Side-Channel Attacks Lab Assignment

Boming Zhang

Lab Report

In this lab, I have learned how to set up a running ChipWhisperer on a Windows computer. To try different attack methods, I modified the attacking scripts by switching the substitution-box function to last-round-state-function, adding jitter, and customizing noise. Interpreting results is also something that I learned more. I wasn't sure about the PGE before the assignment, but the lack of understanding stopped me from understanding the results I got from attacks I made. O'Flynn has a great explanation:" The ‘guessing entropy’ is defined as the “*average number of successive guesses required with an optimum strategy to determine the true value of a random variableX”...... The ‘partial’ refers to the fact that we are finding the guessing entropy on each subkey. This gives us a PGE for each of the 16 subkeys*."[1] After reading this, I knew how to interpret my horrifying looking results-table. I also dug into the documentation of the CWAnalysis, so that I knew how to add noise and jitter in the data preprocessing process.

```
import chipwhisperer as cw
from chipwhisperer.analyzer.attacks.cpa import CPA
from chipwhisperer.analyzer.attacks.cpa_algorithms.progressive import CPAProgressive
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit, SBox_output, LastroundStateDiff
from chipwhisperer.analyzer.preprocessing.add_noise_random import AddNoiseRandom

#self.project = cw.openProject("2017-mar23-xmega-aes.cwp")
traces = self.project.traceManager()

#Example: If you wanted to add noise, turn the .enabled to "True"
self.ppmod[0] = AddNoiseRandom()

#My spire id is:32581750,I work on my own, so the noise level should be: 1/100
self.ppmod[0].noise = 0.01
self.ppmod[0].enabled = True

attack = CPA()
leak_model = AES128_8bit(LastroundStateDiff)
attack.setAnalysisAlgorithm(CPAProgressive, leak_model)
```

This screenshots shows how I imported the LastroundStateDiff function and used it to target the last round key. Adding noise was fairly simple, I just had to set False to True in the template file.

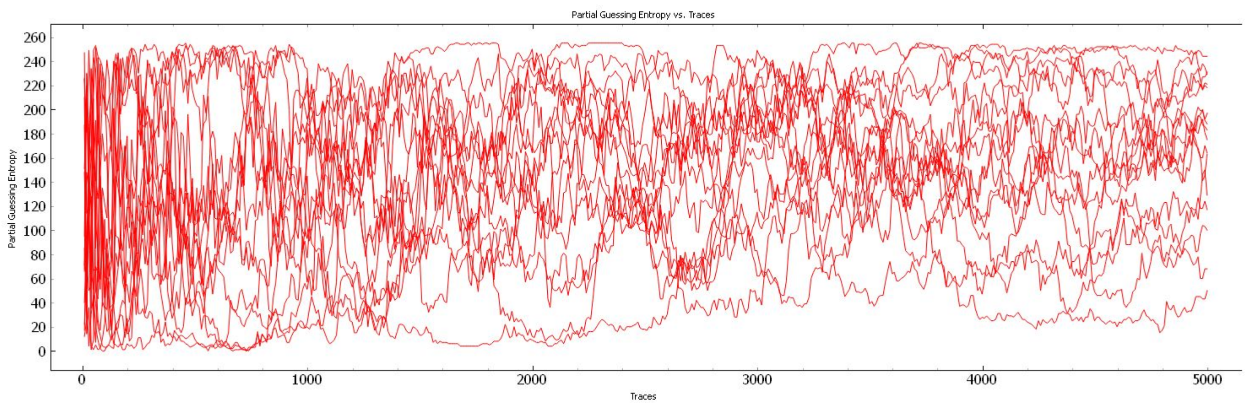
```
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit, SBox_output, LastroundStateDiff
from chipwhisperer.analyzer.preprocessing.add_noise_jitter import AddNoiseJitter

#self.project = cw.openProject("2017-mar23-xmega-aes.cwp")
traces = self.project.traceManager()

#Example: If you wanted to add noise, turn the .enabled to "True"
self.ppmod[0] = AddNoiseJitter()

#My spire id is:32581750,I work on my own, so the jitter level should be: 3
self.ppmod[0].jitter = 3
self.ppmod[0].enabled = True

attack = CPA()
leak_model = AES128_8bit(LastroundStateDiff)
```

This screenshots shows how I imported and used the Add jitter function. Adding jitter is little tricker than adding noise, since it requires an integer as a parameter, unlike the noise one. However, the Lab pdf gave us a hint since it wanted us to use a module number.

Let's take a look at my results.

Default CPA:

	0	**1**	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	221	175	117	183	197	50	231	68	164	163	100	244	129	188	229	218
0	57 0.0665	1A 0.0573	FF 0.0597	AB 0.0549	92 0.0661	9E 0.0607	86 0.0594	36 0.0550	E7 0.0657	61 0.0628	E2 0.0578	FE 0.0618	1D 0.0553	FA 0.0632	8E 0.0613	D7 0.0595
1	B3 0.0584	21 0.0561	77 0.0583	55 0.0545	01 0.0557	B7 0.0556	5E 0.0583	43 0.0521	70 0.0563	B4 0.0516	9E 0.0493	04 0.0531	E2 0.0538	BB 0.0630	F0 0.0555	1B 0.0537
2	DB 0.0573	B9 0.0542	84 0.0562	AE 0.0512	FD 0.0549	0C 0.0551	4D 0.0547	2A 0.0502	5D 0.0563	16 0.0490	38 0.0490	F4 0.0519	F5 0.0514	F0 0.0594	EE 0.0535	E8 0.0519
3	F1 0.0572	8E 0.0539	34 0.0525	7C 0.0497	17 0.0542	AF 0.0539	1D 0.0535	39 0.0498	1C 0.0541	51 0.0489	84 0.0490	B4 0.0517	32 0.0512	CD 0.0573	63 0.0527	22 0.0504
4	82 0.0526	77 0.0531	02 0.0523	71 0.0490	67 0.0539	1B 0.0535	08 0.0534	B7 0.0497	10 0.0536	65 0.0480	2A 0.0487	4A 0.0515	C0 0.0502	85 0.0550	E1 0.0523	7E 0.0496
5	0F 0.0522	E1 0.0522	79 0.0518	8A 0.0488	0C 0.0530	DB 0.0527	8A 0.0530	EE 0.0493	75 0.0531	33 0.0480	EC 0.0487	01 0.0510	F9 0.0487	73 0.0543	CD 0.0516	A2 0.0489

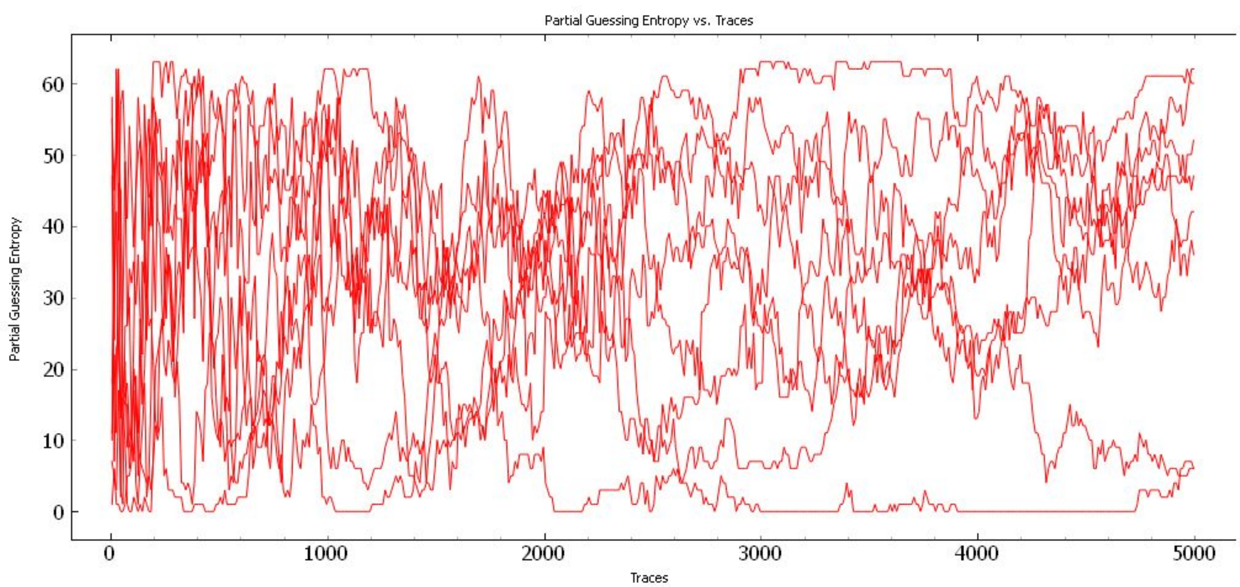


As we can see, the PGE remained pretty high at the end of 5000 traces. The average ranking of the correct guess was above 150. I consider it an unsuccessful attack.

Default DPA:

Results Table

	0	1	2	3	4	5	6	7
PGE	6	36	6	52	60	62	47	42
0	3D 0.0546	31 0.0466	03 0.0495	28 0.0552	03 0.0478	24 0.0599	35 0.0528	07 0.0561
1	25 0.0524	21 0.0438	2B 0.0493	31 0.0526	19 0.0447	3B 0.0572	14 0.0469	2D 0.0518
2	2C 0.0505	06 0.0438	3B 0.0482	07 0.0491	35 0.0444	25 0.0525	30 0.0466	1F 0.0477
3	33 0.0502	2E 0.0435	2D 0.0471	11 0.0481	29 0.0432	19 0.0523	2C 0.0451	21 0.0457
4	14 0.0498	22 0.0423	28 0.0470	0A 0.0474	3D 0.0430	07 0.0521	01 0.0448	1A 0.0431
5	10 0.0491	17 0.0422	2F 0.0468	26 0.0465	20 0.0426	06 0.0511	31 0.0446	2A 0.0428
6	22 0.0489	24 0.0420	30 0.0459	23 0.0434	2B 0.0415	32 0.0502	36 0.0441	33 0.0423
7	3A 0.0475	16 0.0419	3A 0.0458	1D 0.0424	04 0.0409	11 0.0497	37 0.0433	16 0.0419
8	02 0.0451	11 0.0418	09 0.0446	1E 0.0424	28 0.0404	3A 0.0484	0F 0.0430	09 0.0415



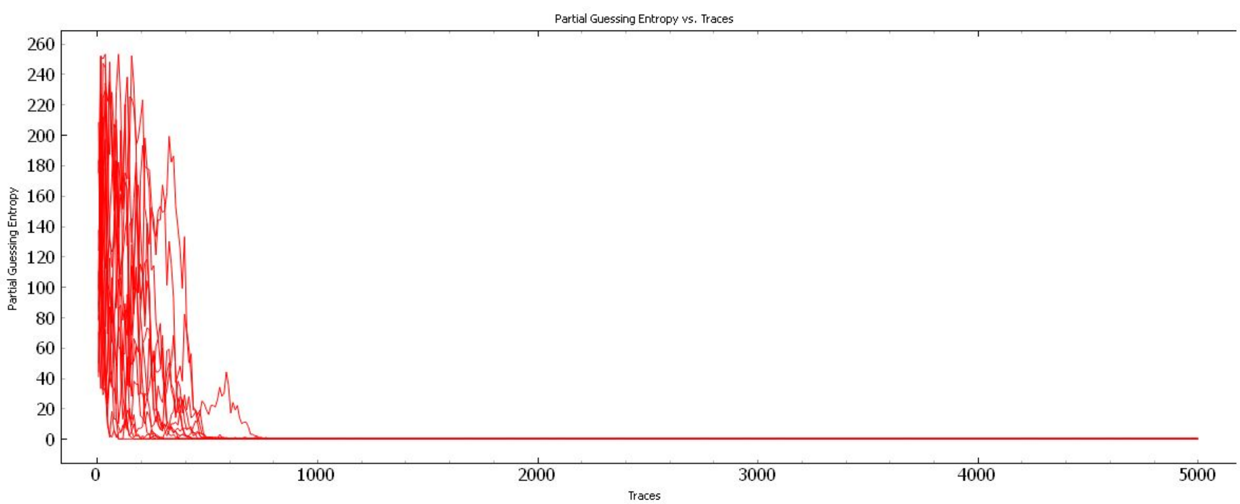
The DPA received a much higher PGE score. However, I didn't understand why it had only 8 subkeys. I know this attacking script was meant to be used in a DES, 56 + 8 bit, in a total of 8 byte. Though, I was running it against a 128 bit, 16 byte, 16 subkeys AES encryption. When

I tried to modify the target keys, it gave me an error of index out of bound. So, I let the script run as its default mode.

From the PGE plot, there was one subkey reached 0, but I didn't know how to verify the correctness for this attack.

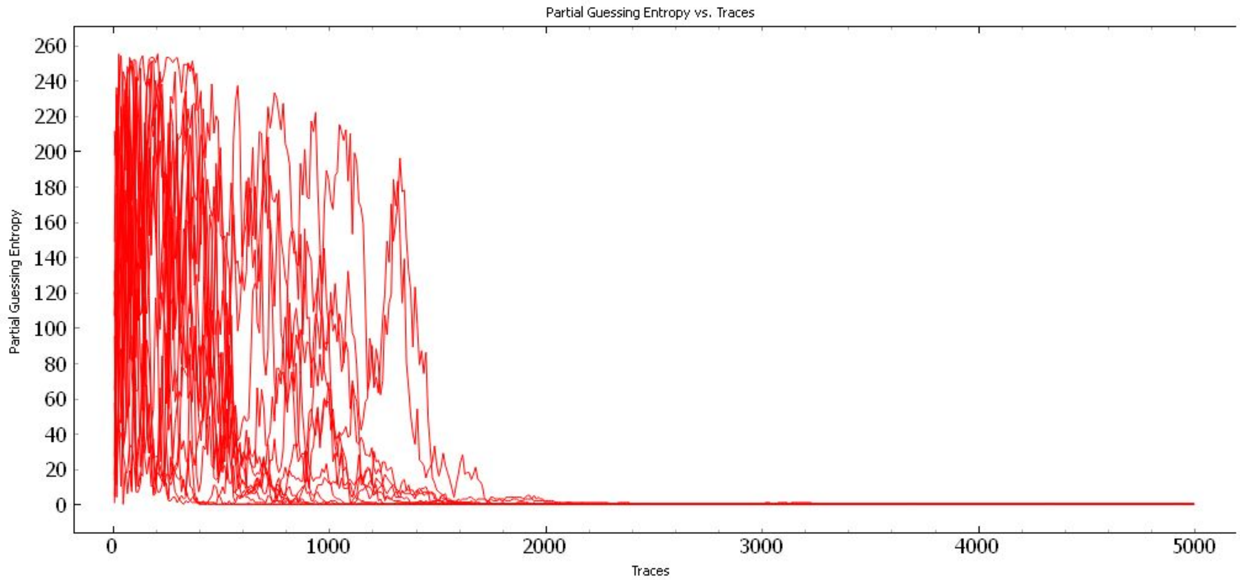
CPA with Last Round State key leakage:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	D0 0.2130	14 0.1947	F9 0.1803	A8 0.1837	C9 0.2160	EE 0.2286	25 0.2236	89 0.2098	E1 0.1718	3F 0.2212	0C 0.1818	C8 0.2066	B6 0.1789	63 0.2102	0C 0.1993	A6 0.1776
1	7C 0.0689	A8 0.0587	3E 0.0729	B9 0.0770	4C 0.0704	E7 0.0598	F7 0.0574	63 0.0593	B1 0.0730	13 0.0600	E0 0.0690	D5 0.0544	33 0.0648	BC 0.0612	DF 0.0635	89 0.0607
2	75 0.0625	A2 0.0536	A0 0.0580	09 0.0572	0D 0.0573	23 0.0579	7C 0.0524	03 0.0562	02 0.0715	F5 0.0594	FD 0.0561	CF 0.0541	AE 0.0604	7F 0.0567	E2 0.0569	E2 0.0596
3	57 0.0588	5A 0.0528	DD 0.0556	91 0.0538	07 0.0555	8F 0.0544	05 0.0515	DB 0.0556	33 0.0599	24 0.0547	9A 0.0560	0E 0.0535	43 0.0541	22 0.0540	E0 0.0567	D0 0.0582
4	85 0.0578	E0 0.0521	5E 0.0522	46 0.0517	FE 0.0532	8E 0.0513	8E 0.0508	5B 0.0503	18 0.0580	C3 0.0543	2D 0.0558	32 0.0528	CF 0.0531	F6 0.0525	8F 0.0543	9E 0.0539
5	01 0.0572	83 0.0499	52 0.0514	65 0.0509	6E 0.0529	55 0.0508	4D 0.0502	FE 0.0488	DD 0.0575	E7 0.0540	F0 0.0545	09 0.0504	A2 0.0526	E4 0.0519	81 0.0529	3E 0.0526
6	D2 0.0560	F1 0.0486	26 0.0513	96 0.0505	B6 0.0527	8D 0.0501	E2 0.0485	75 0.0463	F5 0.0570	35 0.0530	73 0.0544	08 0.0499	0E 0.0524	35 0.0518	68 0.0527	1E 0.0517



When I targeted the last round state key using CPA attack, the result table showed all subkeys' PGE were equal to zero and converged to zero pretty quickly. This was indicating that the algorithm predicted all the correct subkeys for the last round. Since we can infer all round keys from the last round key, this is a successful attack.

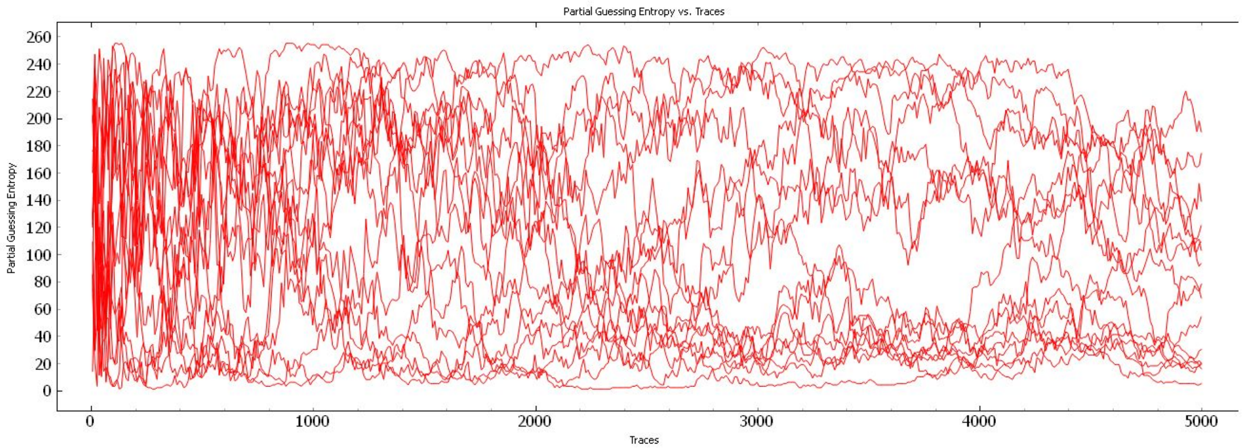
CPA with LRS and noise:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	D0 0.1195	14 0.1197	F9 0.1103	A8 0.1162	C9 0.1267	EE 0.1561	25 0.1198	89 0.1325	E1 0.0907	3F 0.1163	0C 0.0933	C8 0.1274	B6 0.0840	63 0.1317	0C 0.1185	A6 0.1008
1	AB 0.0566	97 0.0574	AE 0.0644	2A 0.0592	84 0.0614	50 0.0589	D8 0.0637	B2 0.0643	46 0.0682	82 0.0736	06 0.0623	00 0.0590	27 0.0597	5B 0.0586	00 0.0749	07 0.0628
2	7C 0.0560	76 0.0564	01 0.0641	1F 0.0590	FE 0.0594	A0 0.0576	DA 0.0618	D4 0.0602	E3 0.0615	F8 0.0572	27 0.0594	96 0.0569	E8 0.0591	99 0.0577	11 0.0658	1F 0.0604
3	5D 0.0558	04 0.0560	0B 0.0591	10 0.0588	F7 0.0568	28 0.0571	F8 0.0583	3F 0.0581	21 0.0593	36 0.0570	2E 0.0569	5D 0.0558	AD 0.0571	C3 0.0562	6D 0.0586	F1 0.0590
4	A5 0.0550	DC 0.0549	62 0.0573	A5 0.0550	7A 0.0556	94 0.0564	79 0.0582	77 0.0581	B9 0.0585	4C 0.0565	03 0.0567	0E 0.0544	8E 0.0554	51 0.0560	F6 0.0582	A8 0.0572
5	C5 0.0550	02 0.0547	A7 0.0566	C4 0.0545	4E 0.0548	D6 0.0564	CE 0.0579	CB 0.0580	CD 0.0575	DF 0.0563	DC 0.0556	41 0.0541	51 0.0542	0F 0.0552	CB 0.0576	1E 0.0571
6	5B 0.0550	2F 0.0545	3E 0.0562	60 0.0542	A8 0.0546	EA 0.0551	66 0.0570	2B 0.0578	EE 0.0551	D5 0.0561	B0 0.0552	44 0.0539	35 0.0535	3C 0.0540	8F 0.0574	D6 0.0560

I added 1% noise to this attack. The result was similar to the previous attack. However, the converge speed was slower. All subkeys converged at the 3240th trace compared to the previous 779th trace. From my data science knowledge, adding the noise prevented the overfit but also increased the training difficulty here. To summarize, it's a successful and slow attack since it got all the correct subkeys at the end. The noise adding technique might be helpful for more complicated situations in the future.

CPA with LRS and jitter:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	**14**	15
PGE	139	30	54	68	20	103	20	109	5	93	78	16	174	121	19	190
0	25 0.0585	4A 0.0535	00 0.0557	2B 0.0522	2E 0.0667	62 0.0518	F2 0.0524	C6 0.0512	30 0.0523	24 0.0580	FF 0.0473	18 0.0597	C0 0.0613	D2 0.0538	A2 0.0567	08 0.0569
1	95 0.0557	EB 0.0481	3B 0.0537	C4 0.0514	69 0.0562	4D 0.0511	57 0.0499	A1 0.0494	0C 0.0520	CF 0.0567	55 0.0466	E0 0.0596	F0 0.0513	14 0.0505	46 0.0525	40 0.0525
2	85 0.0518	AD 0.0481	42 0.0535	42 0.0484	51 0.0506	9D 0.0493	CE 0.0498	20 0.0480	E0 0.0460	FE 0.0540	72 0.0460	2A 0.0571	42 0.0510	15 0.0476	4D 0.0497	B8 0.0502
3	48 0.0507	05 0.0480	B5 0.0519	C1 0.0474	FB 0.0481	6C 0.0476	26 0.0456	4D 0.0462	A3 0.0456	5A 0.0506	3D 0.0441	10 0.0519	08 0.0499	5F 0.0468	47 0.0459	E1 0.0494
4	2F 0.0497	1B 0.0476	03 0.0477	E7 0.0453	13 0.0477	73 0.0462	27 0.0454	23 0.0462	10 0.0450	FA 0.0504	9D 0.0427	E4 0.0511	E3 0.0476	6F 0.0458	F4 0.0448	E5 0.0490
5	2A 0.0484	75 0.0471	17 0.0473	9D 0.0452	17 0.0466	59 0.0448	69 0.0452	A9 0.0452	E1 0.0447	15 0.0487	C8 0.0427	7E 0.0506	1D 0.0473	1A 0.0453	F6 0.0434	43 0.0478

I set the jitter parameter to 3. The attack didn't make much progress due to the introduced jitter. The highest PGE score was 5 at the 9th subkey. It was slightly better than the plain CPA attack from the overall PGE score. I consider it a failed attack since it could not produce the correct subkeys at the end.

Overall, I think it's a fun and challenging lab to work on! It would be even better if we can use the latest software version. The CWA is not very friendly to Mac users due to the incompatibility of some outdated dependencies.

References:

1. O'Flynn C, Chen Z D. Side channel power analysis of an AES-256 bootloader[C]//2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE). IEEE, 2015: 750-755.