The Dark Arts of Hacking the “Unhackable”

Some random guy yapping about side channels and more useless junk you never knew you didn’t not need

Batman’s Kitchen
Feb 19th, 2025

Hacking is pretty simple in theory...

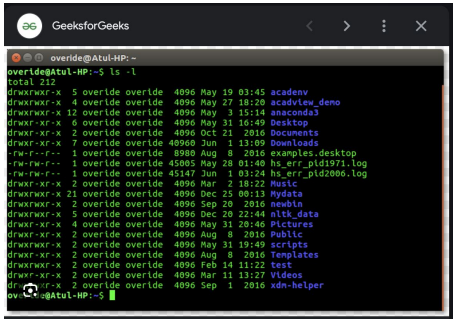
Some guy writes shitty code → You find a bug in it → ??? → Profit

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  char true_pass[8] = "********";
  char user_pass[8];
  gets(user_pass);

  int match = memcmp(user_pass, true_pass, 8);

  if (match != 0) {
    puts("incorrect password\n");
    exit(1);
  }

  puts("you win!");
  exec("/bin/sh");
}
```

Bugs

Never use **gets**(). Because it is impossible to tell without knowing the data in advance how many characters **gets**() will read, and because **gets**() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use **fgets**() instead.

^ from: https://linux.die.net/man/3/gets



Hacking is pretty simple in theory...

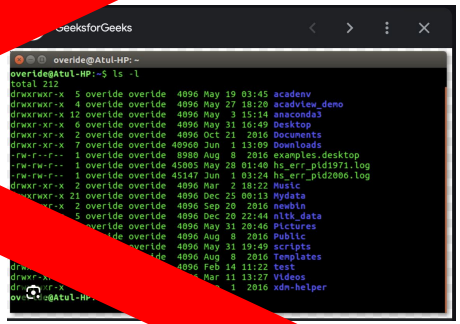
Some guy writes ~~shitty~~ code → d a bug in it → ??? → P

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  char true_pass[8] = "********";
  char user_pass[8];
  fgets(user_pass, 8, stdin);

  int match = memcmp(user_pass, true_pass, 8);

  if (match != 0) {
    puts("incorrect password\n");
    exit(1);
  }

  puts("you win!");
  exec("/bin/sh");
}
```

Bugs

Never use **gets**(). Because it is impossible to tell without knowing the data in advance how many characters **gets**() will read, and because **gets**() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Us **fgets**() instead.

^ from: htt dle.net/man /3 ts



What is a hacker?

- Someone who exploits the gaps between assumptions and designs
- Not just computer systems!
  - Social systems (people and organizations)
  - Physical security systems
  - Public information systems

- Andrew “Middy”, Circa Sept. 2024

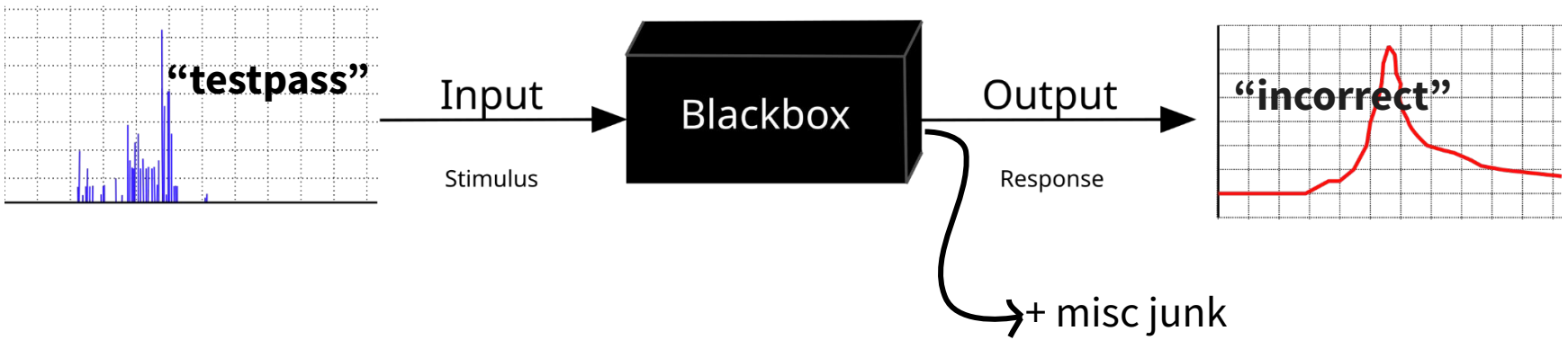
Challenging Computer Assumptions

The language computers speak is a superset of the language we think computers speak

Challenging Computer Assumptions

The language computers speak is a superset of the language we think computers speak

??? ^ wtf does this mean



You Retweeted

JavaScript Daily
@JavaScriptDaily

Presenting the "sleep sort", our favorite O(wtf) sorting algorithm. (C'mon, it's the weekend..) 😂

```
// sleep sort

numbers = [8, 42, 38, 111, 2, 39, 1];

numbers.forEach(num => {
  setTimeout(() => { console.log(num) }, num);
});
```

3:03 PM · Apr 23, 2017 · Buffer

1.2K Retweets **1.7K** Likes

Woo! Let's do an activity

Case Study #1: Programs aren’t instantaneous

poopoopeepee

“Secure” programs aren’t always safe

- The threat model programmers use != the real world threat model
- We’re slowly getting there! (... sort of)
- https://www.chosenplaintext.ca/articles/beginners-guide-constant-time-cryptography.html
- Are these attacks practical??

Breaking Bad: How
Compilers Break Constant-Time Implementations

Moritz Schneider
ETH Zurich

Daniele Lain
ETH Zurich

Ivan Puddu
ETH Zurich

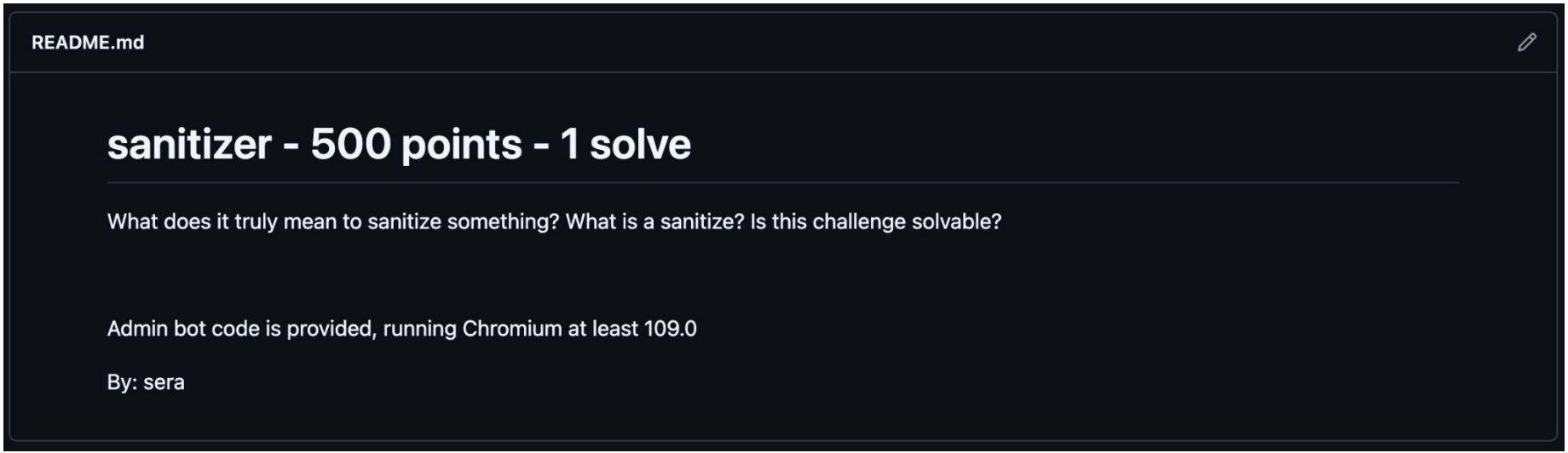
Nicolas Dutly
ETH Zurich

Srdjan Čapkun
ETH Zurich

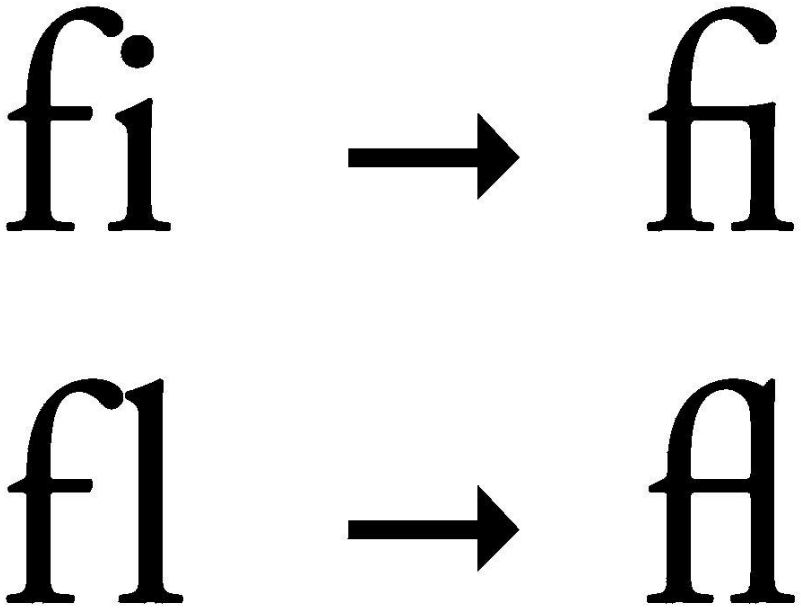
Case Study #2: Web exploitation on secure pages

xslaeks

Web exploitation without javascript??



Wtf is a ligature?



Wtf is a ligature?

f -> **f**

i -> **i**

fi -> 

The  ery  nch

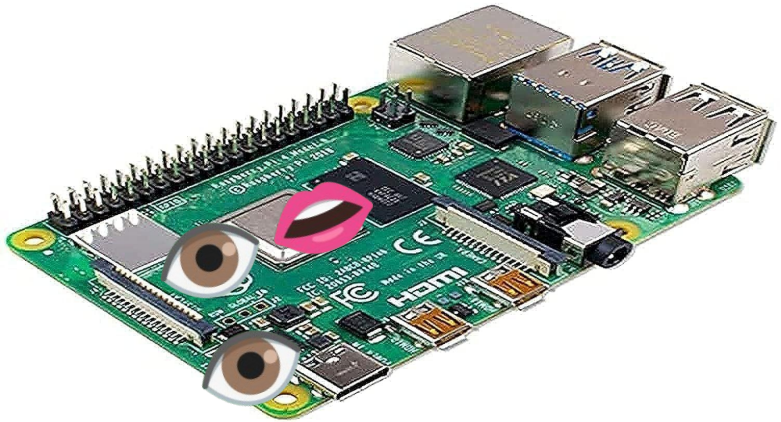
swiftly flicked its  ne feathers

Extra Credit: Breaking Government-Approved Crypto

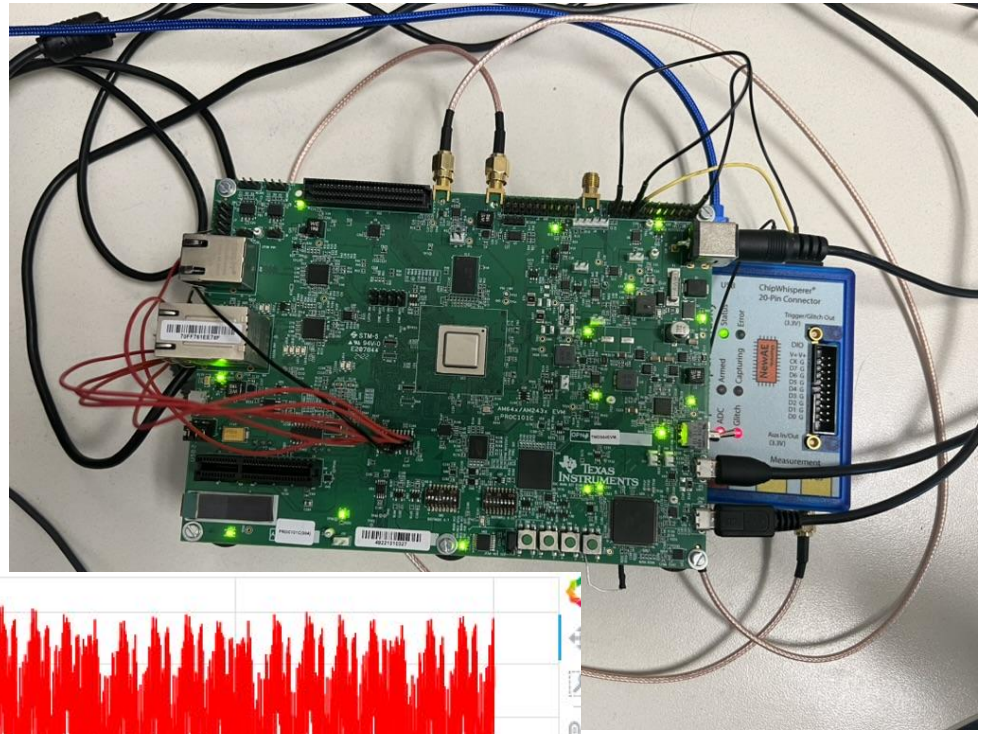
(it’s brute forcing but cooler)

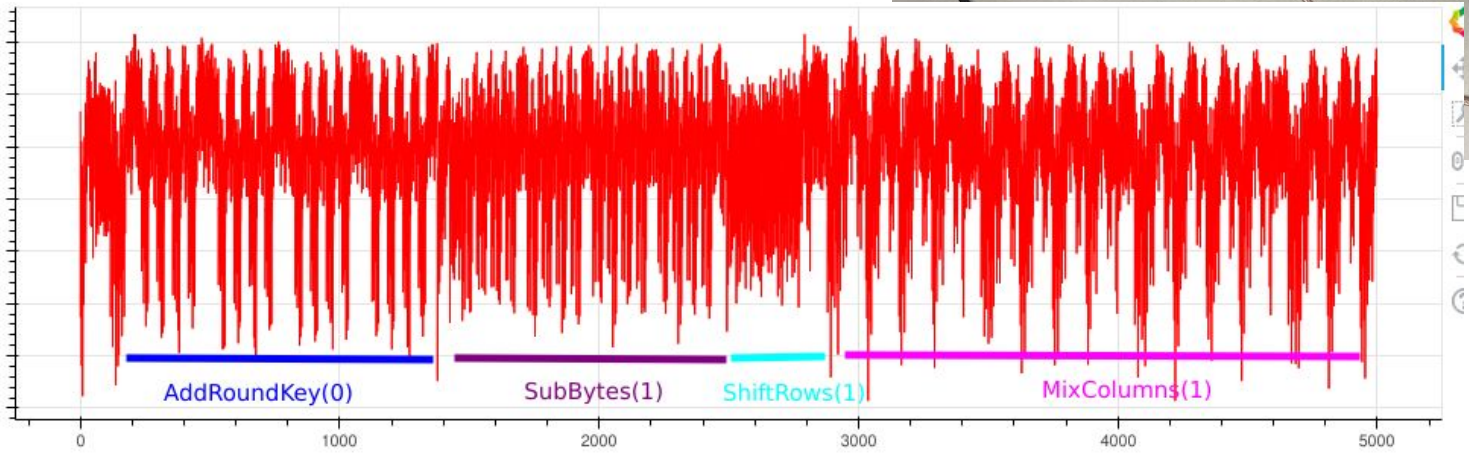
Breaking Government-Approved Crypto

- You have some device, it has some firmware on it
- You wanna figure out what it’s doing, let’s look at that firmware!
- The firmware is encrypted with AES >:(



Let's Power Analysis it!





How does it work

Simple: magic

Complex: ML magic

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{E[(X-\mu_X)^2]E[(Y-\mu_Y)^2]}}$$

$$r_{i,j} = \frac{\sum_{d=1}^{D}\left[\left(h_{d,i}-\overline{h_i}\right)\left(t_{d,j}-\overline{t_j}\right)\right]}{\sqrt{\sum_{d=1}^{D}\left(h_{d,i}-\overline{h_i}\right)^2 \sum_{d=1}^{D}\left(t_{d,j}-\overline{t_j}\right)^2}}$$

$$r_{i,j} = \frac{D\sum_{d=1}^{D} h_{d,i}t_{d,j} - \sum_{d=1}^{D} h_{d,i}\sum_{d=1}^{D} t_{d,j}}{\sqrt{\left(\left(\sum_{d=1}^{D} h_{d,i}\right)^2 - D\sum_{d=1}^{D} h_{d,i}^2\right)\left(\left(\sum_{d=1}^{D} t_{d,j}\right)^2 - D\sum_{d=1}^{D} t_{d,j}^2\right)}}$$

Ok but for real:

- Computers take power to do stuff (duh)
- They take more power to do different operations
- 34 + 12 -> 10pW
- 578284 + 7278894 -> 12pW

Hamming Weight Power Model

If a number has more “1”s in it (in binary form), your CPU draws more power when using it

255 = 0b11111111 -> draws “8” power

38 = 0b00100110 -> draws “3” power

0 = 0b00000000 -> draws “0” power

The abridged version:

AES Step 1:

temp = sbox[key[0] ^ ct[0]]

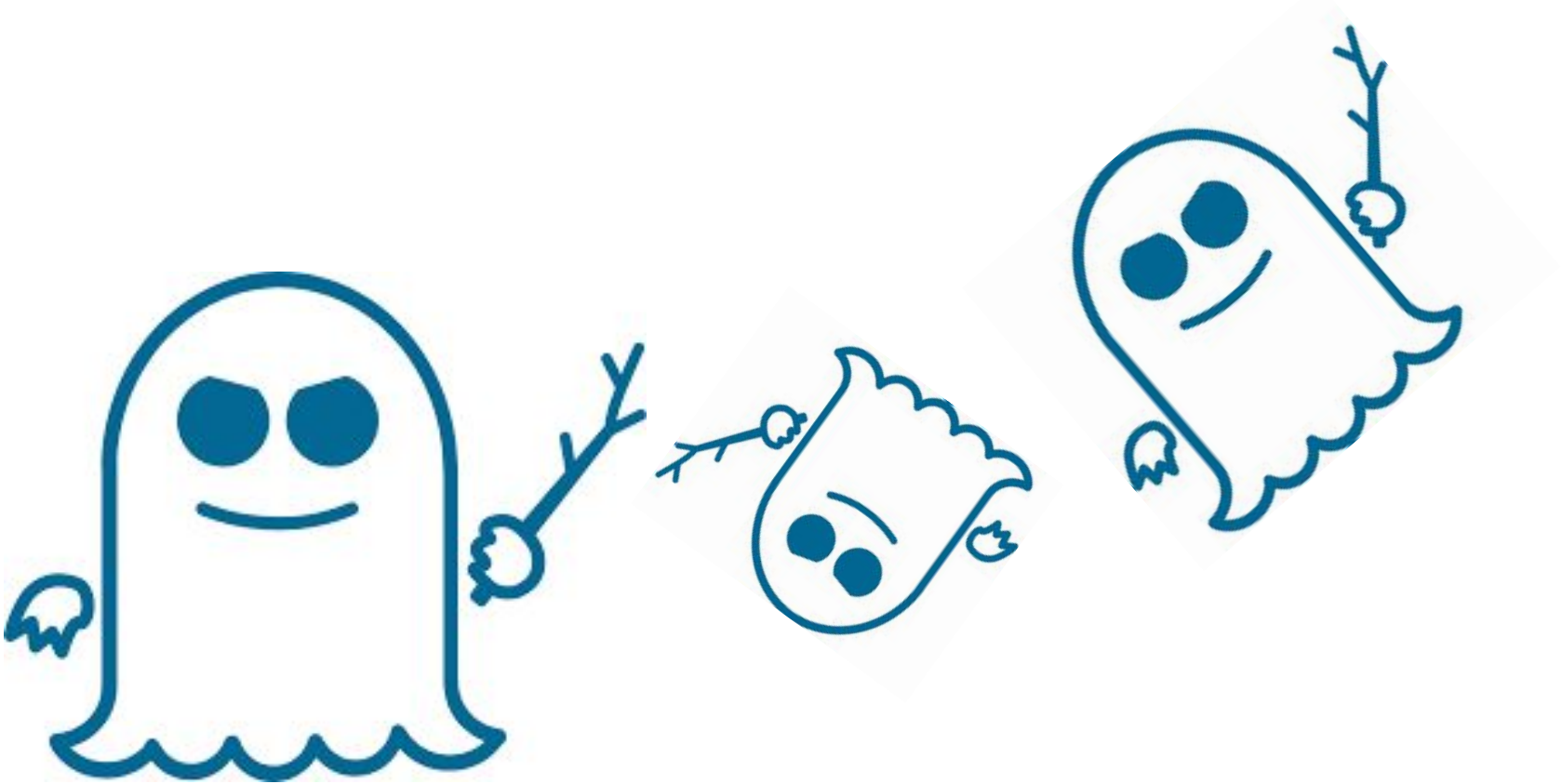
^ this is all we care about!

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The abridged version:

1. Make a guess as to what the key byte is
2. Simulate what the “expected” power draw is for that key byte
3. Find how different it is from the actual power draw
4. Which guess has the closest similarity (correlation) to the actual power draw? -> This is probably what the key byte is!
5. Repeat for next key byte

Extra Credit: Speculative Execution ~~for dummies~~



Click to add title

- nah we’re not getting this far not gonna make slides for this lol