for the solutions that have solved the equations 4

```python
def task2():
    master_key = ''
    input_filename = "set2.txt"
    correct_ciphers= []
    faulty_ciphers = []
    correct_ciphers, faulty_ciphers = process_input(input_filename)
    data = [[] for i in range(4)]
    c = correct_ciphers[0]

    for i in range(20):
        d = faulty_ciphers[i]
        x = dfa_byte_fault(c,d)
        fault_location = find_fault_byte_index(c,d)
        if (x is []):
            continue
        else:
            if len(data[fault_location]) == 0:
                data[fault_location] = np.array(x)
            else:
                data[fault_location] = np.append(data[fault_location],x,axis=0)

    guess = np.array([])
    for i in range(4):
        if len(data[i]) != 0:
            if len(guess)==0:
                guess = np.array(find_most_common_elem(data[i]))
                guess = guess.reshape(1,16)
            else:
                guess =
                ↪  np.append(guess,np.array(find_most_common_elem(data[i])).reshape(1,16),axis=0)
    r10_key = guess.max(axis=0).tolist()
    master_key = get_master_key_from_recovered(bytes_to_str(r10_key))
    print("Master Key {}".format(master_key))
    return master_key
```

4.1 Results

After running the code, first we get the round key for the 10th Round. From there using the Reverse Key Scheduling algorithm, we get the following results -

Round	**Round Key**
10	21a689161102fd820e0336150022956e
9	eaaca8bd30a474941f01cb970e21a37b
8	46e9663fda08dc292fa5bf03112068ec
7	51e7b98d9ce1ba16f5ad632a3e85d7ef
6	256a1f92cd06039b694cd93ccb28b4c5
5	465686a8e86c1c09a44adaa7a2646df9
4	67ffdec7ae3a9aa14c26c6ae062eb75e
3	5f5c5211c9c54466e21c5c0f4a0871f0
2	a18444d3969916772bd91869a8142dff
1	1e12d43f371d52a4bd400e1e83cd3596
Master Key	42f0108d290f869b8a5d5cba3e8d3b88

References

[1] Giraud, C. Dfa on aes. In *International Conference on Advanced Encryption Standard* (2004), Springer, pp. 27–41.

[2] Tunstall, M., Mukhopadhyay, D., and Ali, S. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices* (2011), Springer, pp. 224–233.