Assignment Homework 1

Due date: Jan 27, 2023 (AoE) = Anywhere on Earth

Please complete this assignment (100 pts total) and submit your report/program code on Canvas (all files compressed in one `.zip` without the `.csv` file)

1. Please consider the following AES inputs and compute the output after the first SubBytes operation. Only use a pocket calculator and the table given in Figure 1. This is intended to make sure you fully understood the concept. (10 pts)

   ```
   Plaintext = 00 00 00 00 00 00 C1 A5 51 F1 ED C0 FF EE B4 BE
   Key       =  00 00 01 02 03 04 DE CA F0 C0 FF EE 00 00 00 00
   ```

   Note: FIPS 197 Appendix B is always a great reference for this kind of task[*].

2. Your coworker needs to implement the AES MixColumns operation in software. He found some code on `stackexchange.com`. You remember the best practice to review (and test) code before using it in production. Therefore, you offer to review the code. (10 pts)

   a) What is the type of problem here? (2 pts)

   b) Identify the function(s) with undesired behavior. (2 pts)

   c) Suggest a code fragment that solves the problem under idealized assumptions. (4 pts)

   d) Is your solution processor independent? Please provide appropriate reasoning. (2 pts)

Listing 1: Example Multiply by 2 and 3 for AES MixColumns

```c
unsigned char MixColumns_Mult_by2(unsigned char Input) {
  unsigned char Output = Input << 1;
  if (Input & 0x80)
    Output ^= 0x1b;

  return (Output);
}

unsigned char MixColumns_Mult_by3(unsigned char Input) {
  unsigned char Output = MixColumns_Mult_by2(Input) ^ Input;

  return (Output);
}
```

[*] `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf`

3. The file `timing_noisy.csv` contains 1 million timing samples of the AES with random plaintexts and a fixed key. This is representative for an actual attack scenario, as the attacker controls the input but not the key. The file format is `.csv` (Comma-Separated Values) and you should be able to read it in your preferred environment or convert it to any format that you need. Its first 16 columns contain the plaintext bytes in ASCII (from 0 to 255) and the last column shows the timing value. Your duty is to perform a timing side-channel analysis.

   a) Recover the key. Please assume that the Most Significant Bit (MSB) is a good hypothesis. Provide figures to support that you found the correct key. (40 pts)

   b) Check how many samples you need for each key byte. This can be done, e.g., in 1 000 step increments and does not need to be an exact number. (20 pts)

   c) Optimize your code such that the overall attack time (without incremental steps) is well below **3** minutes. Please report your execution time *in addition* to your processor and memory specification. Include a note which operating system you used. If you needed to optimize your code to improve runtime, *briefly include a note how you optimized.* (20 pts)

   Please write your program in one of the following languages/environments: Python/Jupyter (*strongly recommended*), C/C++, Java, Rust, Matlab/Octave. Choose wisely, as you may be able to reuse code in follow-up assignments. Your `.zip` file should contain your code, instructions how to make it run (if needed), the figures, and the number of required samples.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 1: AES Sbox as Look Up Table (LUT). Note that input byte `0x00` returns the value `0x63` and the input byte `0x01` gives `0x7c`. Do not separate the lookup based on nibbles and use the byte directly.