University of Pittsburgh

Applied Cryptography and Network Security
CS 1653

Summer 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee's CS1653 slides.)

Announcements

- Homework 2 due this Friday @ 11:59 pm
  - 3 attempts
- Phase 1 of Project due next Tuesday @ 11:59 pm

Example: Advanced Encryption Standard (AES)

Develop a (brief) history of modern cryptography

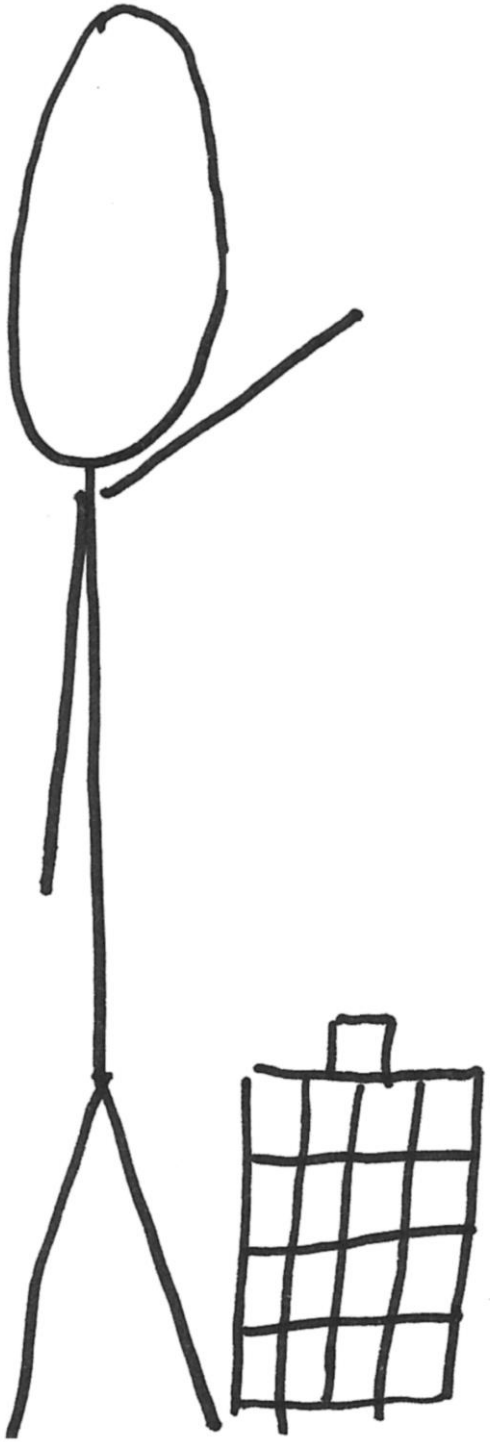
Learn about the AES standardization process

Understand AES:

- ✓ High level: Appreciate how AES utilizes confusion/diffusion
- **Low level: Gain exposure to the mathematics behind AES**

Appreciate that implementing something as complex as AES is a very non-trivial task

A Stick Figure Guide to the Advanced Encryption Standard (AES)



http://www.moserware.com/

In the early 70s, Lucifer was created at IBM

Lucifer became DES

- Slightly modified
  - Shorter key and new, (not so) suspicious S-boxes
- standardized by NSA/NIST*

Over the years, DES suffered many breaks and brute forces

- 1977 Diffie/Hellman propose $20M machine to find key in a day
- 1993 Wiener proposed $1M machine, 7 hours
- 1997 RSA Security held contest, distributed DESCHALL broke DES
- 1998 EFF built Deep Crack, $250,000, about 2 days

1998, 3DES: **More DES!**

- Encrypt three times with DES with different keys
- Oh man, this is so slow…

Even by the late 80s, replacements started to appear

Summary of last lecture …

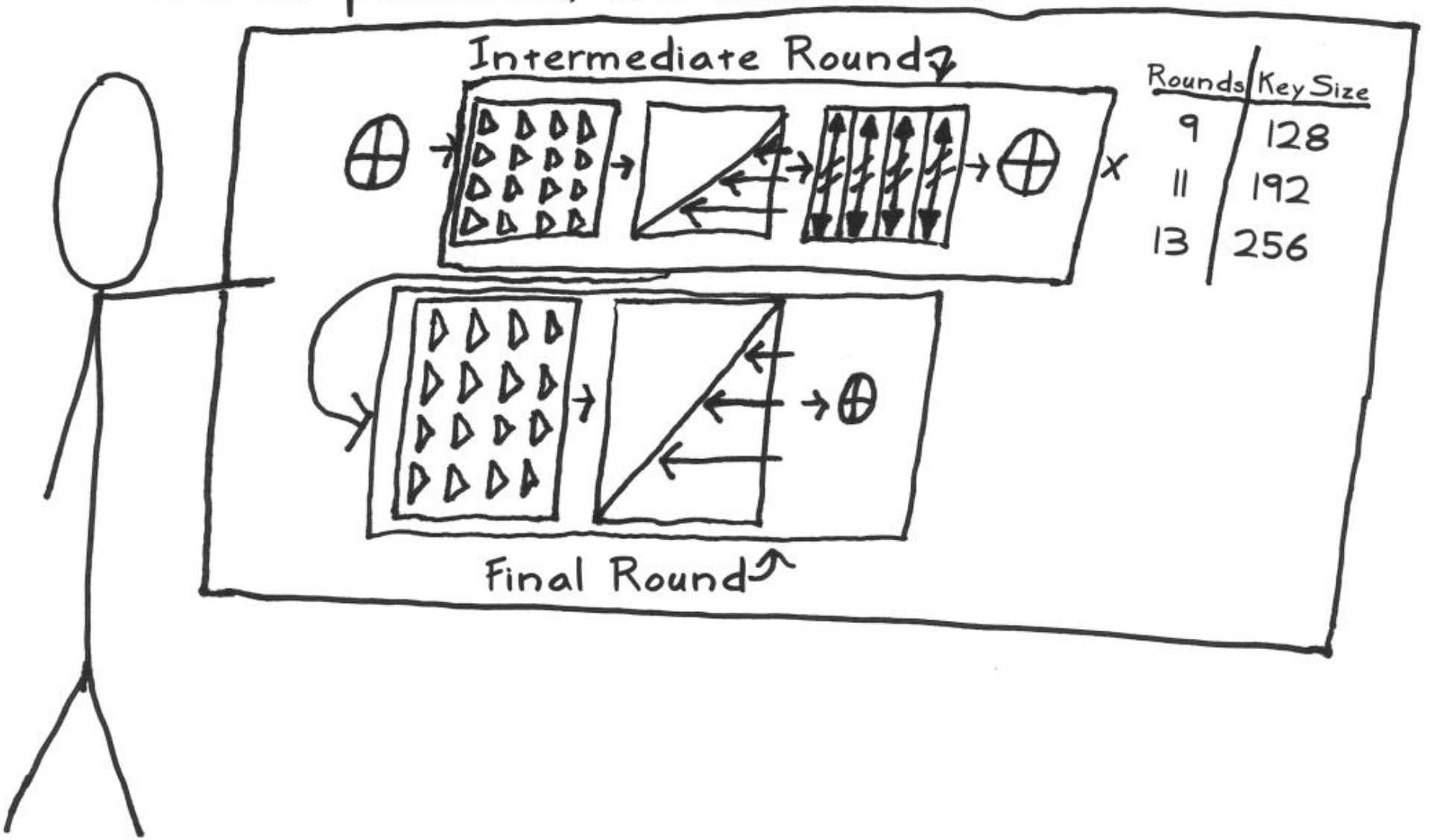
In theory, you now understand how 128-bit AES works

High level: Apply **confusion** and **diffusion** to 128-bit blocks

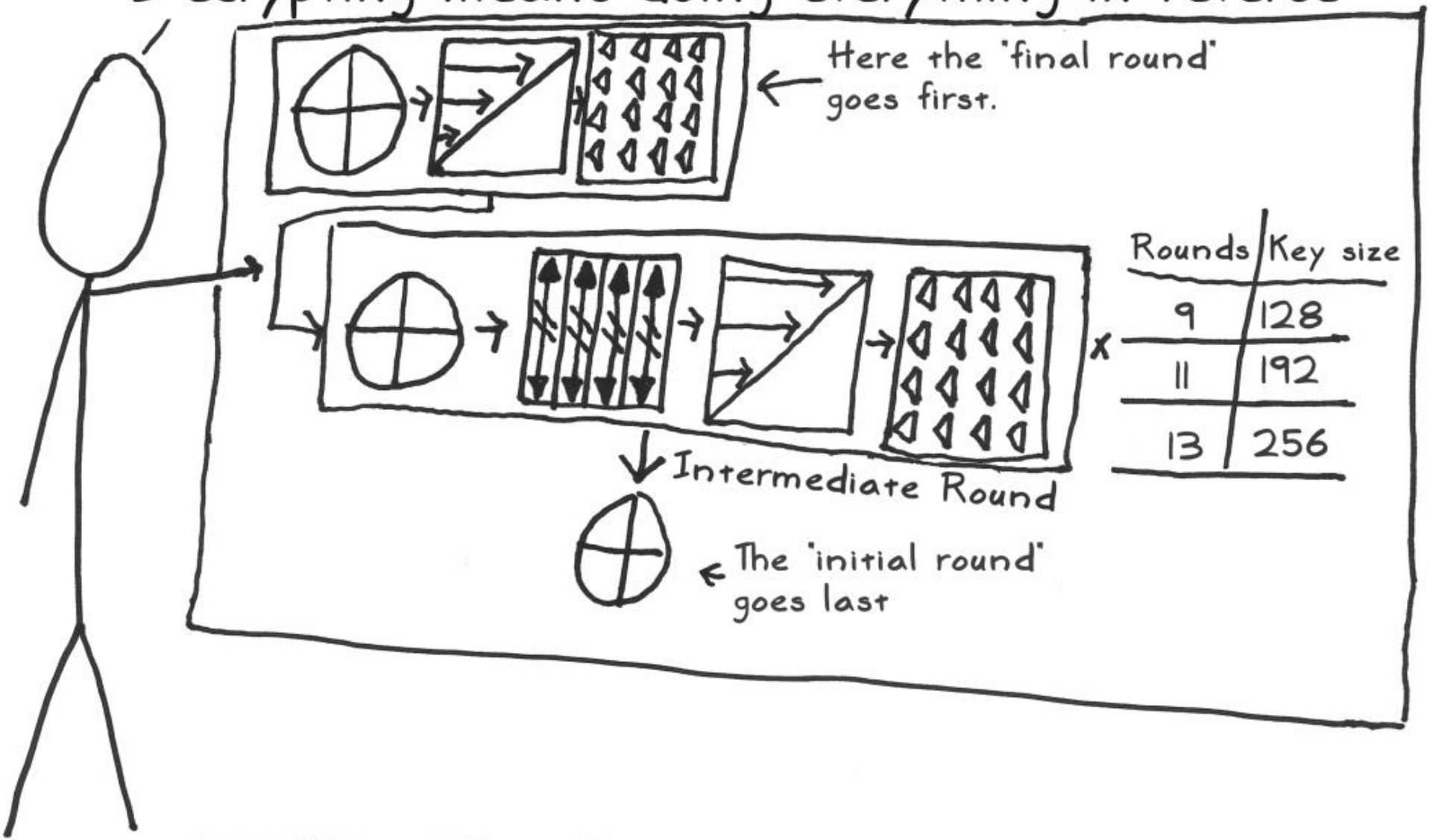
Medium level:

- Key expansion to get 10 round keys
- SubBytes via some crazy S-Box
- ShiftRows to spread around the bytes
- MixColumns to further spread around the bytes
- XOR with round key, and repeat

So in pictures, we have this:



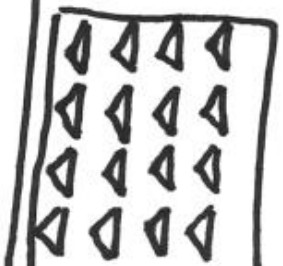
Decrypting means doing everything in reverse



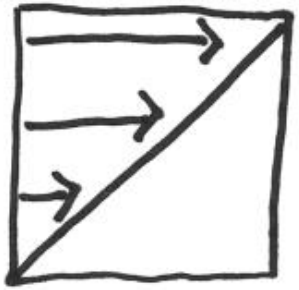
Rounds	Key size
9	128
11	192
13	256

Add Round Key Inverse

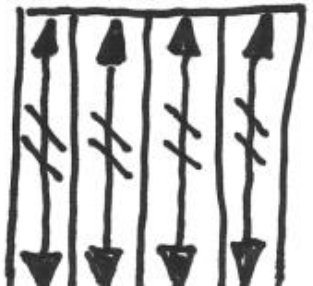
Inverse Substitute Bytes

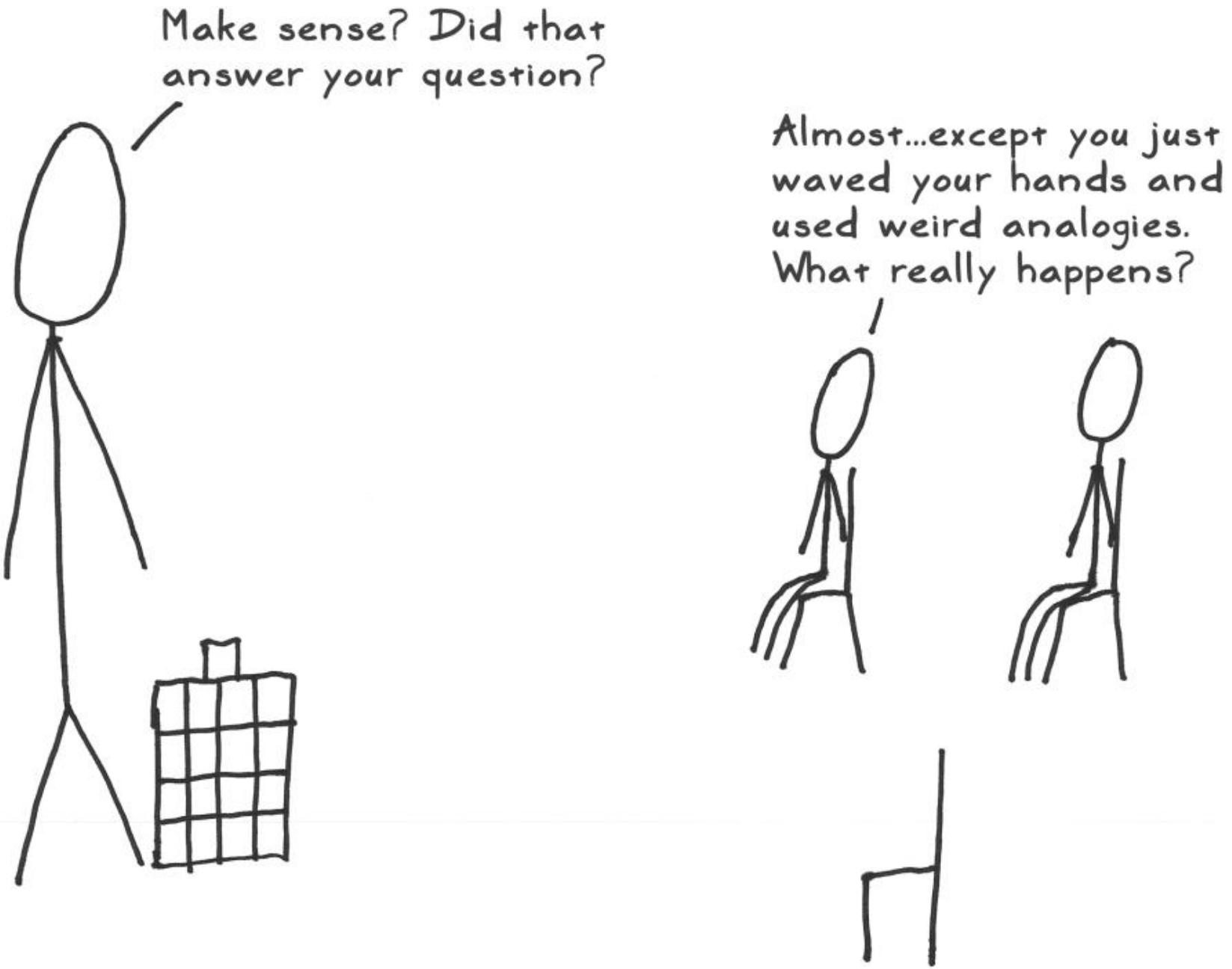


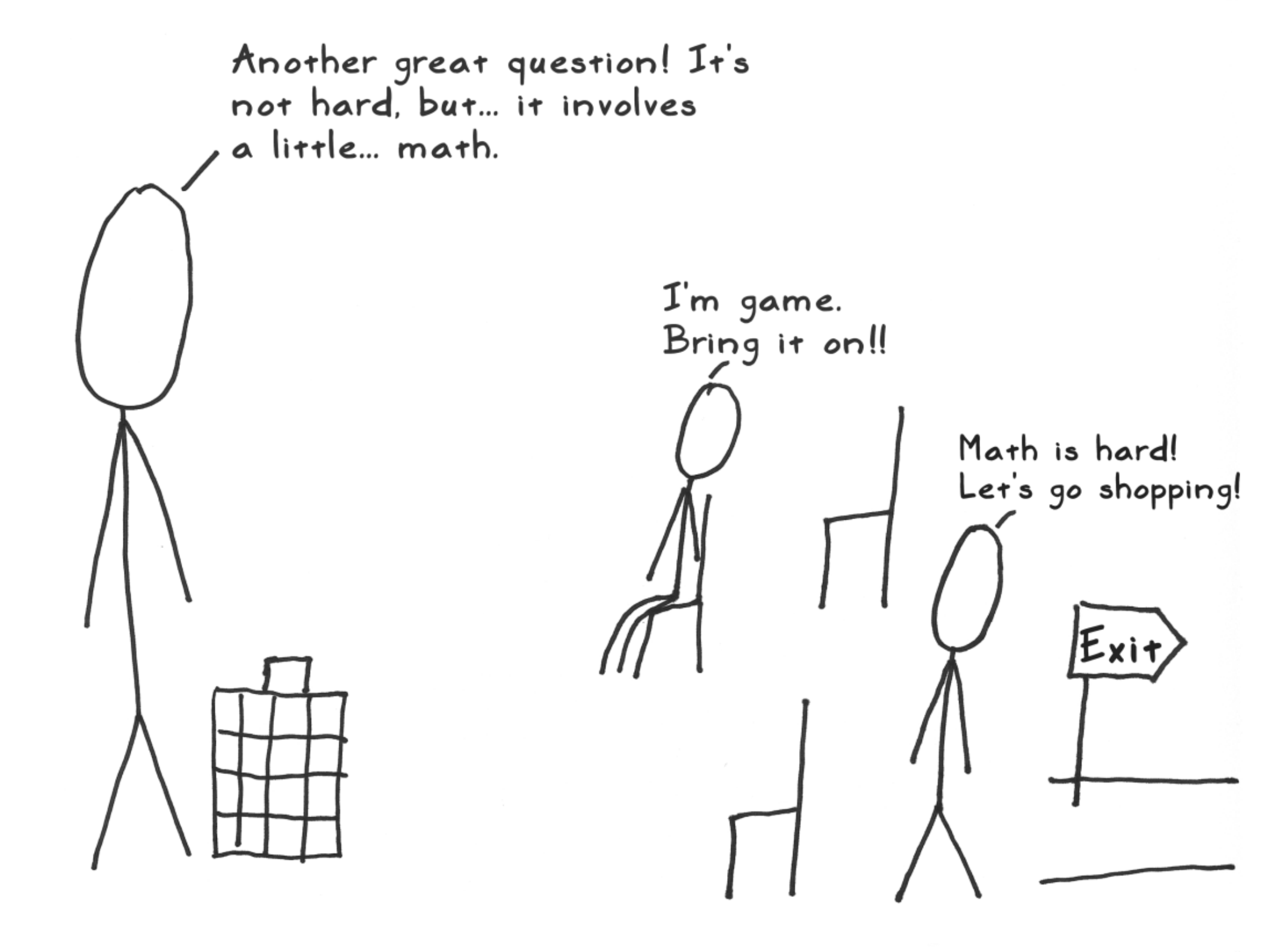
Inverse Shift Rows



Inverse Mix Columns

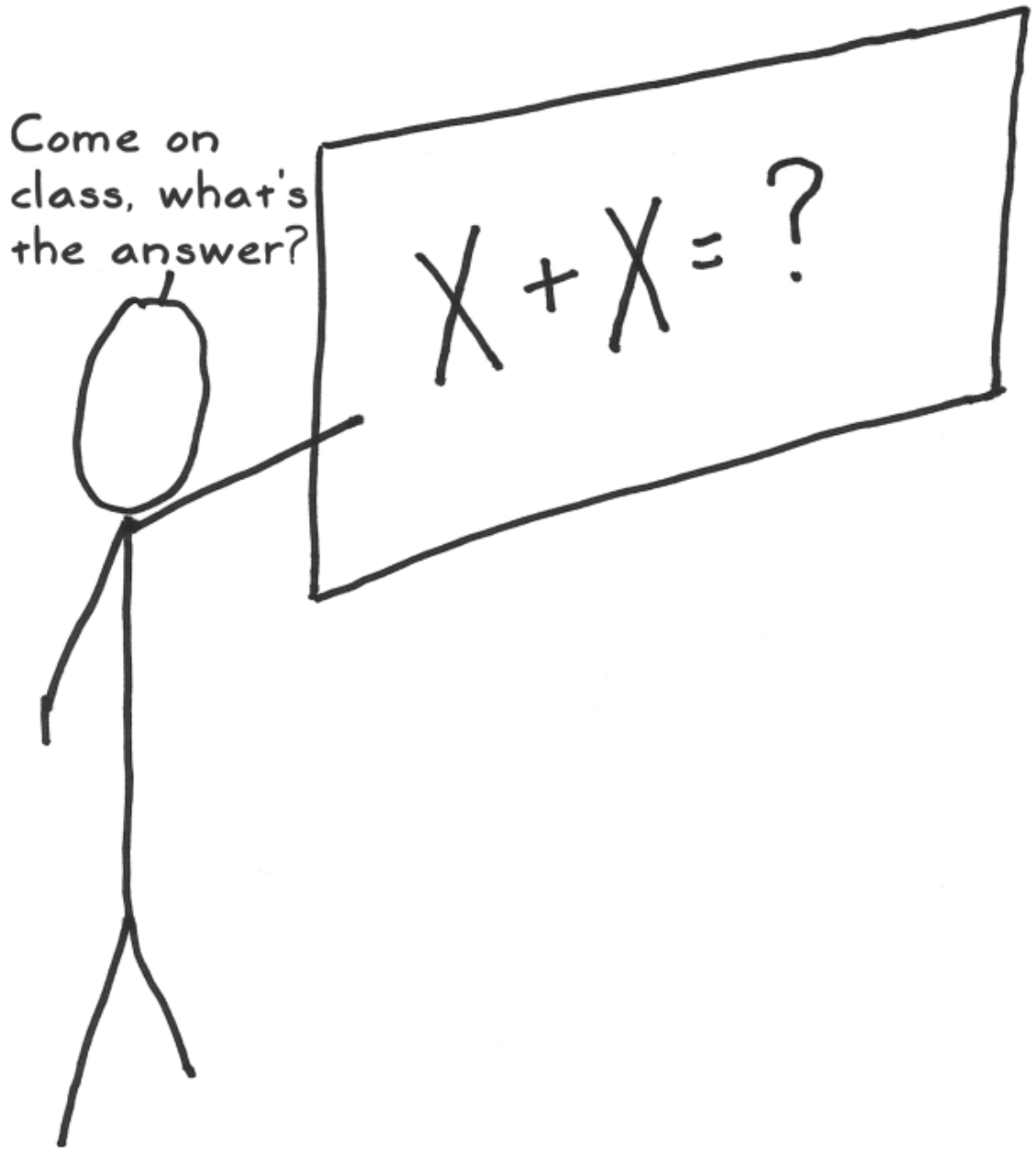


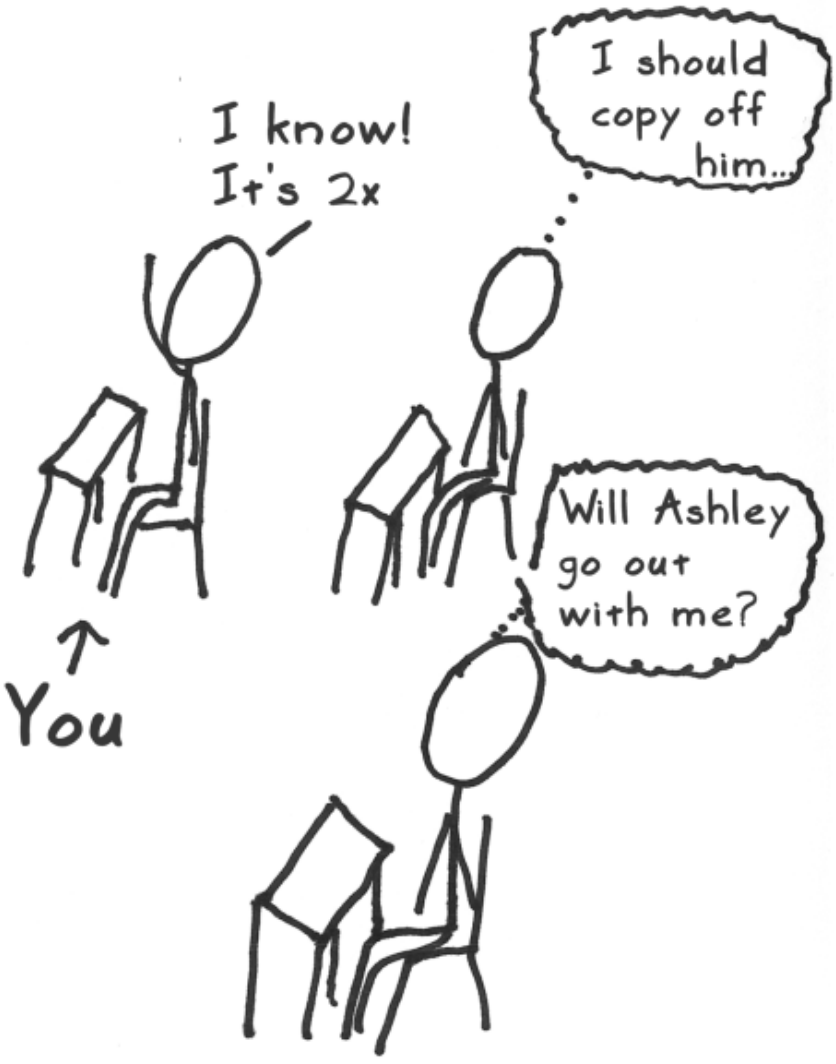

Make sense? Did that answer your question?
Almost...except you just waved your hands and used weird analogies. What really happens?


Another great question! It's not hard, but... it involves a little... math.
I'm game. Bring it on!!
Math is hard! Let's go shopping!
Exit

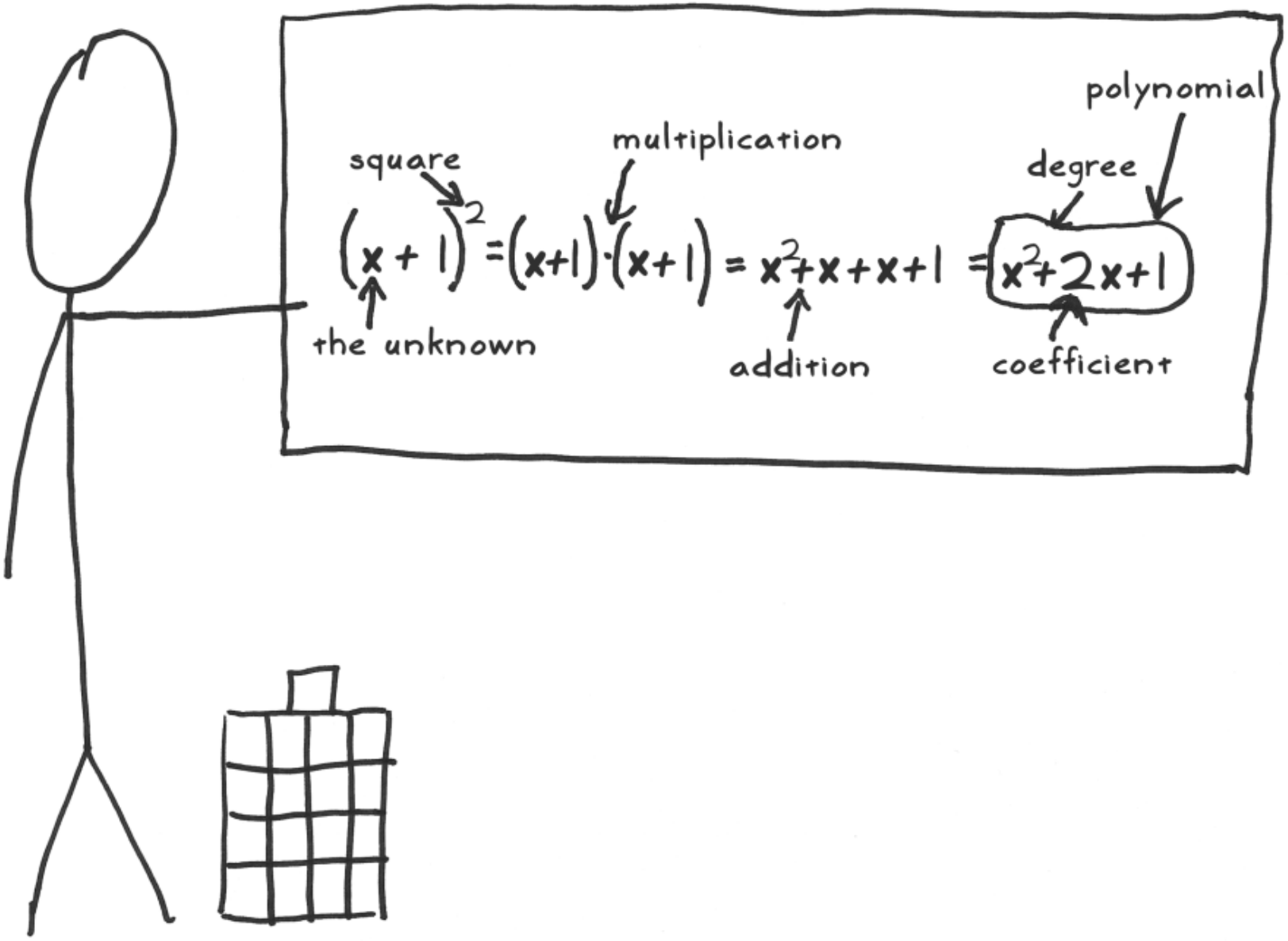
Act 4: Math!

Let's go back to your algebra class...

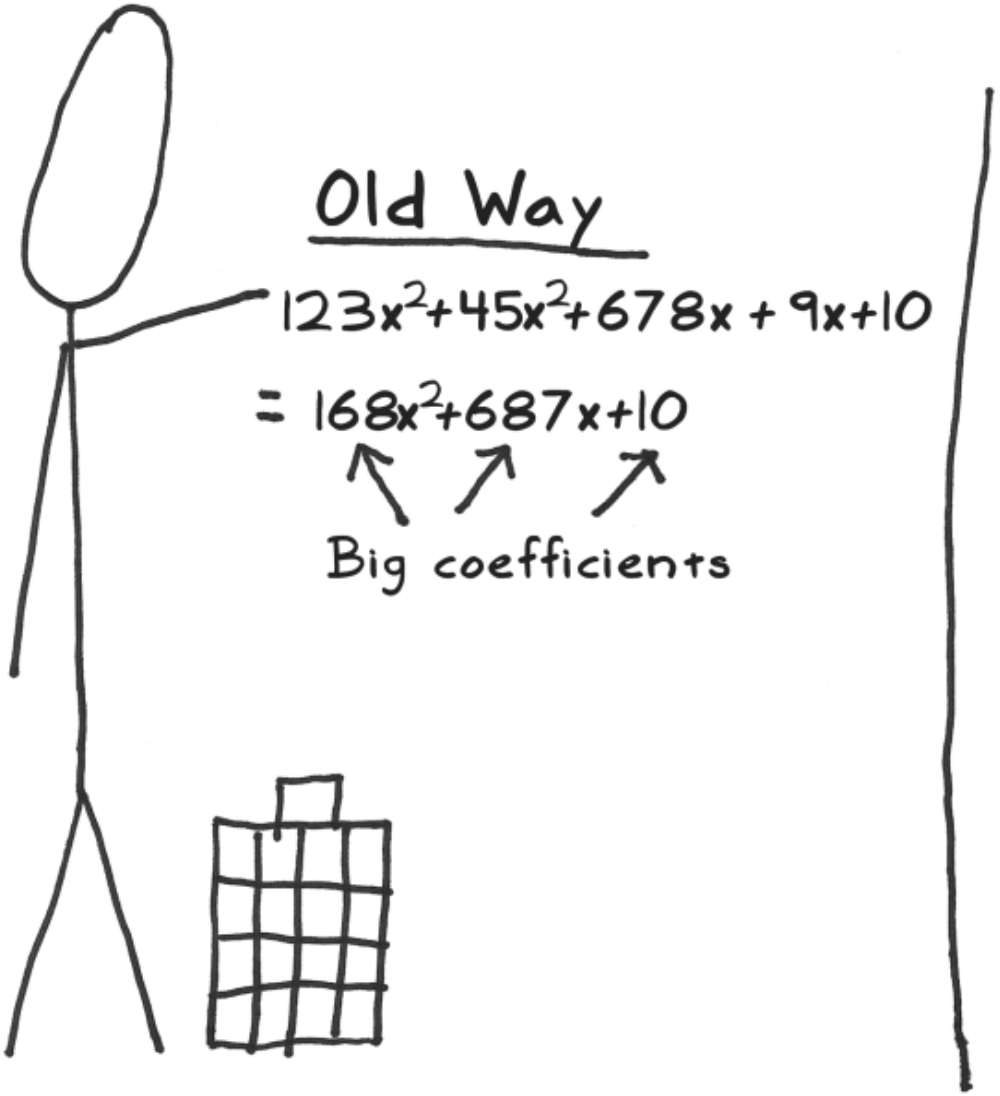




Reviewing the Basics...



We'll change things slightly. In the old way, coefficients could get as big as we wanted. In the new way, they can only be 0 or 1:



Old Way

$123x^2+45x^2+678x+9x+10$

$= 168x^2+687x+10$

Big coefficients

New Way

$x^2 \oplus x^2 \oplus x^2 \oplus x \oplus x \oplus 1$

$= x^2 \oplus 1$

The 'new' add*

Small coefficients

$x^2 \oplus x^2 \oplus x^2 = (x^2 \oplus x^2) \oplus x^2$

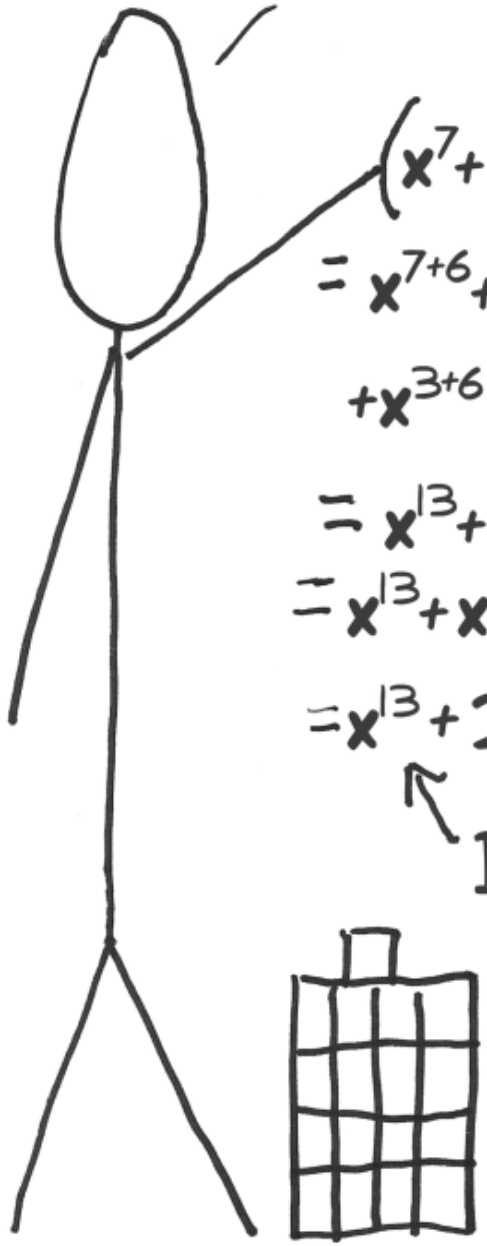
$= 0 \oplus x^2$

$= x^2$

i.e., Coefficients drawn from Z_2. You need material from CS0441 again!

*Nifty Fact: In the new way, addition is the same as subtraction (e.g. $x \oplus x = x - x = 0$)

Remember how multiplication could make things grow fast?



$$(x^7+x^5+x^3+x)\cdot(x^6+x^4+x^2+1)$$

$$= x^{7+6}+x^{7+4}+x^{7+2}+x^{7+0}+x^{5+6}+x^{5+4}+x^{5+2}+x^{5+0}$$

$$+x^{3+6}+x^{3+4}+x^{3+2}+x^{3+0}+x^{1+6}+x^{1+4}+x^{1+2}+x^{1+0}$$

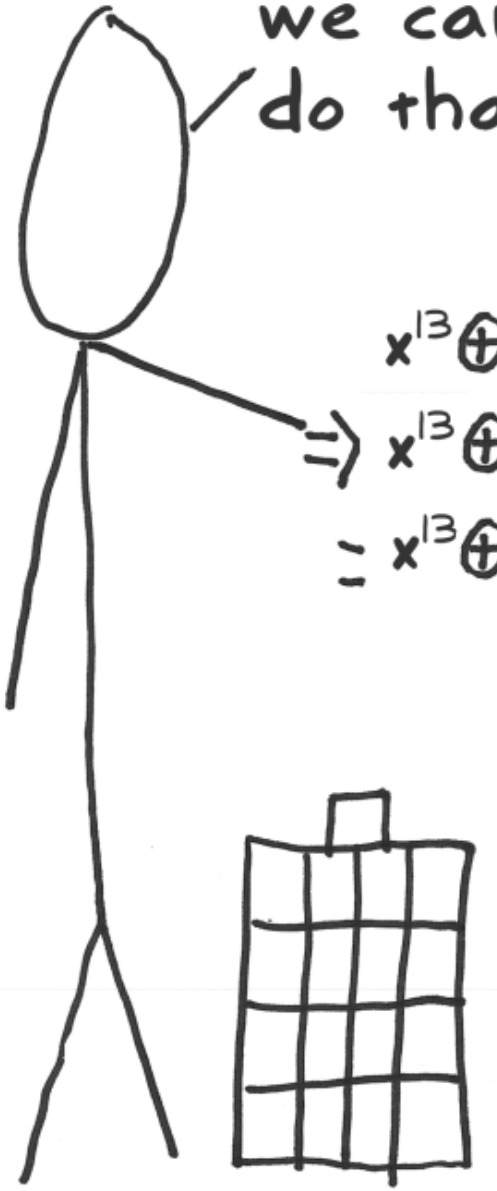
$$= x^{13}+x^{11}+x^9+x^7+x^{11}+x^9+x^7+x^5+x^9+x^7+x^5+x^3+x^7+x^5+x^3+x$$

$$= x^{13}+x^{11}+x^{11}+x^9+x^9+x^9+x^7+x^7+x^7+x^7+x^5+x^5+x^5+x^3+x^3+x$$

$$= x^{13}+2x^{11}+3x^9+4x^7+3x^5+2x^3+x$$

Big and yucky!

With the "new" addition, things are simpler, but the x^{13} is still too big. Let's make it so we can't go bigger than x^7. How can we do that?



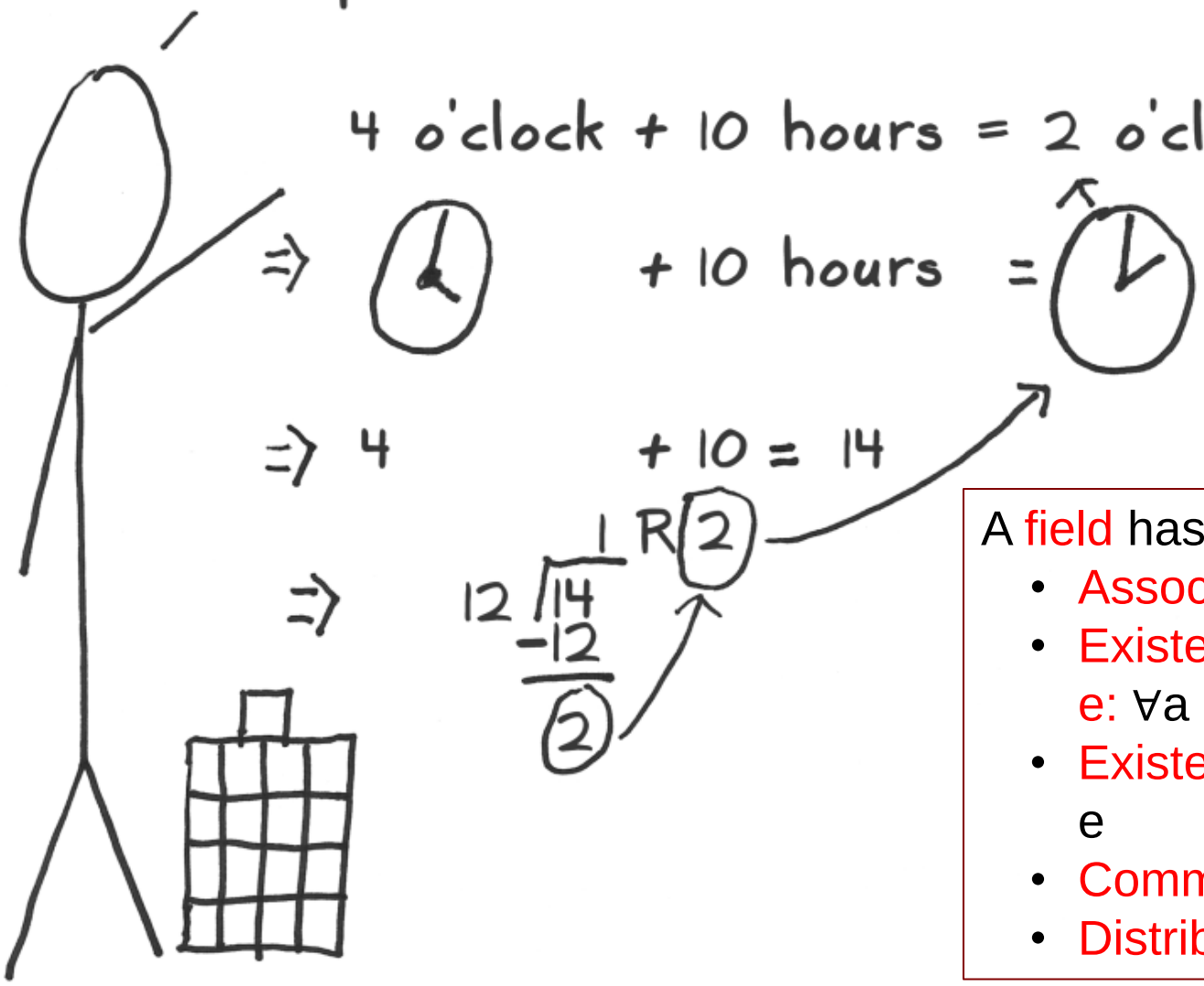
$$x^{13} \oplus 2x^{11} \oplus 3x^9 \oplus 4x^7 \oplus 3x^5 \oplus 2x^3 \oplus x$$

$$\Rightarrow x^{13} \oplus 0x^{11} \oplus x^9 \oplus 0x^7 \oplus x^5 \oplus 0x^3 \oplus x$$

$$= x^{13} \oplus x^9 \oplus x^5 \oplus x$$

Question: How might we accomplish this?

We use our friend, "clock math*," to do this. Just add things up and do long division. Keep a close watch on the remainder:



A field has the following properties:

- Associativity: (ab)c = a(bc)
- Existence of an identity element e: $\forall a : ae = a$
- Existence of inverses: $\forall a : aa^{-1} = e$
- Commutativity: ab = ba
- Distributivity: a(b+c) = ab + bc

* This is also known as "modular addition." Math geeks call this a "group." AES uses a special group called a "finite field."

We can do "clock" math with polynomials. Instead of dividing by 12, my creators told me to use $m(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$. Let's say we wanted to multiply $x \cdot b(x)$ where $b(x)$ has coefficients $b_7 \ldots b_0$:

$$x \cdot b(x)$$

$$= x \cdot (b_7 x^7 \oplus b_6 x^6 \oplus b_5 x^5 \oplus b_4 x^4 \oplus b_3 x^3 \oplus b_2 x^2 \oplus b_1 x \oplus b_0)$$

$$= b_7 x^8 \oplus b_6 x^7 \oplus b_5 x^6 \oplus b_4 x^5 \oplus b_3 x^4 \oplus b_2 x^3 \oplus b_1 x^2 \oplus b_0 x)$$

Eeek! x^8 is too big. We must make it smaller.

Note: Rather than reducing elements of our field modulo an integer, we will reduce them modulo another polynomial. This polynomial is irreducible, which means it's kind of like a prime number (i.e., it has no other factors).

* Remember that each b_n (e.g. b_7) is either 0 or 1.

We divide it by $m(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ and take the remainder:

$$
\begin{array}{r|l}
 & b_7 \\
\hline
x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1 & b_7x^8 \oplus b_6x^7 \oplus b_5x^6 \oplus b_4x^5 \oplus b_3x^4 \oplus b_2x^3 \oplus b_1x^2 \oplus b_0x \\
\oplus & b_7x^8 \qquad\qquad\qquad\qquad \oplus b_7x^4 \oplus b_7x^3 \oplus b_7x \oplus b_7 \\
\hline
 & b_6x^7 \oplus b_5x^6 \oplus b_4x^5 \oplus (b_3 \oplus b_7)x^4 \oplus (b_2 \oplus b_7)x^3 \\
 & \qquad\qquad \oplus b_1x^2 \oplus (b_0 \oplus b_7)x \oplus b_7
\end{array}
$$

$m(x)$

Remainder

$$b_6x^7 \oplus b_5x^6 \oplus b_4x^5 \oplus b_3x^4 \oplus b_2x^3 \oplus b_1x^2 \oplus b_0x$$

$$\oplus b_7 \cdot (x^4 \oplus x^3 \oplus x \oplus 1)$$

Note how the b's are shifted left by 1 spot.

This is just b_7 multiplied by a small polynomial.

Now we're ready for the hardest blast from the past: logarithms. After logarithms, everything else is cake! Logarithms let us turn multiplication into addition:

$$\log(x \cdot y) = \log(x) + \log(y)$$

So... $\log(10 \cdot 100) = \log(10^1) + \log(10^2)$

$= 2 + 1 = 3$

In reverse:

$\log^{-1}(1) = 10^1 = 10$

$\log^{-1}(2) = 10^2 = 100$

$\log^{-1}(3) = 10^3 = 1{,}000$

$\Rightarrow 10 \cdot 100 = 1{,}000$

We can use logarithms in our new world. Instead of using 10 as the base, we can use the simple polynomial of $x \oplus 1$ and watch the magic unravel.*

$$(x \oplus 1)^1 = x \oplus 1$$

$$(x \oplus 1)^2 = (x \oplus 1)(x \oplus 1) = x^2 \oplus x \oplus x \oplus 1 = x^2 \oplus 1$$

$$(x \oplus 1)^3 = (x \oplus 1)^2 \cdot (x \oplus 1) = x^3 \oplus x^2 \oplus x \oplus 1$$

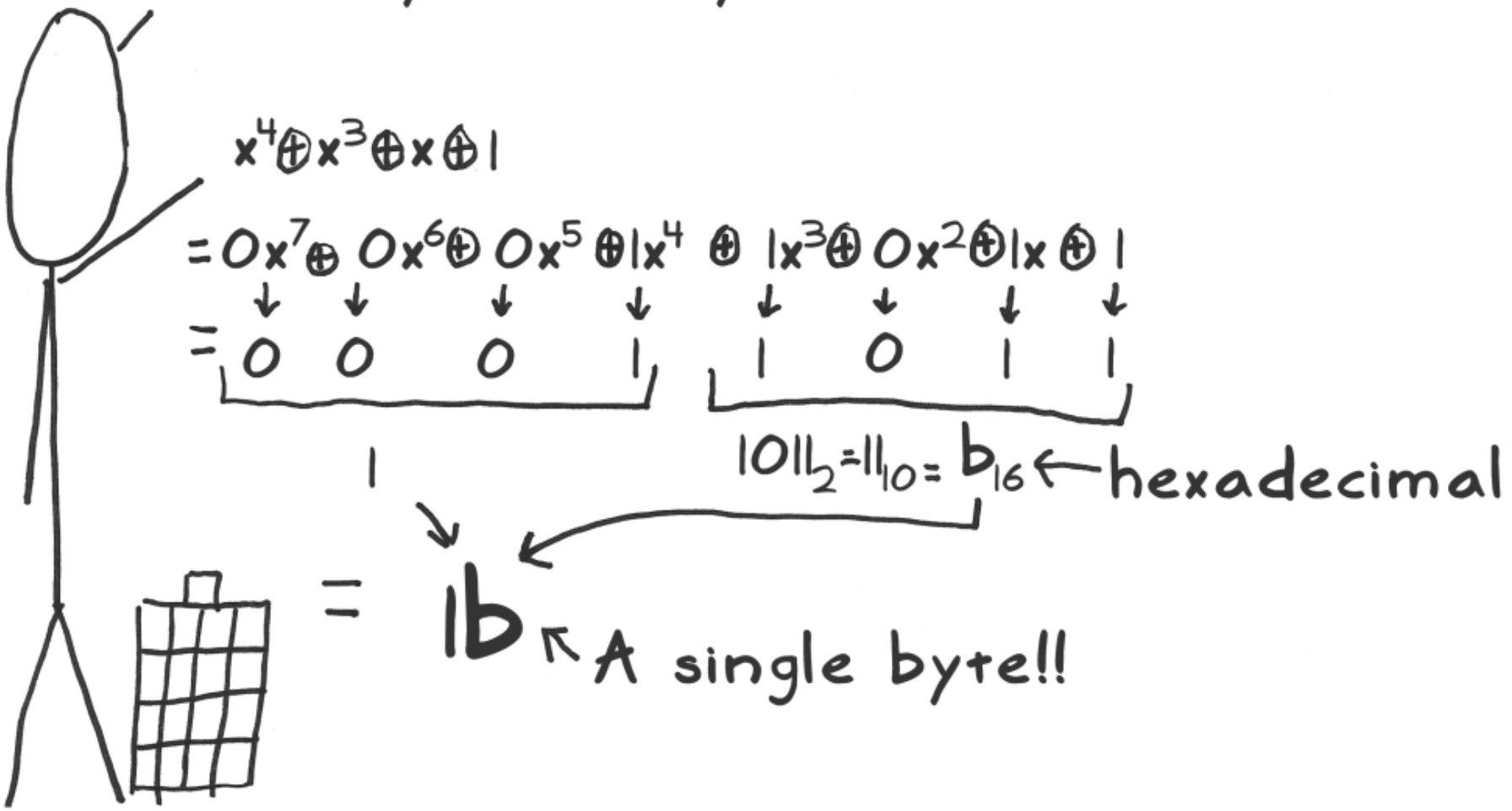
So...

$$\log_{x \oplus 1}(x \oplus 1) = 1, \log_{x \oplus 1}(x^2 \oplus 1) = 2, \log_{x \oplus 1}(x^3 \oplus x^2 \oplus x \oplus 1) = 3$$

The polynomial $(x \oplus 1)$ is called a <u>generator</u> of our field $GF(2^8)$

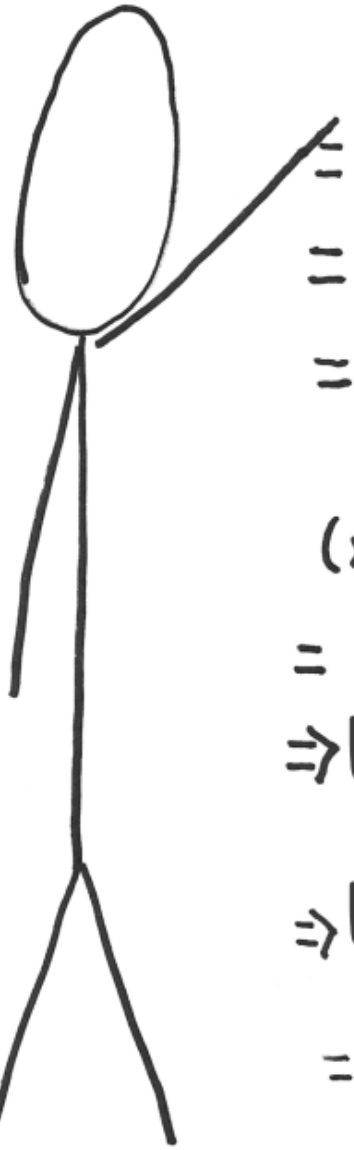
*If you keep multiplying by $(x \oplus 1)$ and then take the remainder after dividing by $m(x)$, you'll see that you generate all possible polynomial below x^8. This is very important!

Why bother with all of this math?* Encryption deals with bits and bytes, right? Well, there's one last connection: a 7^{th} degree polynomial can be represented in exactly 1 byte since the new way uses only 0 or 1 for coefficients:



*Although we'll work with bytes from now on, the math makes sure everything works out.

With bytes, polynomial addition becomes a simple xor. We can use our logarithm skills to make a table for speedy multiplication.*



$$(x^4 \oplus x^3 \oplus x \oplus 1) \oplus (x^7 \oplus x^5 \oplus x^3 \oplus x)$$

$$= 1b \oplus aa \quad \leftarrow \text{byte xor}$$

$$= b1$$

$$= x^7 \oplus x^5 \oplus x^4 \oplus 1$$

$$(x^4 \oplus x^3 \oplus x \oplus 1) \cdot (x^7 \oplus x^5 \oplus x^3 \oplus x)$$

$$= 1b \cdot aa \quad \text{logarithm table lookup}$$

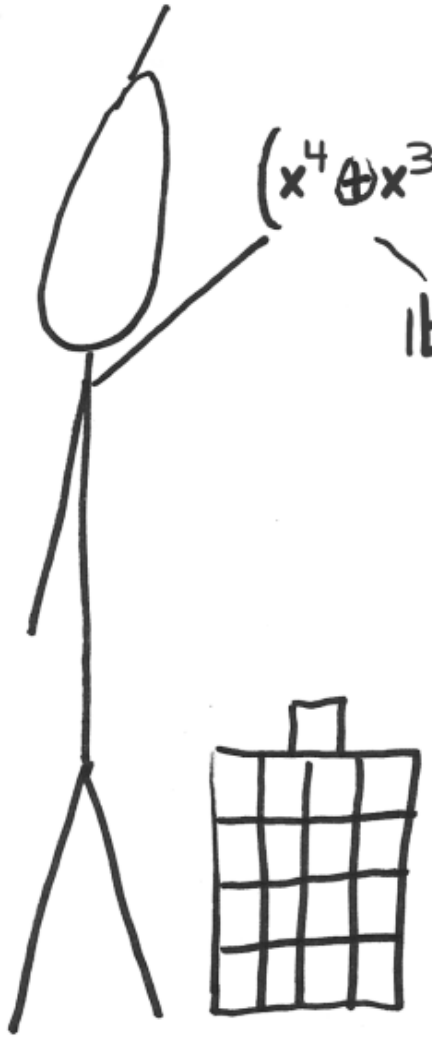
$$\Rightarrow \log(1b) + \log(aa) = c8 + 1f = e7$$

$$\Rightarrow \log^{-1}(e7) = 8c \Rightarrow 1b \cdot aa \quad \text{inverse table lookup}$$

$$= x^7 \oplus x^3 \oplus x^2$$

* We can create the table as we keep multiplying by $(x \oplus 1)$.

Since we know how to multiply, we can find the "inverse" polynomial byte for each byte. This is the byte that will undo/invert the polynomial back to 1. There are only 255* of them, so we can use brute force to find them:



$(x^4 \oplus x^3 \oplus x \oplus 1) \cdot ? = 1$

$1b \cdot cc = 1$

↖ found using a brute force for-loop

* There are only 255 instead of 256 because 0 has no inverse.

Now we can understand the mysterious s-box. It takes a byte "a" and applies two functions. The first is "g" which just finds the byte inverse. The second is "f" which intentionally makes the math uglier to foil attackers.

$$g(a) = a^{-1}$$

$$f(a) = \begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Why build our S-box using properties of the inverse over $GF(2^8)$? It is known to have good non-linearity properties. This makes cryptanalysis harder!

sbox[a] = f(g(a))

sbox[58] = f(g(58))

sbox[58] = f(18) = 6a

58·18=01

Translation

Linear transformation

The S-Box, Redux

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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
	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

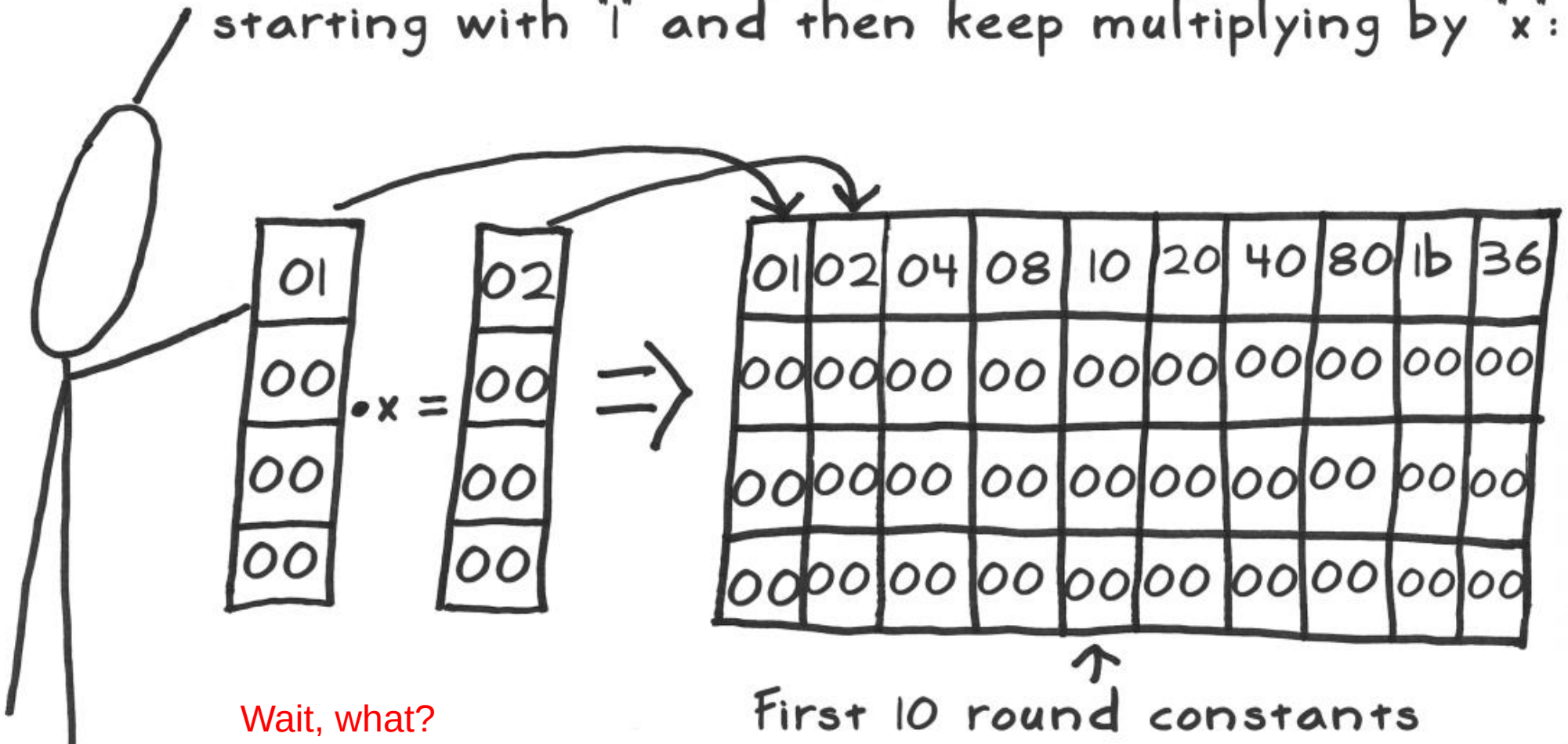
Recall: sbox[58] = f(g(58))

sbox[58] = f(18) = 6a

58·18=01

Doing the lookup is way faster than doing the actual math!

We can also understand those crazy round constants in the key expansion. I get them by starting with "1" and then keep multiplying by "x":



Wait, what?

$1 = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 0x + 1 = 0000\ 0001 = 01$

$1 \cdot x = x = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x + 0 = 0000\ 0010 = 02$

$x \cdot x = x^2 = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 1x^2 + 0x + 0 = 0000\ 0100 = 04$

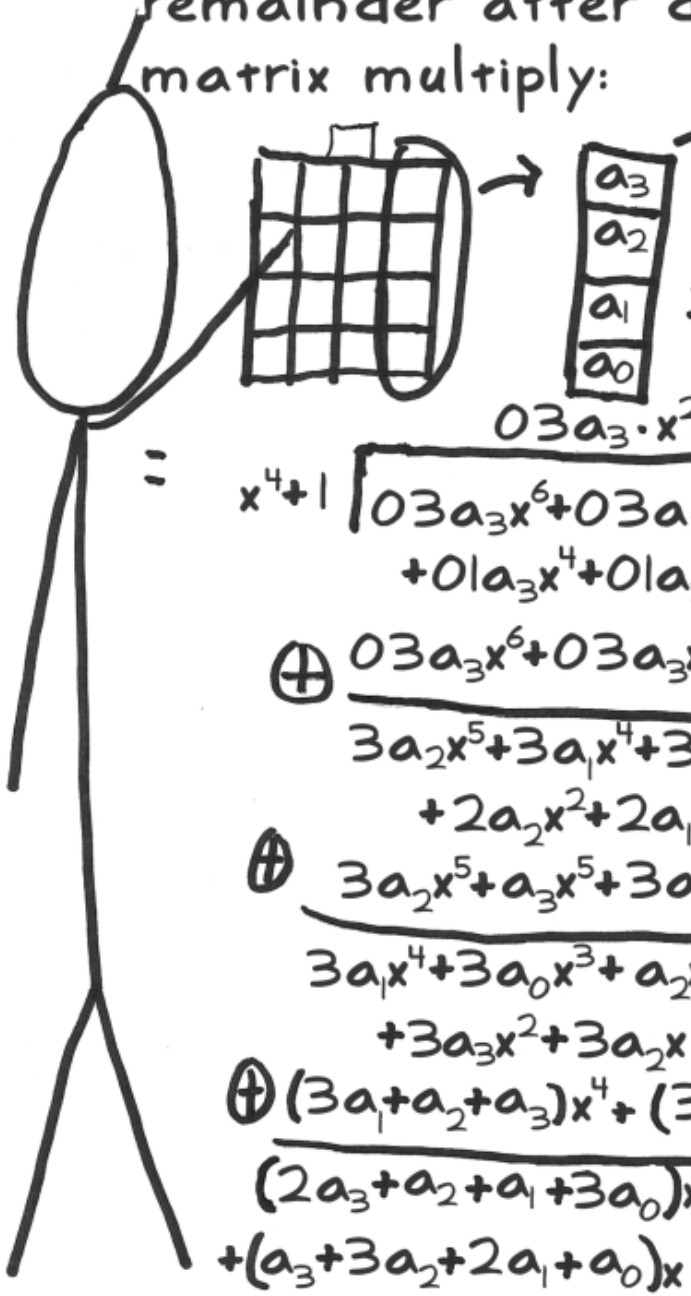
$x^2 \cdot x = x^3 = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 1x^3 + 0x^2 + 0x + 0 = 0000\ 1000 = 08$

…

Observation: We're just left rotating here…

Mix Columns is the hardest. I treat each column as a polynomial. I then use our new multiply method to multiply it by a specially crafted polynomial and then take the remainder after dividing by x^4+1. This all simplifies to a matrix multiply:

i.e., Table lookups and XORs



$$b(x) = c(x) \cdot a(x) \bmod x^4+1$$

$$= (03x^3+01x^2+01x+02) \cdot (a_3x^3+a_2x^2+a_1x+a_0) \bmod x^4+1$$

$(03x^3+01x^2+01x+02)$: special polynomial

$(a_3x^3+a_2x^2+a_1x+a_0)$: the column

$$= x^4+1 \;\Big|\; \begin{array}{l} 03a_3 \cdot x^2+(3a_2+a_3)x+(3a_1+a_2+a_3) \\ \hline 03a_3x^6+03a_2x^5+03a_1x^4+03a_0x^3+01a_3x^5+01a_2x^4+01a_1x^3+01a_0x^2 \\ \quad +01a_3x^4+01a_2x^3+01a_1x^2+01a_0x+02a_3x^3+02a_2x^2+02a_1x+02a_0 \end{array}$$

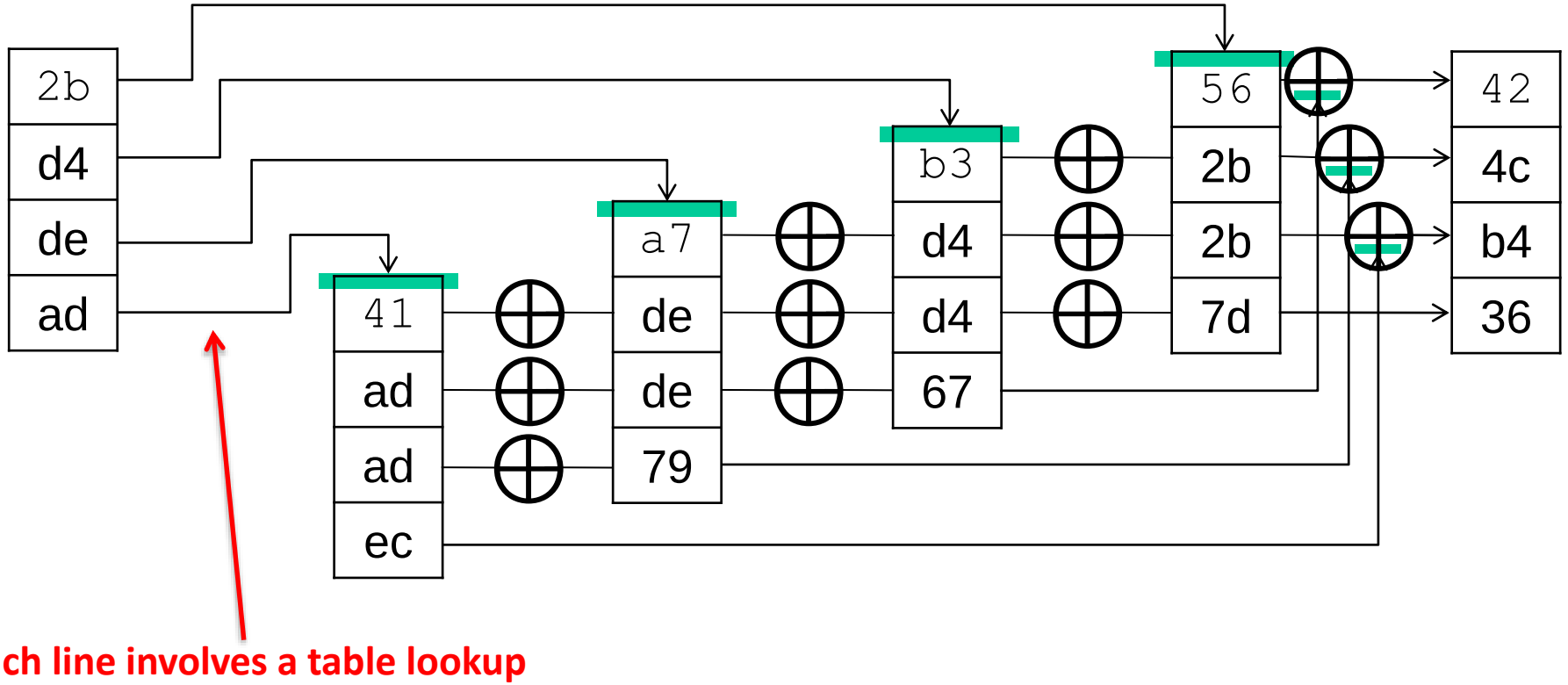
$$\oplus \; \frac{03a_3x^6+03a_3x^2}{3a_2x^5+3a_1x^4+3a_0x^3+a_3x^5+a_2x^4+a_1x^3+a_0x^2+a_3x^4+a_2x^3+a_1x^2+a_0x+2a_3x^3 \\ +2a_2x^2+2a_1x+2a_0+3a_3x^2}$$

$$\oplus \; \frac{3a_2x^5+a_3x^5+3a_2x+a_3x}{3a_1x^4+3a_0x^3+a_2x^4+a_1x^3+a_0x^2+a_3x^4+a_2x^3+a_1x^2+a_0x+2a_3x^3+2a_2x^2+2a_1x+2a_0 \\ +3a_3x^2+3a_2x+a_3x}$$

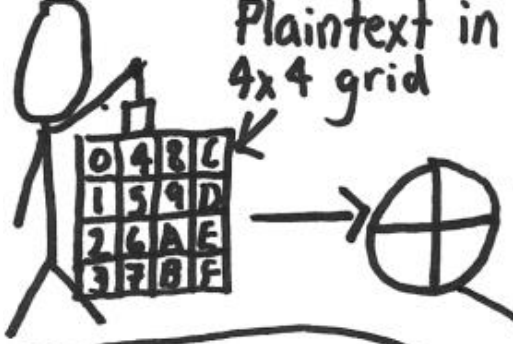
$$\oplus \; \frac{(3a_1+a_2+a_3)x^4+(3a_1+a_2+a_3)}{(2a_3+a_2+a_1+3a_0)x^3+(3a_3+2a_2+a_1+a_0)x^2 \\ +(a_3+3a_2+2a_1+a_0)x+(a_3+a_2+3a_1+2a_0)}$$

$$\Rightarrow \begin{bmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix}$$

MixColumns is essentially table lookups and XORs



AES Crib Sheet
(Handy for memorizing)

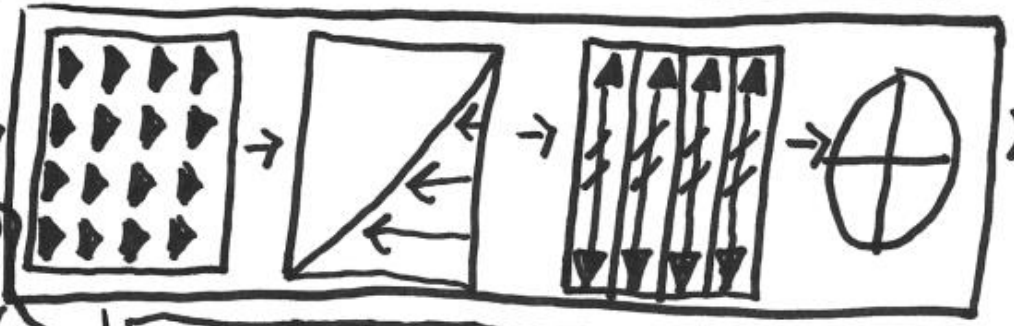


Initial Round

Intermediate Rounds

#	Key
9	128
11	192
13	256

X



General Math

11B = AES Polynomial = m(x)

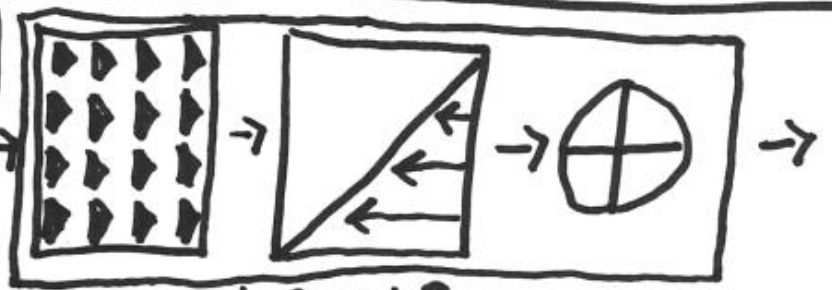
$x^8 + x^4 + x^3 + x + 1$

Fast Multiply

$x \cdot a(x) = (a << 1) \oplus (a_7 == 1)\ ?\ 1B : 00$

$\log(x \cdot y) = \log(x) + \log(y)$

Use $(x+1) = 03$ for log base



Final Round

Ciphertext

S-Box (SRD)

SRD[a] = f(g(a))

$g(a) = a^{-1} \bmod m(x)$

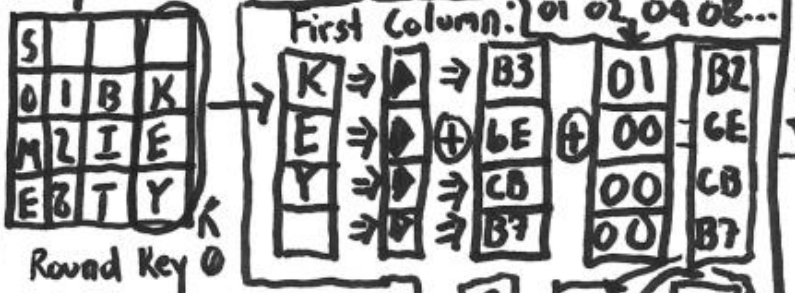
f(a) Think $\vec{5}3 \oplus 63^T$

5 1's and 3 0's $[0110\ 0011]^T$

$$\begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Key Expansion:

Round Constants: 01 02 04 08 ...



Other Columns:

Prev Col ⊕ Col from Previous round key

Mix Columns:

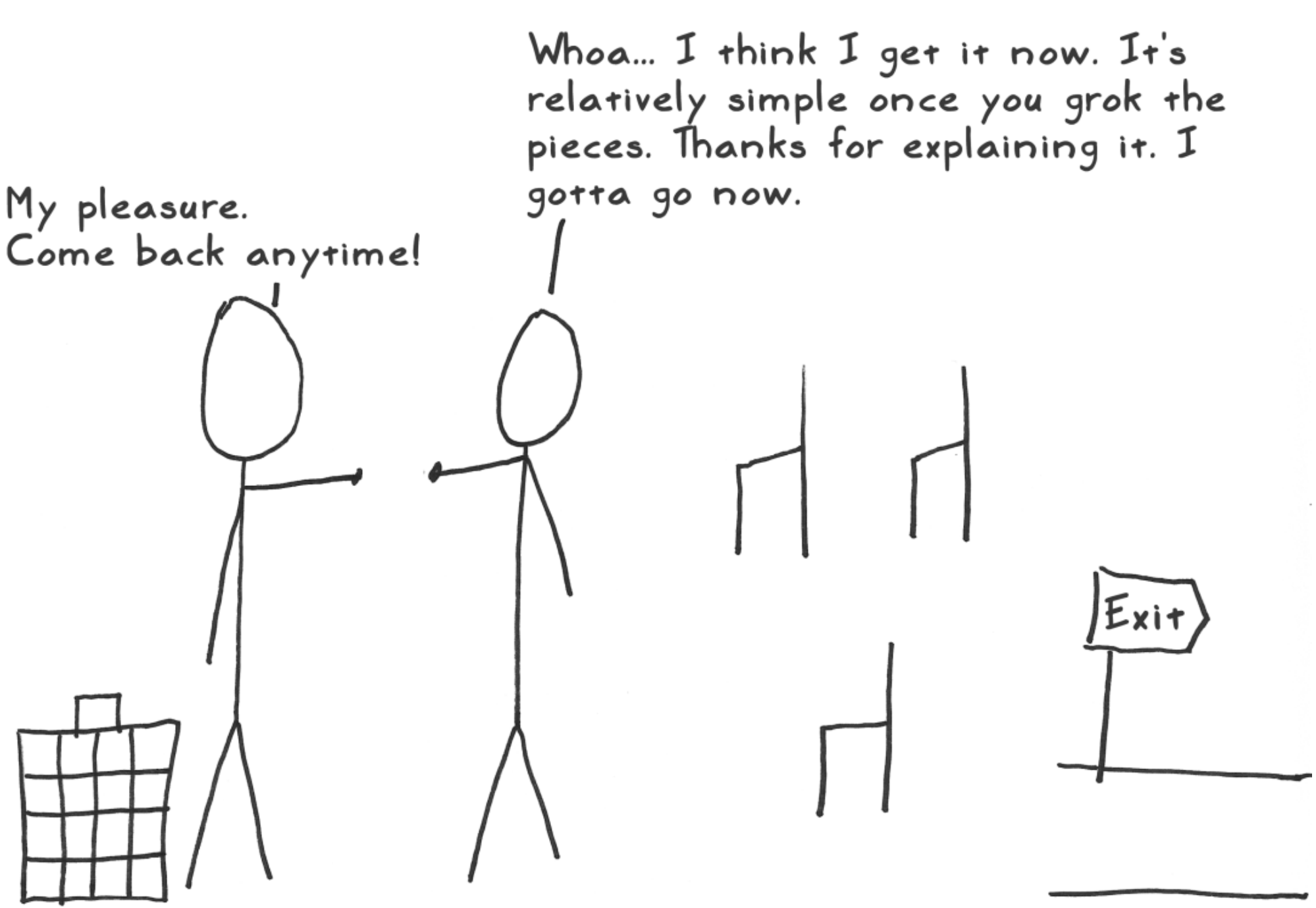
2113

$$\begin{bmatrix} 2&1&1&3 \\ 3&2&1&1 \\ 1&3&2&1 \\ 1&1&3&2 \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

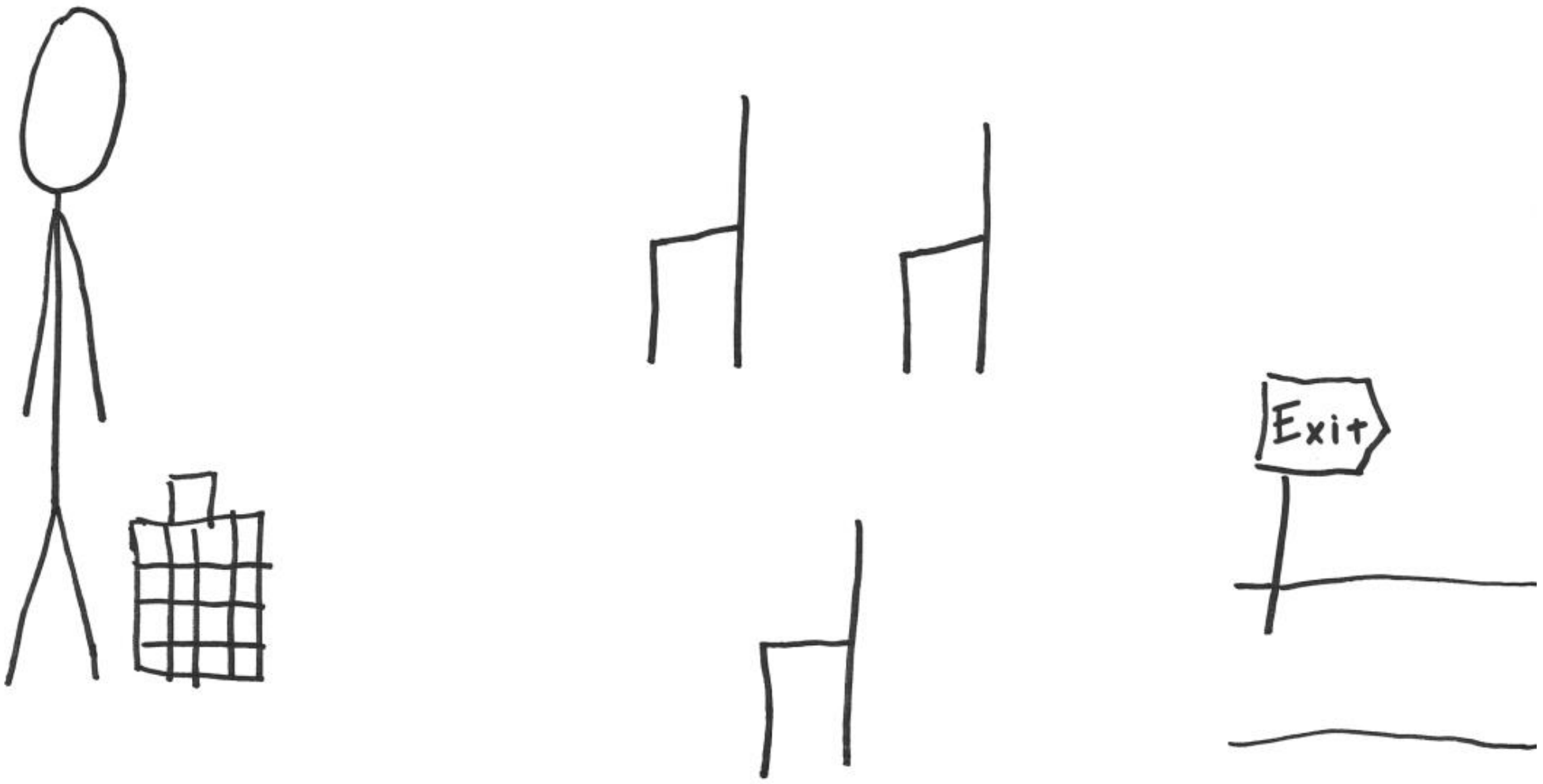
Inverse Mix

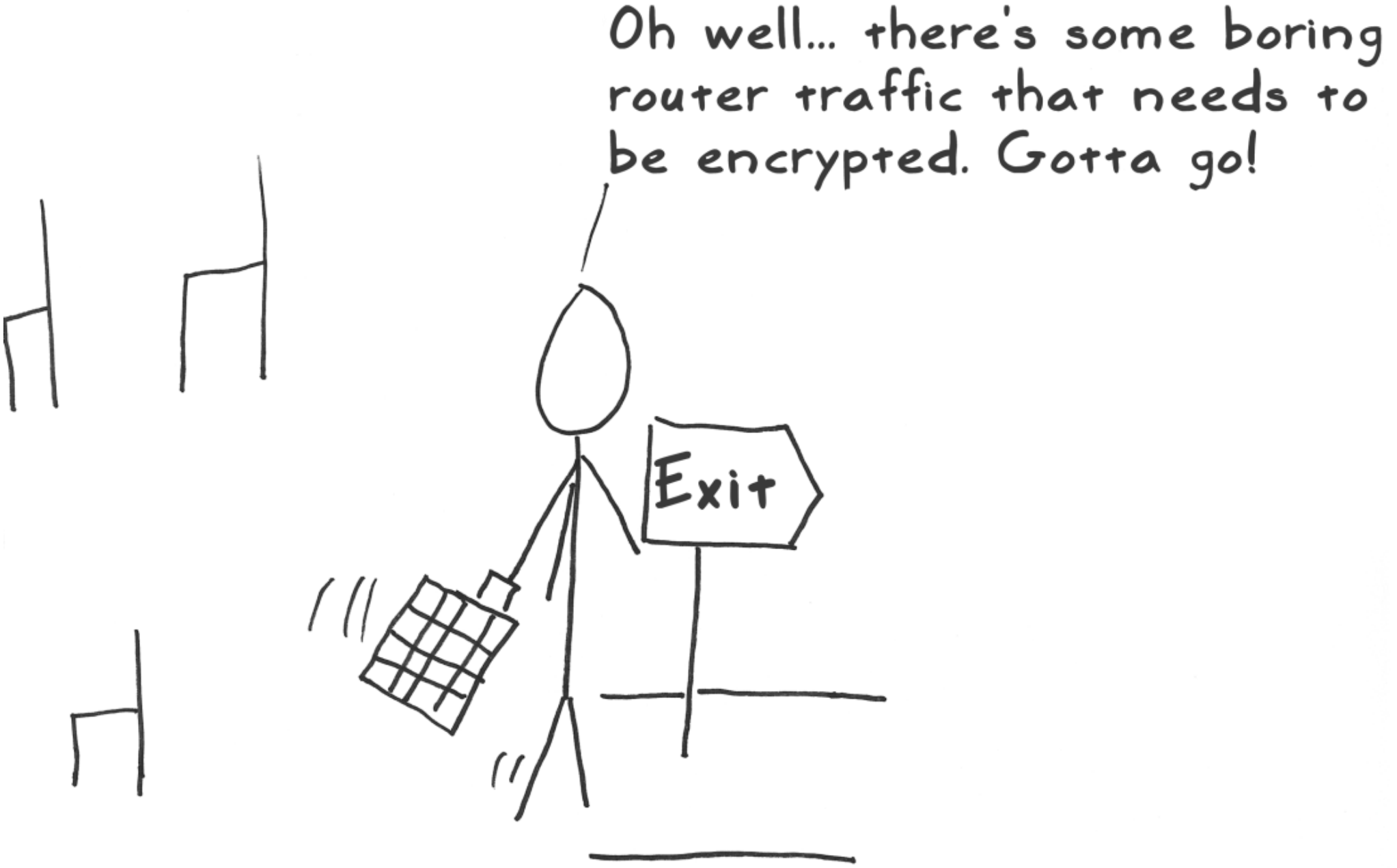
EBD9

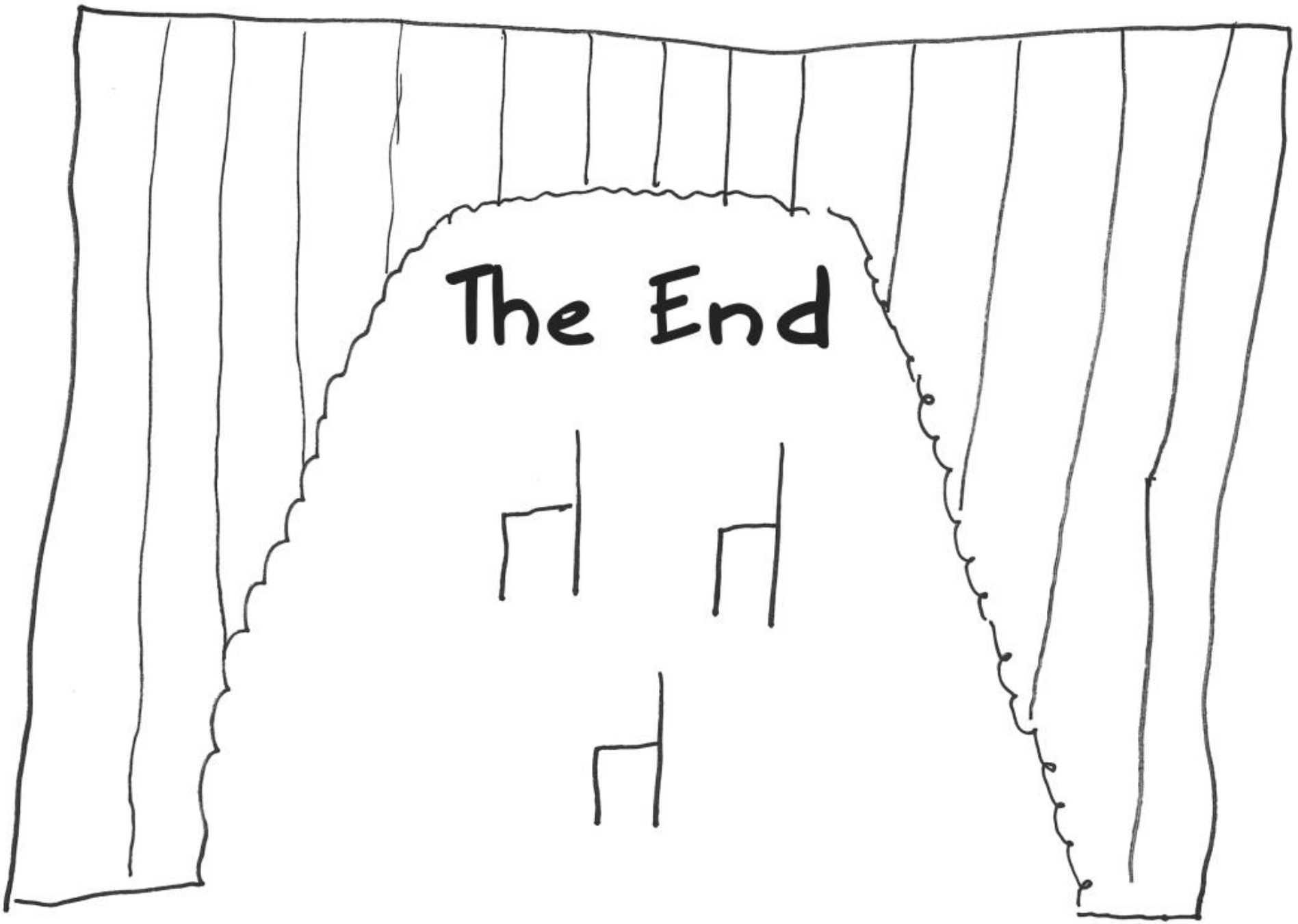
$$\begin{bmatrix} E&B&D&9 \\ 9&E&B&D \\ D&9&E&B \\ B&D&9&B \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$


Whoa... I think I get it now. It's relatively simple once you grok the pieces. Thanks for explaining it. I gotta go now.
My pleasure.
Come back anytime!
Exit

But there's so much more to talk about: my resistance to linear and differential cryptanalysis, my Wide Trail Strategy, impractical related-key attacks, and... so much more... but no one is left.




Oh well... there's some boring router traffic that needs to be encrypted. Gotta go!
Exit


The End

Final Thoughts…

In theory, you now understand how 128-bit AES works

Low level:

Good non-linearity properties

- S-Box is not random, it's based on inverses in GF(2^8)
- Constants in MixColumns also derived from math in GF(2^8)
- All of the insane shifting, XORing, and table lookups are really additions, multiplications, and inversion of polynomials in disguise!

Take away point: Crypto is difficult to implement properly, let alone design. Unless you plan to pursue higher education in mathematics, probably best to understand it as a tool, but leave the design to experts ☺

One alternative was Blowfish, by Schneier et al.

Designed in 1993

No patents

64-bit block size

Huge subkeys

Slow to switch keys

Fast otherwise

…

Now, we'll look closely at Blowfish

Despite its long history, Blowfish is still interesting to study

Still no bad cryptographic breaks

- (Some weak keys)

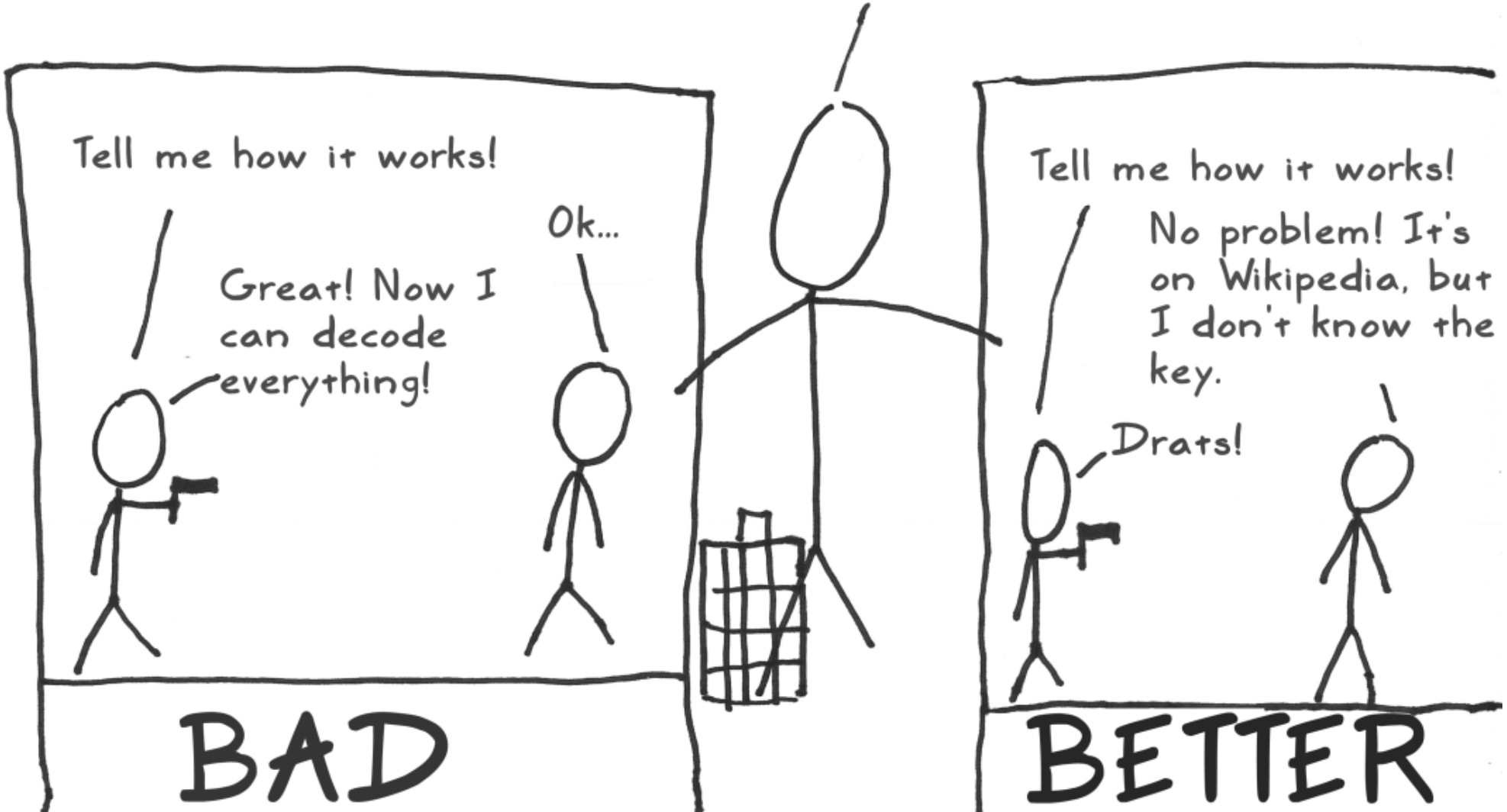
Some very interesting properties that Rijndael does not have

- Key-dependent S-boxes
- Long key schedule (computing subkeys)
- Pi (?!?)
- …

Big Idea #3: Secrecy Only in the Key

Kerchoffs's principle!

After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.



Foot-Shooting Prevention Agreement

I, _____ , promise that once
Your Name

Blowfish

I see how simple ~~AES~~ really is, I will not implement it in production code even though it would be really fun.

This agreement shall be in effect until the undersigned creates a meaningful interpretive dance that compares and contrasts cache-based, timing, and other side channel attacks and their countermeasures.

X _____ _____
Signature Date

First of all, what are these subkeys you keep mentioning?

Data computed from the key, used in encryption

P-array

- 18 subkeys
- 32 bits each
- $P_1, P_2, ..., P_{18}$

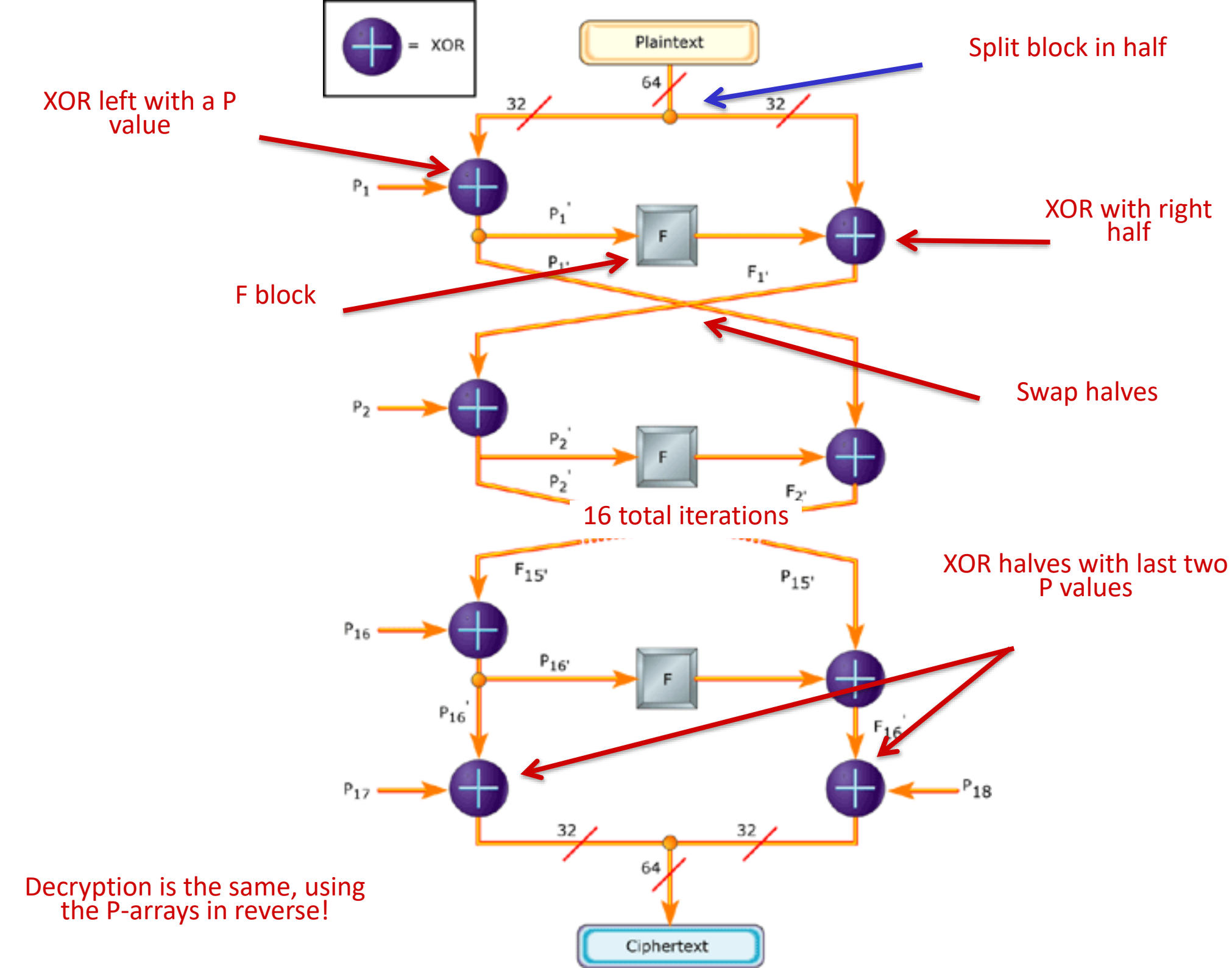
72 bytes

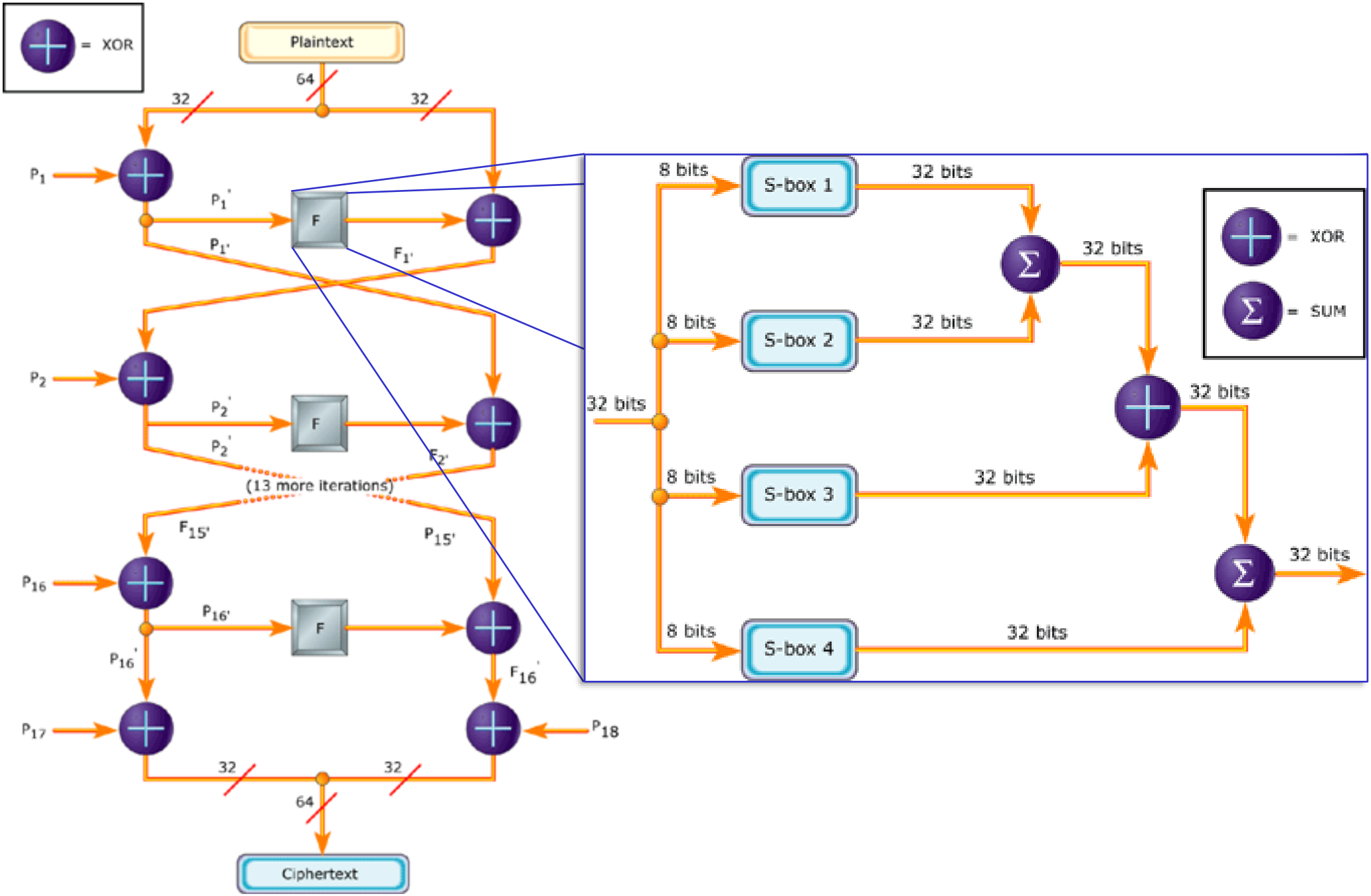
S-boxes

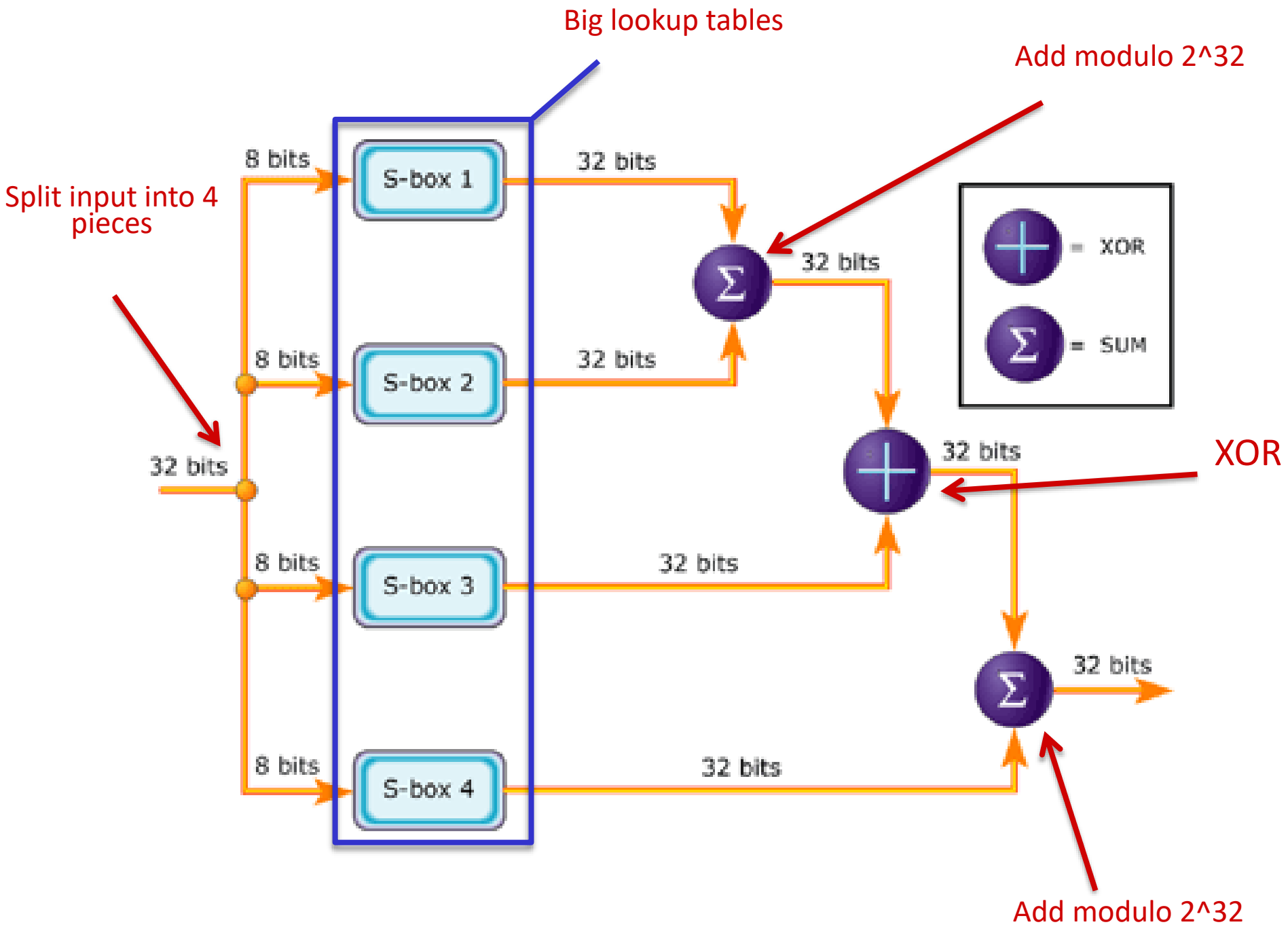
- 4 lookup tables
- 8 bits → 32 bits
- $S_{1,0}, S_{1,1}, ..., S_{1,255}$;
  $S_{2,0}, S_{2,1}, ..., S_{2,255}$;
  $S_{3,0}, S_{3,1}, ..., S_{3,255}$;
  $S_{4,0}, S_{4,1}, ..., S_{4,255}$;

4096 bytes

18 32-bit P values + four 8-bit to 32-bit S-boxes
= 4168 bytes from (max) 448-bit (56 byte) key!


= XOR
Plaintext
64
32
32
Split block in half
XOR left with a P value
P1
P1'
F
XOR with right half
F block
F1'
Swap halves
P2
P2'
F
P2'
F2'
16 total iterations
F15'
P15'
XOR halves with last two P values
P16
P16'
F
P16'
F16'
P17
P18
32
32
64
Ciphertext
Decryption is the same, using the P-arrays in reverse!


= XOR
Plaintext
64
32
32
P1
P1'
F
P1'
F1'
P2
P2'
F
P2'
F2'
(13 more iterations)
F15'
P15'
P16
P16'
F
P16'
F16'
P17
P18
32
32
64
Ciphertext
8 bits
S-box 1
32 bits
32 bits
8 bits
S-box 2
32 bits
32 bits
32 bits
8 bits
S-box 3
32 bits
32 bits
8 bits
S-box 4
32 bits
= XOR
= SUM


Big lookup tables
Add modulo 2^32
Split input into 4 pieces
8 bits
S-box 1
32 bits
32 bits
= XOR
= SUM
8 bits
S-box 2
32 bits
32 bits
XOR
32 bits
8 bits
S-box 3
32 bits
32 bits
8 bits
S-box 4
32 bits
Add modulo 2^32

Where do the P-array and S-boxes come from?

1. P-array and S-boxes (4168 bits total) depend on 448-bit (56 byte) key
2. Fill P-array and S-boxes with the (hex) digits of Pi (Π=3.14159265359……)
3. XOR the key into the P-array

   $P_1 = P_1 \oplus$ first 32 bits of key

   $P_2 = P_2 \oplus$ second 32 bits of key

   … repeat key as needed

   521 encryptions!!

4. Encrypt 0 string, replace P_1, P_2 with output (64-bit)
5. Encrypt output, replace P_3, P_4 with new output
6. Repeat until entire P-array and all S-boxes are replaced

Discussion question!

Why initialize the constants with the digits of pi?

Nothing up my sleeve!

Discussion question!

Which step in Blowfish is very inefficient?

Deriving subkeys, changing keys

In what way is that a good thing?

Makes brute forcing hard!

Blowfish Summary

You now understand how Blowfish applies confusion and diffusion to 64-bit blocks

1. Key expansion: P-array and S-boxes

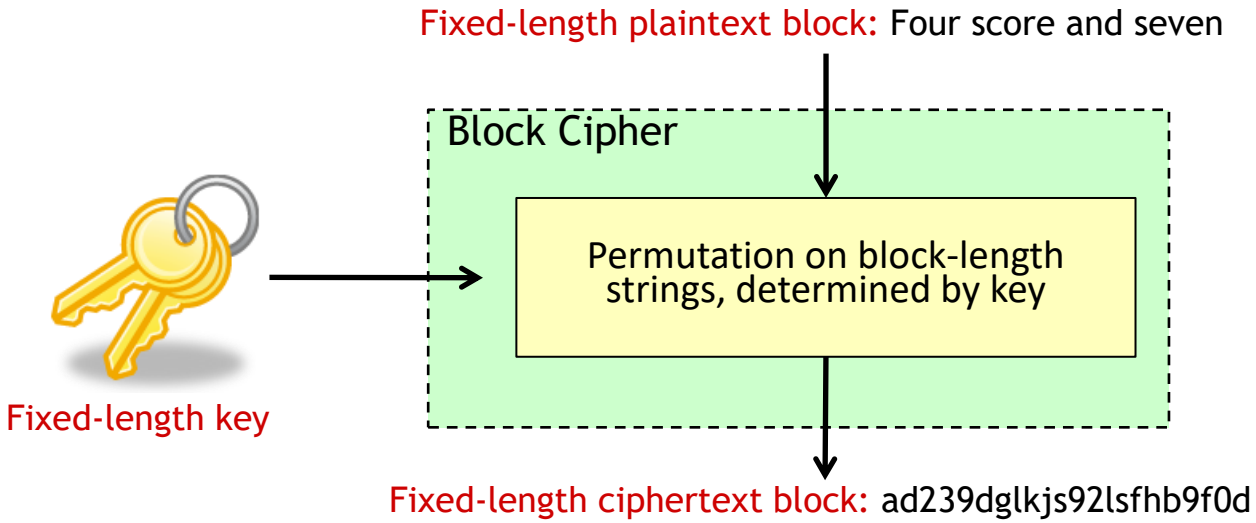
   Long key schedule, 521 encrypts (4 KB)
2. XOR with a P value
3. Break into 4 chunks, feed to S-boxes
4. Recombine with XORs and modular additions
5. Swap halves and repeat 2–4

How is that different from AES?

AES uses Additions, multiplications, and inversion of polynomials in GF(2^8) disguised as column shifts, XORs, and huge table lookups

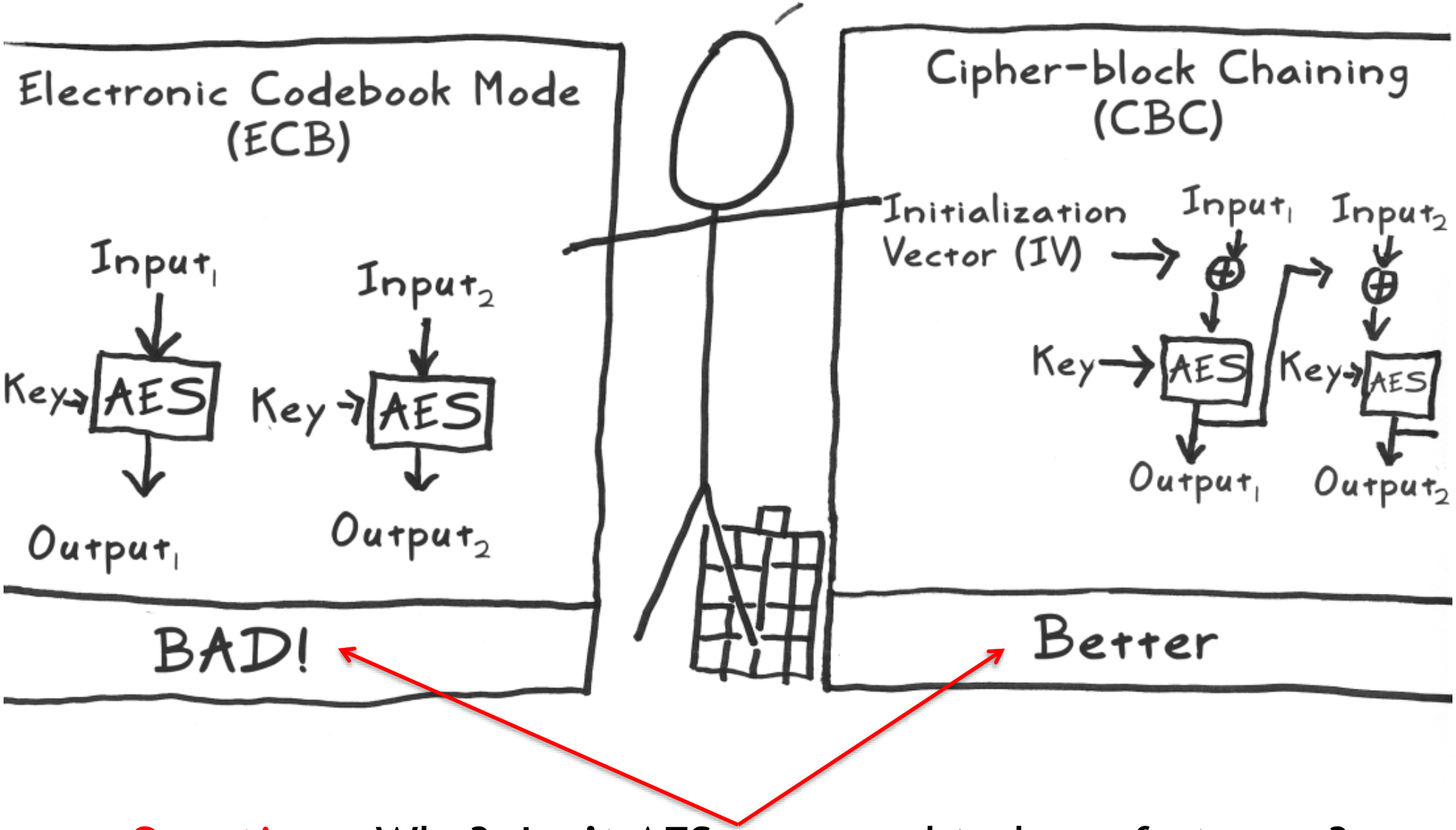
So what? Crypto is difficult to implement properly, let alone design. Understand it as a tool, but leave design/implementation to experts

Block Cipher Modes of Operation



Question: What happens if we need to encrypt more than one block of plaintext?

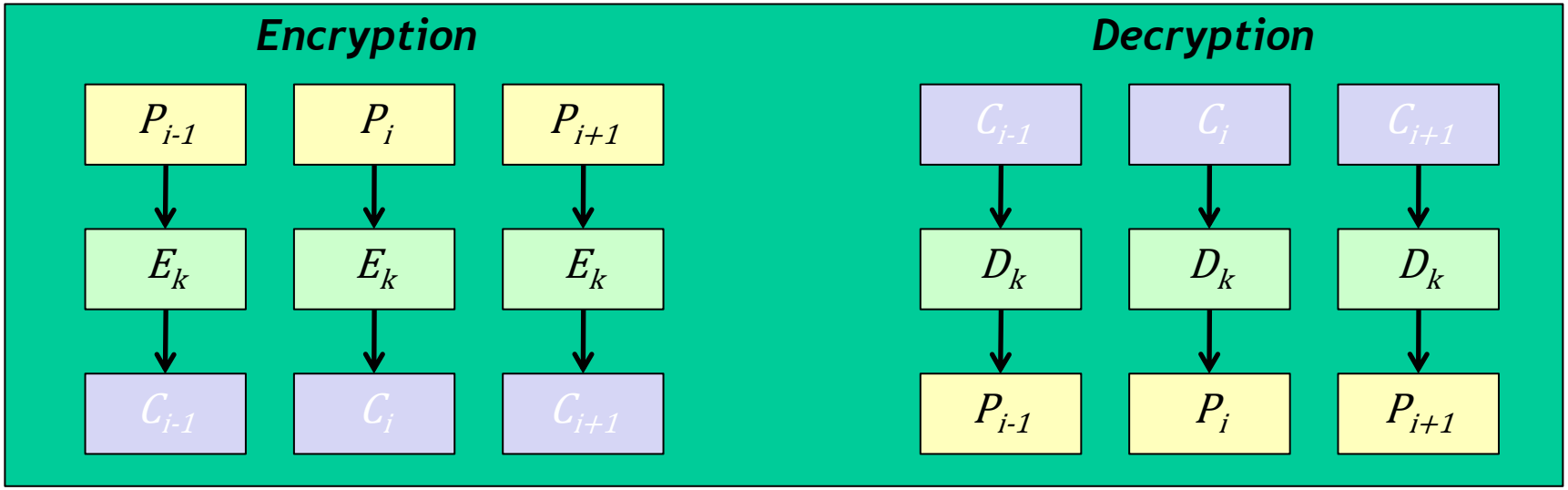
One last tidbit: I shouldn't be used as-is, but rather as a building block to a decent "mode."



Question: Why? Isn't AES supposed to be safe to use?

Block ciphers have many modes of operation

The most obvious way of using a block cipher is called electronic codebook mode (ECB)



Benefit: Errors in ciphertext do not propagate past a single block

What is wrong with ECB?

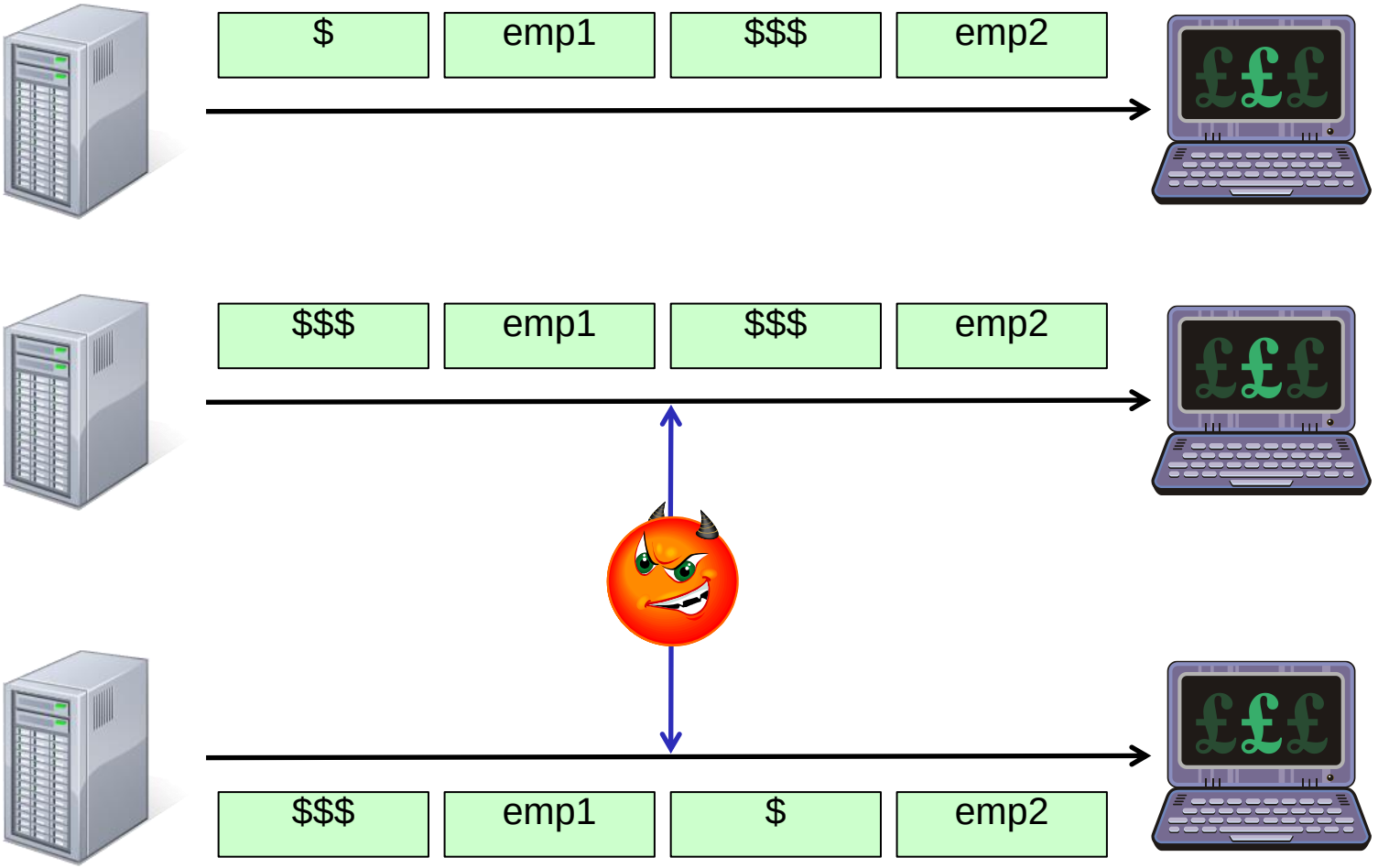
- Known plaintext/ciphertext pairings
- Block replay attacks

This is called a code book

In general, using ECB mode is not a great idea…

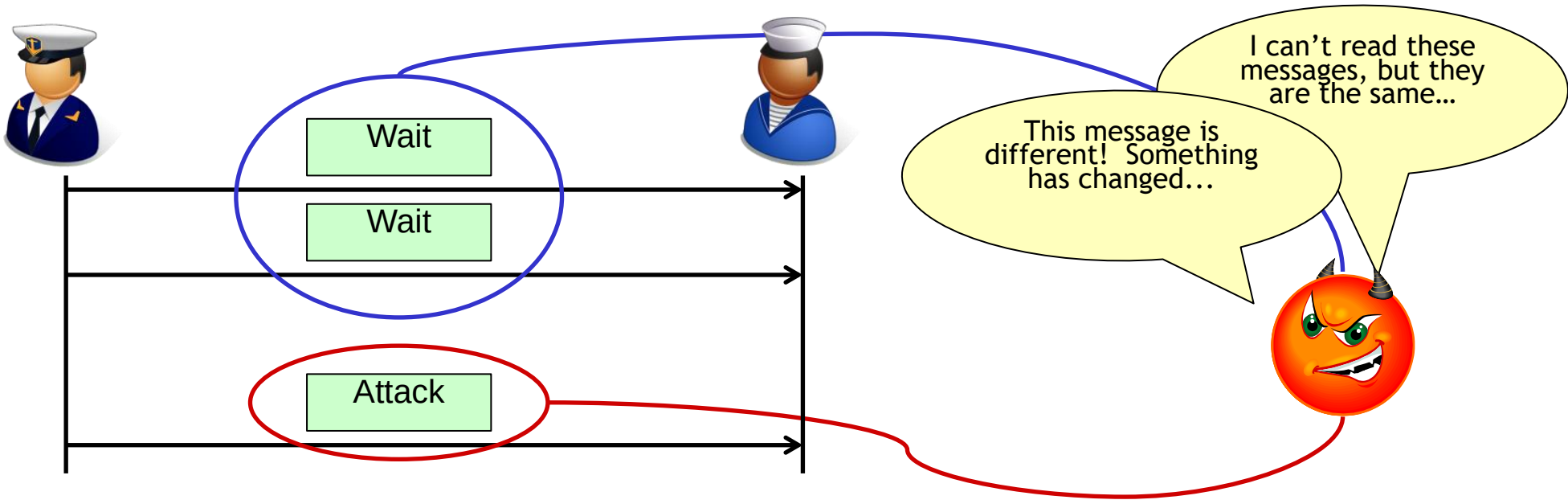
The use of ECB mode can lead to block replay or substitution attacks

Example: Salary data transmitted using ECB



Why is the ability to build a codebook dangerous?

Observation: When using ECB, the same block will always be encrypted the same way



To protect against this type of guessing attack, we need our cryptosystem to provide us with semantic security.