System Security (201700086)
Lab 2 - Side Channel Attacks

Æde Symen Hoekstra, Luigi Coniglio (Group 24)

June 21, 2018

Note to the reader: All code provided in this report its also given in the zip archive. The code has been tested under python version 2.7.14.

1 Breaking AES with CPA

For this part of the assignment we were given two sets of plaintext/ciphertext pairs and the corresponding power traces representing the hamming weight of the whole AES-128 state after each round.
Our goal was to perform a Correlation Power Attack (CPA) to recover the key.

In our implementation we decided to attempt a CPA attack on the last round (thus recovering the last round-key). Given that the key-schedule algorithm is reversible, recovering any of the round-keys gives us access to the master key.

In order to recover the last round-key we attacked each byte separately.
The attack is very simple: given a possible value that the target round-key byte could have, for every ciphertext we compute an hypothetical Hamming weight starting from the value of the target byte in the ciphertext and traversing backward the last AES round. Afterward we compute the Pearson correlation coefficient of the obtained hamming weight with respect of the Hamming weight of the state after the 9th round which is given in the traces.

We do this operation for every possible value of the target round-key byte. If the attack is successful the value of the target round-key byte is among those with the highest correlation.

The following code shows our implementation of the attack:

```
1  import pandas
2  from scipy import stats
3
4  sboxInv = [
5          0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, ...
6  ]
7
8
```

```
9  def hw(x):
10     """Returns the Hamming weight of x"""
11     return bin(x).count("1")
12
13 def read_hexfile(filename):
14     """Reads and decode a file containing hex values"""
15     output = []
16     with open(filename,"r") as f:
17         for line in f:
18             line = line[:-1] # Remove newline
19             output.append(line.decode("hex"))
20     return output
21
22
23 def get_byte_guesses(hws_state,ctxts,i):
24     """Returns a list of guesses for byte i sorted by correlation"""
25     byte_guesses = [] # A list of tuples (key, correlation)
26
27     # Compute Paerson correlation for each possible byte value
28     for k in range(0xff):
29
30
31         # For each ciphertext ct recover the HW
32         # of byte i just after the 9th round
33         hws_byte = []
34         for ct in ctxts:
35             hws_byte.append(hw(sboxInv[ord(ct[i]) ^ k]))
36
37         # Compute correlation with the HW in the traces
38         corr = stats.pearsonr(hws_byte, hws_state)[0]
39         byte_guesses.append((k,corr))
40         byte_guesses.sort(reverse=True, key=lambda x: x[1])
41
42     return byte_guesses
43
44 def print_best_guesses(byte_guesses,n):
45     for i in range(n):
46         print "%02x (%f)" % byte_guesses[i],
47     print ''
48
49 def main():
50     tracecs_file = "traces_low_noise.csv"
51     ciphertext_file = "ct_low_noise.csv"
52
53     # Read ciphertexts
54     ctxts = read_hexfile(ciphertext_file)
55
56     # Read HWs at the end of the 9th round
57     hws_state = pandas.read_csv(tracecs_file,header=None).ix[:,8]
58
59     # Get list of guesses for each key byte
60     guesses = [get_byte_guesses(hws_state,ctxts,i) for i in range(16)]
```

```
61
62      # Time to show the best guesses
63      n = 3
64      print "### Top %d round key bytes guesses ###" % n
65      for i in range(16):
66          print "---- Byte: %d ----" % i
67          print_best_guesses(guesses[i],n)
68
69      print ("Top last round key guess: " +
70             ''.join('%02x'% x[0][0] for x in guesses)
71            )
72
73
74  if __name__ == '__main__':
75      main()
```

The function *get_byte_guesses* is used to perform the CPA attack on a single byte of the round-key. For each round-key byte guess it generates a list of Hamming weights (lines 31 to 35) and correlates those values with the Hamming weights from the traces (lines 37 to 40).

As we can see the value of the byte before the last round is recovered by simply adding the round-key byte guess and reverting the *Sbox* step. Notice that we don't need to revert the *ShiftRows* step in the last round since we are computing the correlation with respect of the whole state and therefore knowing the exact position of the target byte in the state doesn't add anything to our current attack[1].

1.1 Results

We performed our attack on both sets of traces (low and high noise). In order to visualize the result of our attack we shows for each round-key byte guess the three best guesses sorted by correlation. The result from the first set are the following:

```
1   $ python cpa_attack.py
2   ### Top 3 round key bytes guesses ###
3   ---- Byte: 0 ----
4   13 (0.253623) c2 (0.065183) 48 (0.045315)
5   ---- Byte: 1 ----
6   11 (0.241911) c0 (0.058782) a2 (0.043074)
7   ---- Byte: 2 ----
8   1d (0.256735) cc (0.061005) 4a (0.041849)
9   ---- Byte: 3 ----
10  7f (0.243229) ae (0.063601) c1 (0.044102)
11  ---- Byte: 4 ----
12  e3 (0.246699) 32 (0.063266) a9 (0.052562)
13  ---- Byte: 5 ----
```

[1] This would not be the case if the traces provided a separate Hamming weight for each byte in the state.

```
94 (0.254822) 45 (0.054904) 32 (0.042893)
---- Byte: 6 ----
4a (0.246581) 9b (0.061095) d3 (0.044476)
---- Byte: 7 ----
17 (0.252474) b1 (0.047139) c6 (0.043401)
---- Byte: 8 ----
f3 (0.268915) b9 (0.055929) 9c (0.053205)
---- Byte: 9 ----
07 (0.253567) 5c (0.051501) 37 (0.042667)
---- Byte: 10 ----
a7 (0.257640) c8 (0.051467) 00 (0.047678)
---- Byte: 11 ----
8b (0.248080) 38 (0.060782) 5a (0.057695)
---- Byte: 12 ----
4d (0.253083) 16 (0.055285) 9c (0.052790)
---- Byte: 13 ----
2b (0.254145) fa (0.057442) ba (0.054998)
---- Byte: 14 ----
30 (0.257627) e1 (0.065092) ab (0.040667)
---- Byte: 15 ----
c5 (0.247578) 9e (0.066701) 62 (0.041970)
Top last round key guess: 13111d7fe3944a17f307a78b4d2b30c5
```

As we can see there is a clear gap between the correlation produced by the first byte and all the others. This means that the attack was successful and the combination of the top round-key bytes guesses most likely correspond to the value of the last round-key.

We performed the attack also on the traces with high noise. The results are the following:

```
$ python cpa_attack.py
### Top 3 round key bytes guesses ###
---- Byte: 0 ----
13 (0.120611) 48 (0.032892) 92 (0.031175)
---- Byte: 1 ----
11 (0.122619) a2 (0.030497) 82 (0.029544)
---- Byte: 2 ----
1d (0.130443) cc (0.032119) 8e (0.029534)
---- Byte: 3 ----
7f (0.128183) ae (0.029667) 24 (0.028838)
---- Byte: 4 ----
e3 (0.136248) 32 (0.039104) d3 (0.034425)
---- Byte: 5 ----
94 (0.122106) 2a (0.045962) 45 (0.033237)
---- Byte: 6 ----
4a (0.106523) fc (0.036134) de (0.029086)
---- Byte: 7 ----
17 (0.128616) a9 (0.033944) b1 (0.030140)
---- Byte: 8 ----
f3 (0.149074) 9c (0.053178) 0f (0.037192)
```

```
---- Byte: 9 ----
07 (0.132963) a1 (0.033461) 2d (0.031104)
---- Byte: 10 ----
a7 (0.119922) 14 (0.039260) 80 (0.031200)
---- Byte: 11 ----
8b (0.126884) 38 (0.036175) c1 (0.036078)
---- Byte: 12 ----
4d (0.122547) b6 (0.032882) 9c (0.031743)
---- Byte: 13 ----
2b (0.138580) 8c (0.033568) 0d (0.028357)
---- Byte: 14 ----
30 (0.115152) e1 (0.028392) e7 (0.027051)
---- Byte: 15 ----
c5 (0.110752) 9e (0.031914) 14 (0.031016)
Top last round key guess: 13111d7fe3944a17f307a78b4d2b30c5
```

This time the gap in the correlation value is much smaller than before, however the attack is still successful and we are able to easily recover the value of the last round-key.

1.2 Recovering the key

Recovering the master key from one of the round-keys is straight forward. The operation used in the key schedule algorithm are the following:

- A rotation of 8-bit on each 32-bit word
- A rcon operation (i.e. an exponentiation of 2 in the Galois field).
- A Rijndael's S-box operation

All those operation are invertible, therefore given any key expansion we can go back and recover the master key.
For this step we decided to use a tool[2] that does exactly what we want to achieve. Given the last round-key we are in this way able to recover the master key (highlighted):

```
$ ./aes_keyschedule 13111d7fe3944a17f307a78b4d2b30c5 10
K00: 000102030405060708090A0B0C0D0E0F
K01: D6AA74FDD2AF72FADAA678F1D6AB76FE
K02: B692CF0B643DBDF1BE9BC5006830B3FE
K03: B6FF744ED2C2C9BF6C590CBF0469BF41
K04: 47F7F7BC95353E03F96C32BCFD058DFD
K05: 3CAAA3E8A99F9DEB50F3AF57ADF622AA
K06: 5E390F7DF7A69296A7553DC10AA31F6B
K07: 14F9701AE35FE28C440ADF4D4EA9C026
K08: 47438735A41C65B9E016BAF4AEBF7AD2
K09: 549932D1F08557681093ED9CBE2C974E
K10: 13111D7FE3944A17F307A78B4D2B30C5
```

[2] https://github.com/SideChannelMarvels/Stark

1.3 Other possible approaches

In our attack we used the ciphertext and recovered the last round-key, another possible approach would be to use the plaintext to recover the key. While it is still possible this technique is more difficult than the one we used. This is due to the *MixColumn* step which combines four bytes of the state.

In order to perform such attack one could for example attack four bytes at a time.

2 Template Attacks

For this part of the assignment we implemented template attack. We tested our attack on two different datasets: a dataset containing hamming weight measurements and corresponding points of interest in the power trace, and another set containing a list of output values.

The idea behind a basic template attack is very simple: for every possible candidate value that we would like to predict (e.g. Hamming weight) we combine all points of interests[3] to build a multivariate probability distribution measuring the likelihood of that value to occur. More formally, we build a function that given as input a power trace (i.e. points of interests) tells us how likely is that a certain value generated that power trace.
By comparing the output of different probability distribution functions (PDF) we can estimate if, for a given power trace, a value is more likely than another. It follows that by building and comparing the PDFs of all possible values we can estimate what value is most probably behind a given power trace, this is exactly what we did for this part of the assignment.

In practice for each value we combined all measurements and calculated the mean and variance-covariance matrix that were later used to create the probability distribution function. This is how our code for the template attack on Hamming wheight model looks like:

```
1  import numpy as np
2  import pandas as pd
3  import scipy
4  from scipy.stats import multivariate_normal
5  from sklearn.metrics import accuracy_score
6
7  # Let's first load the data
8  mhw = pd.read_csv("model_HW.csv",header=None)
9  thw = pd.read_csv("traces_50_HW.csv",header=None,sep='\s+')
10
11 trainsize = 25000
12 y_train = mhw.iloc[:trainsize]
13 x_train = thw.iloc[:trainsize]
```

[3] To simplify one could think of the points of interests as the most representative parts of the power traces, but this could vary depending on the nature of the template attack.

```
14  y_test  = mhw.iloc[trainsize:]
15  x_test  = thw.iloc[trainsize:]
16
17
18  # Build pdf for each HW
19  pdfs = []
20  for hw in range(0,9):
21      mean_matrix = x_train[y_train[0] == hw].mean()
22      cov_matrix  = scipy.cov(x_train[y_train[0] == hw],rowvar=False)
23      pdfs.append(multivariate_normal(mean_matrix,cov_matrix))
24
25  # Try to predict HW using a single trace (we just pick the best guess)
26  predict = lambda trace: np.argmax(map(lambda p: p.pdf(trace), pdfs))
27  y_pred  = x_test.apply(predict,axis=1)
28  print accuracy_score(y_test,y_pred)
29
30  # Try to predict HW using a using multiple traces
31  # (see https://wiki.newae.com/Template_Attacks "Combining the Results")
32  y_pred = []
33  for hw in range(0,9):
34      tmp = []
35      for p in pdfs:
36          probs = p.pdf(x_test[y_test[0] == hw])
37          tmp.append(reduce(lambda x,y: x+np.log(y), probs,0))
38      y_pred.append(np.argmax(tmp))
39
40  print accuracy_score(range(0,9),y_pred)
```

While using a template attack is theoretically possible to recover the key using a single power trace, this is not always feasible. In a more relaxed context one would rather use multiple power traces from the victim in order to retrieve the key.

The first attack in our code (lines 25 to 28) aims at predicting the hidden value using a single power trace, in the last part of our code (from line 30) we implemented this second typology of attack.
The complete code, which contain also the attack for the value model, can be found in the zip repository.

2.1 Results

Table 1 shows the performance of our attack when using 25 000 measurements for profiling. While the attack performs very well on better quality traces such as the one provided during the lab session, its accuracy is much closer to random guess when using the traces containing random delays.
As shown by the results, attacking multiple traces at the same time (column Combined) can be helpful.

	HW model		Value model	
	Single trace	Combined	Single trace	Combined
Lab traces	0.81	0.78	0.10	0.25
Assignment traces	0.14	0.11	0.004	0.004

Table 1: Accuracy score TA using 25 000 traces for profiling,

Table 2 shows the same attack performed on a lower number of measurements. Here the performances are slightly degraded for the HW model, while the accuracy of the TA on the value model remains almost unchanged.

	HW model		Value model	
	Single trace	Combined	Single trace	Combined
Lab traces	0.79	0.77	0.10	0.25
Assignment traces	0.14	0.11	0.004	0.009

Table 2: Accuracy score TA using 10 000 traces for profiling,