

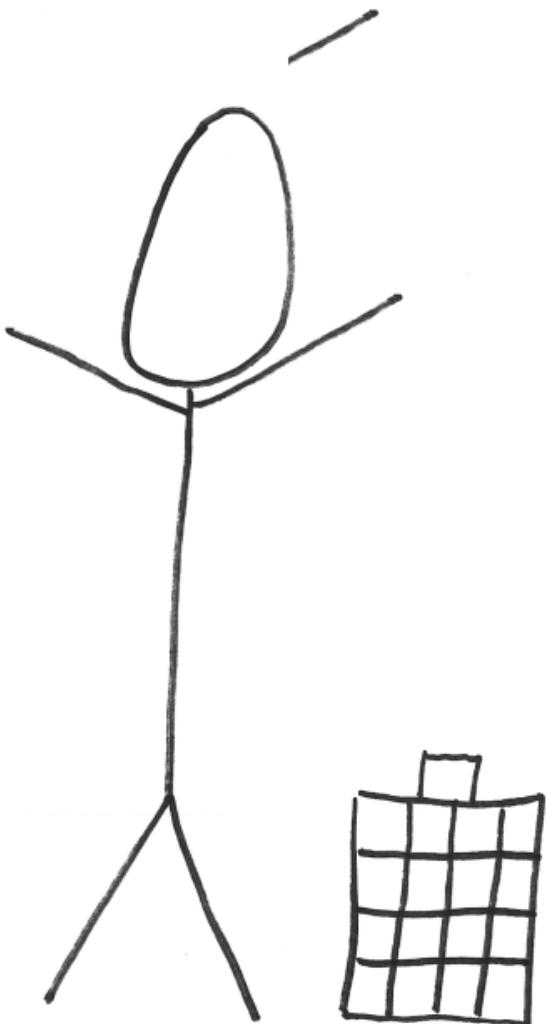
# A Stick Figure Guide to the Advanced Encryption Standard (AES)



© Copyright 2009, Jeff Moser  
<http://www.moserware.com/>

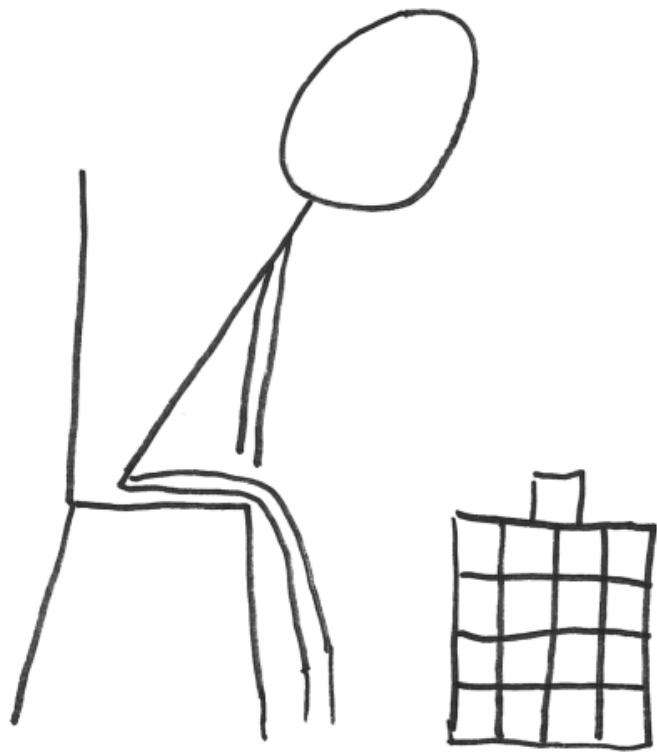
# Act 1: Once Upon a Time...

I handle petabytes\* of data every day. From encrypting juicy Top Secret intelligence to boring packets bound for your Wifi router, I do it all!

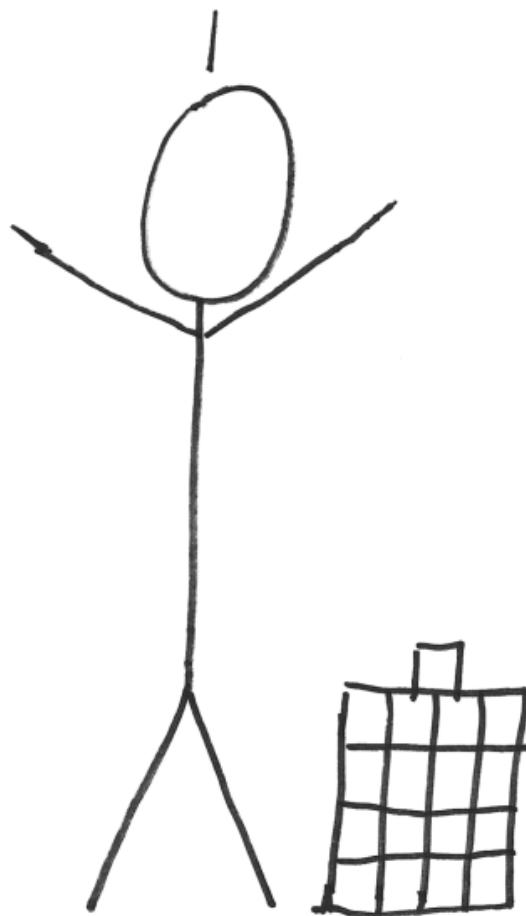


\* 1 petabyte ≈ a lot

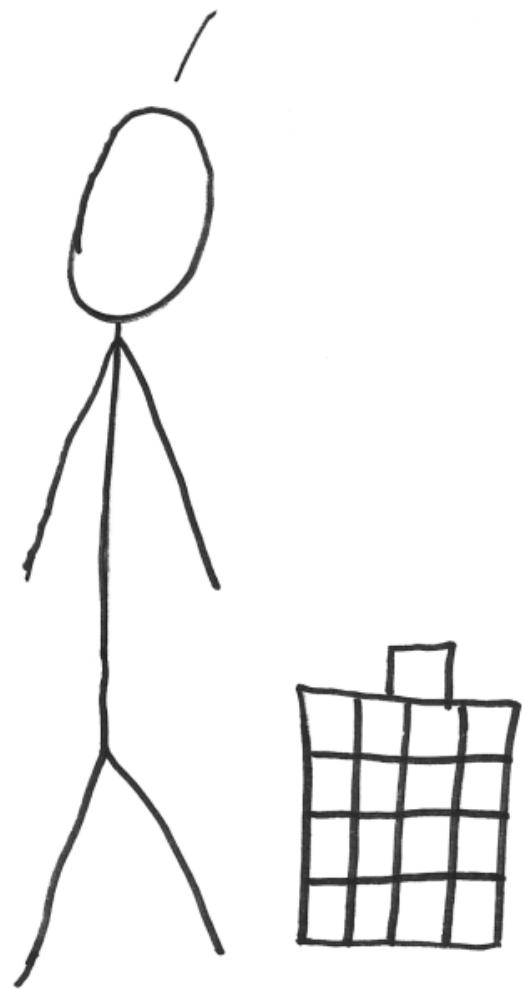
... and still no one seems to care  
about me or my story.



I've got a better-than-Cinderella  
story as I made my way to become  
king of the block cipher world.



Whoa! You're still there. You want  
to hear it? Well let's get started...



Once upon a time,\* there was no good way for people outside secret agencies to judge good crypto.

EBG13 vf terner!

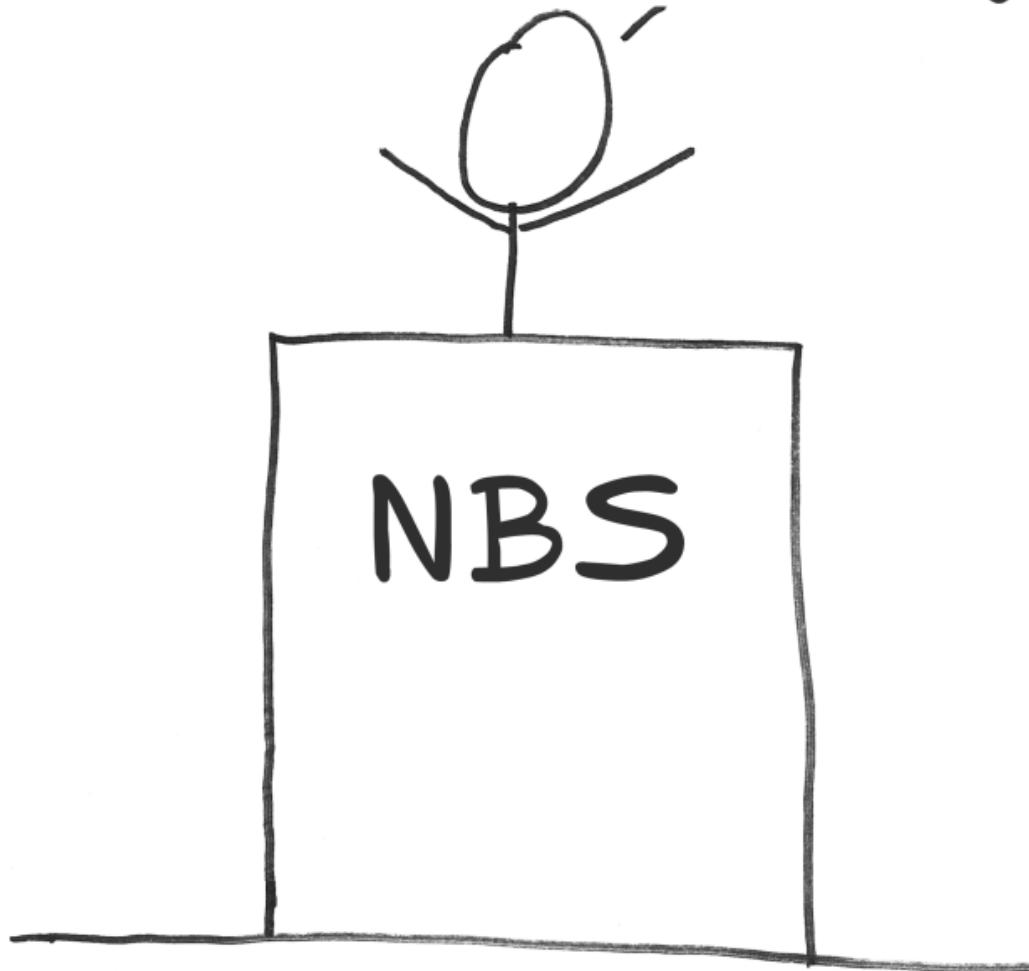
Double ROT13 –  
is better!



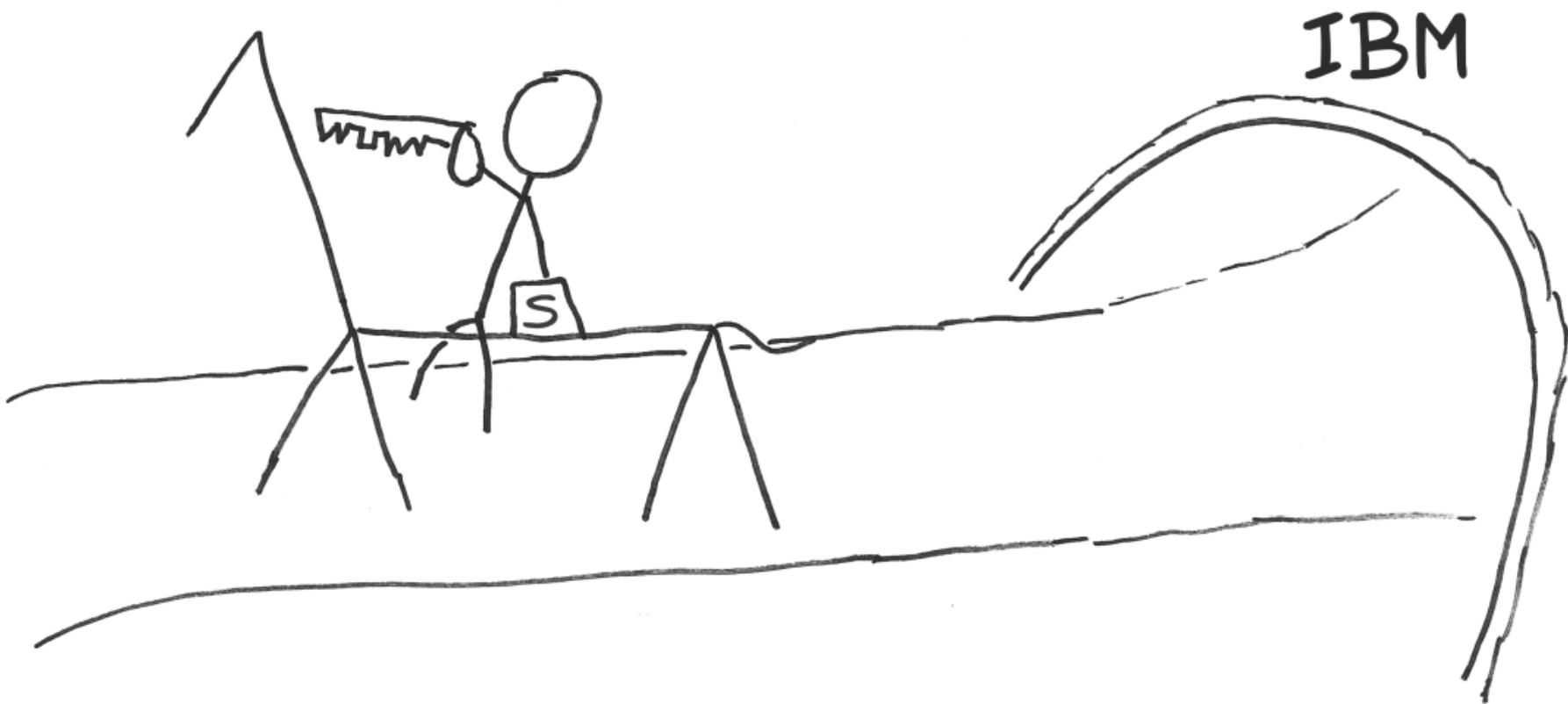
\* ~ pre-1975 for the general public

A decree went throughout the land to find a good, secure, algorithm.

We need a good cipher!

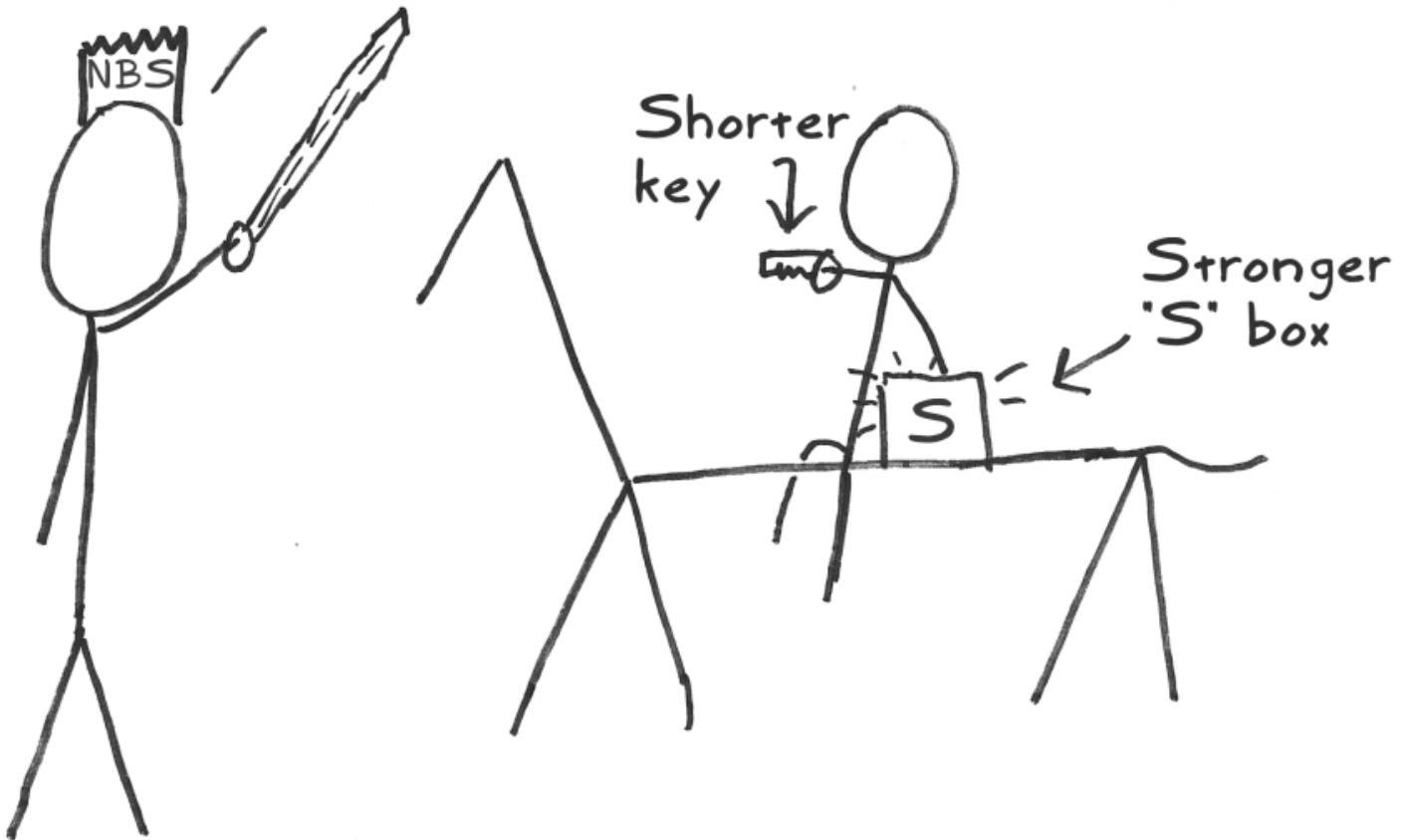


One worthy competitor named Lucifer came forward.



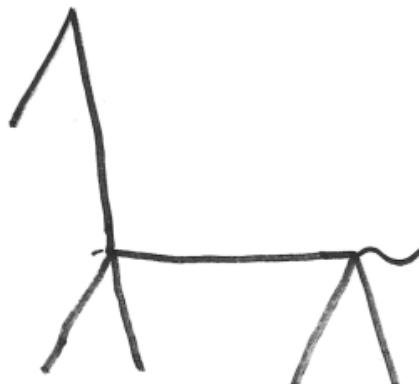
After being modified by the National Security Agency (NSA), he was anointed as the Data Encryption Standard (DES).

I anoint thee as DES!



DES ruled in the land for over 20 years. Academics studied him intently. For the first time, there was something specific to look at. The modern field of cryptography was born.

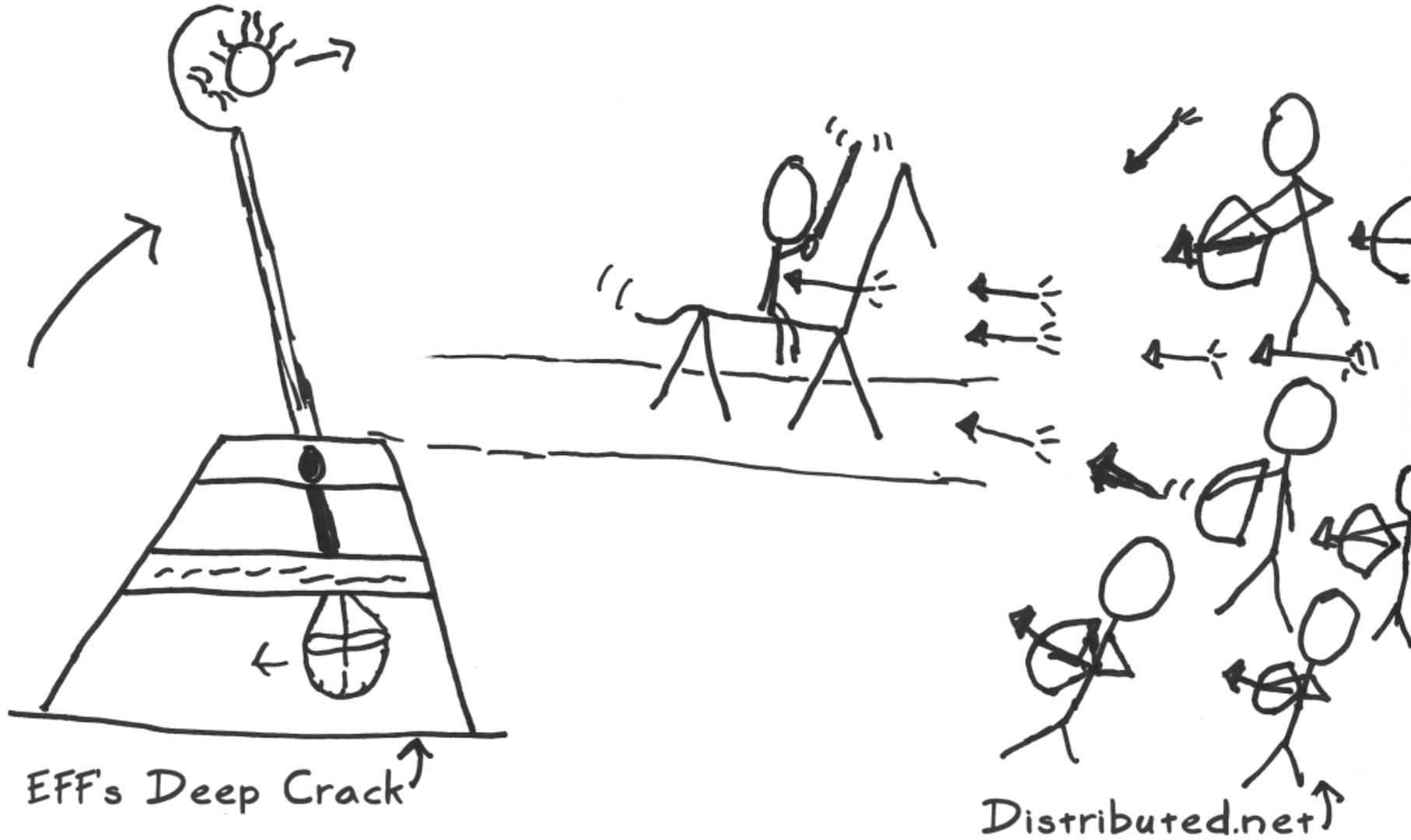
"... to the best of our knowledge, DES is free from any statistical or mathematical weakness."



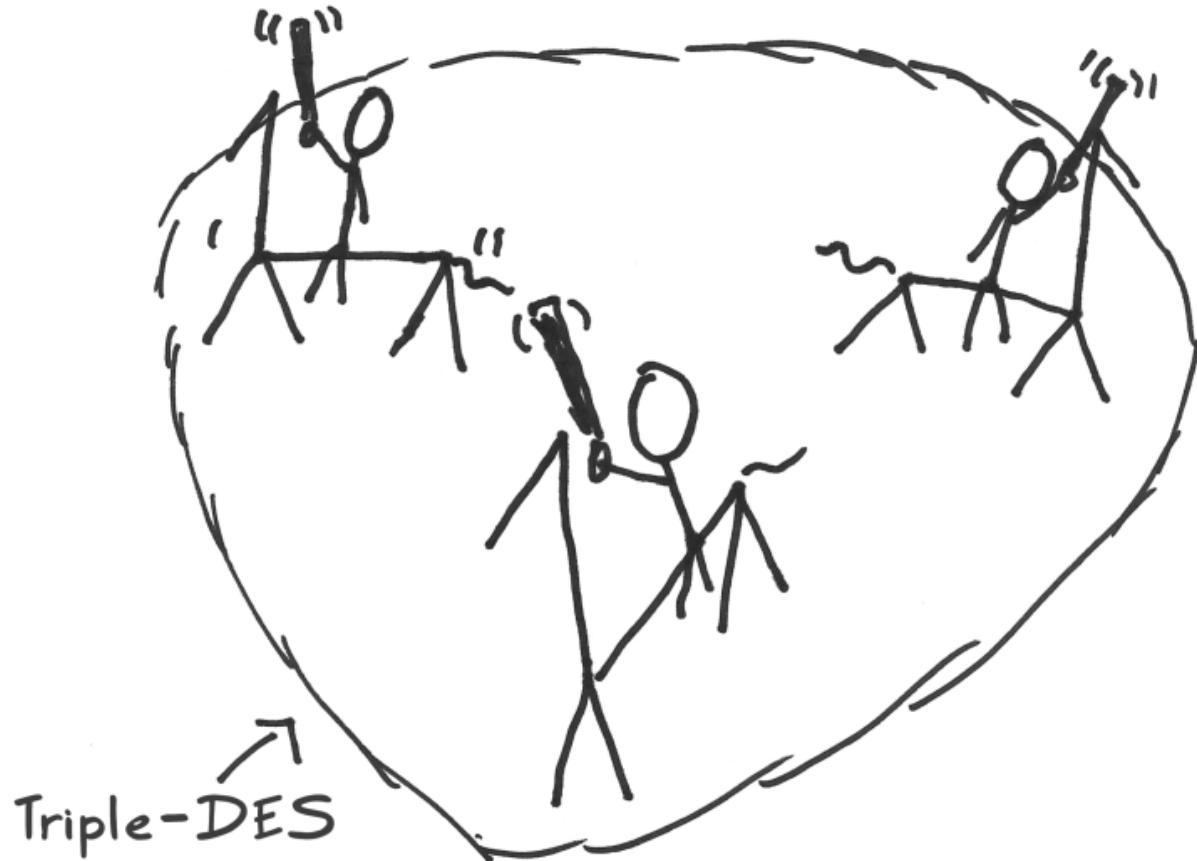
Check out that Feistel network!



Over the years, many attackers challenged DES. He was defeated in several battles.



The only way to stop the attacks was to use DES 3 times in row to form "Triple-DES." This worked, but it was awfully slow.



Another decree went out\*...



We need something at  
least as strong as  
Triple-DES, but it has  
to be fast and flexible.

NIST

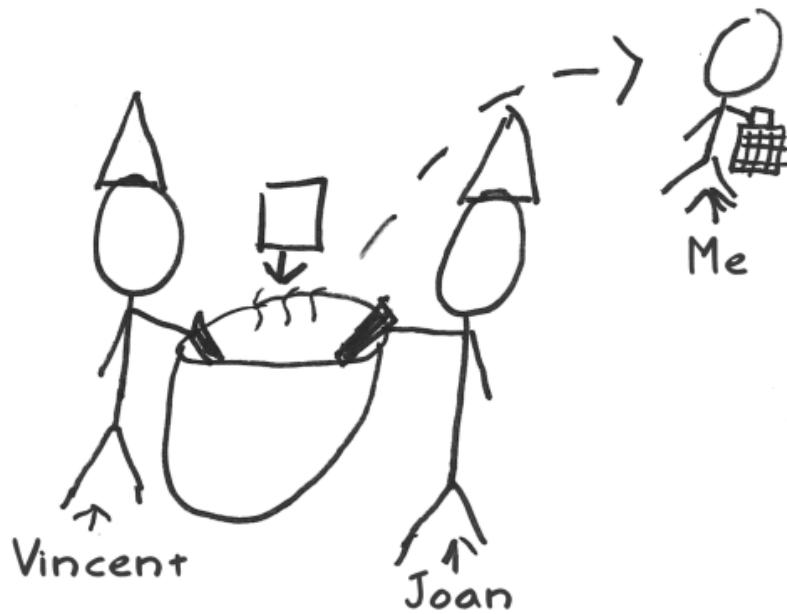
A large, empty rectangular box with a thin black border. The word "NIST" is written in a bold, sans-serif font inside the box, centered both horizontally and vertically.

\* ~early 1997

This call rallied the crypto wizards  
to develop something better.

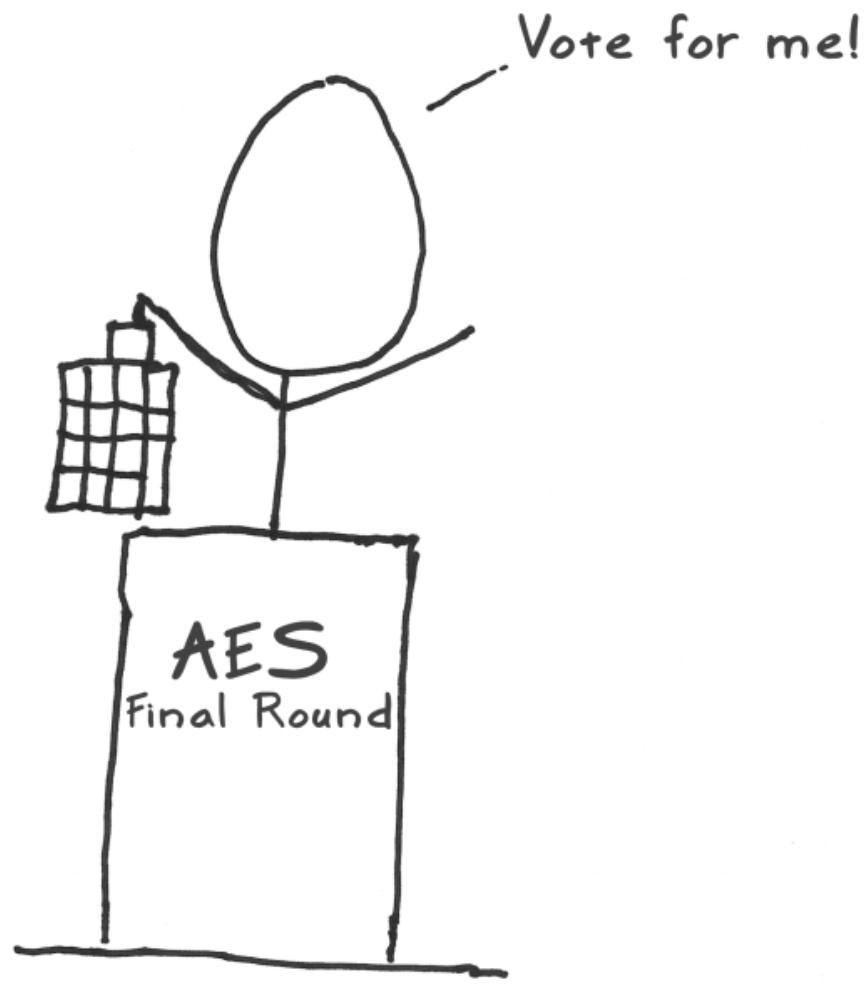


My creators, Vincent Rijmen and Joan Daemen, were among these crypto wizards. They combined their last names to give me my birth name: Rijndael.\*



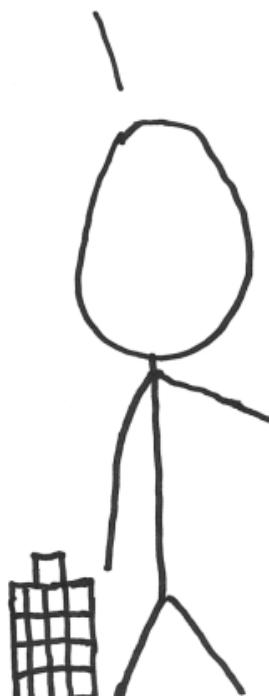
\* That's pronounced "Rhine Dahl" for the non-Belgians out there.

Everyone got together to vote and...

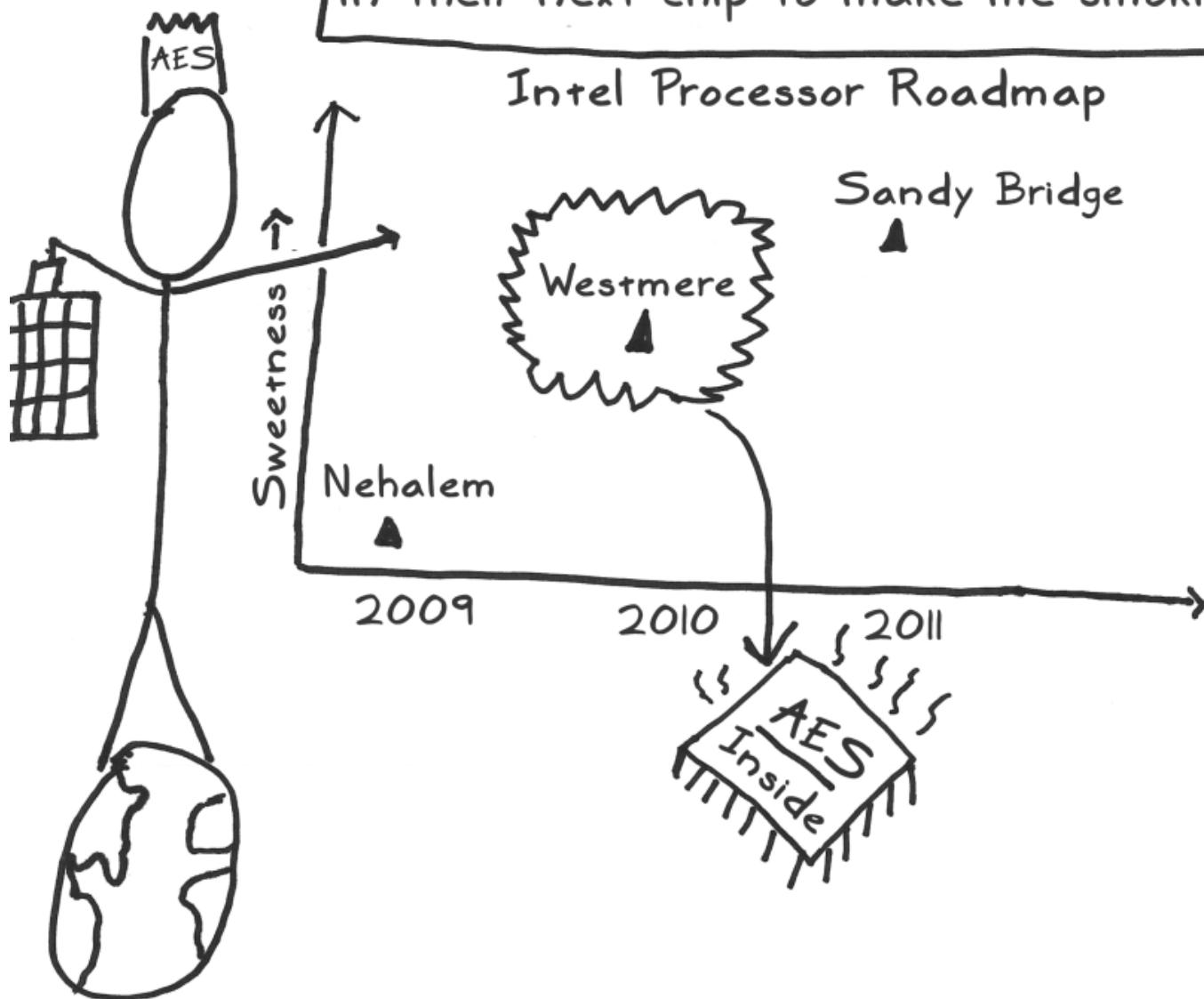


	Rijndael	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	16	14	13	10	9

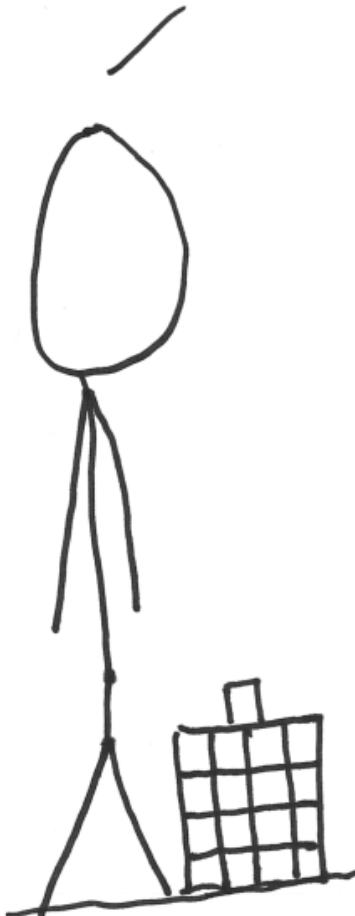
I won!!



...and now I'm the new king of the crypto world. You can find me everywhere. Intel is even putting native instructions for me in their next chip to make me smokin' fast!



Any questions?



Nice story and  
all, but how does  
crypto work?

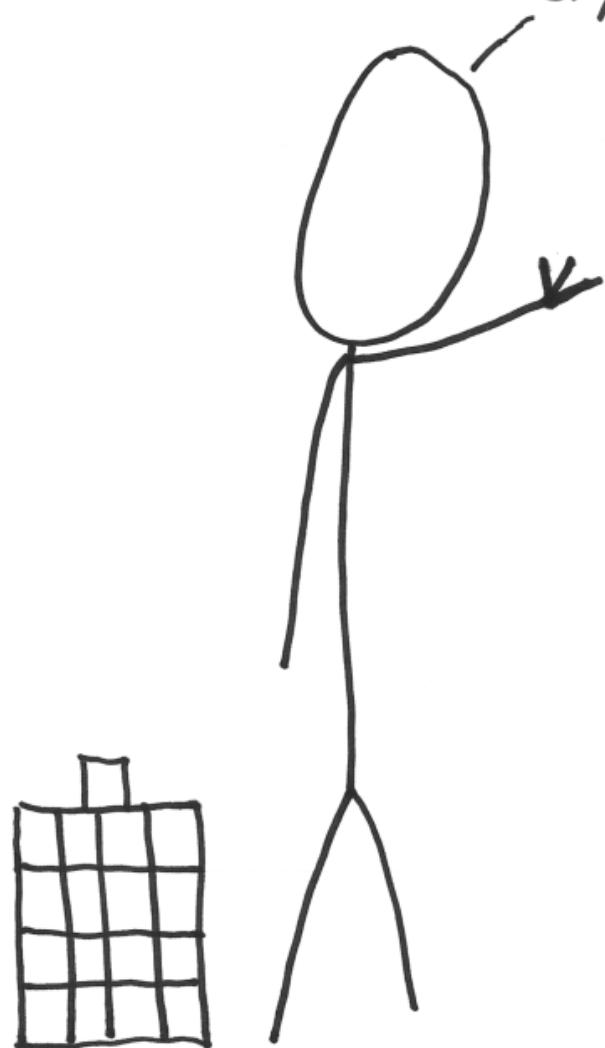


Weird. I'm out...



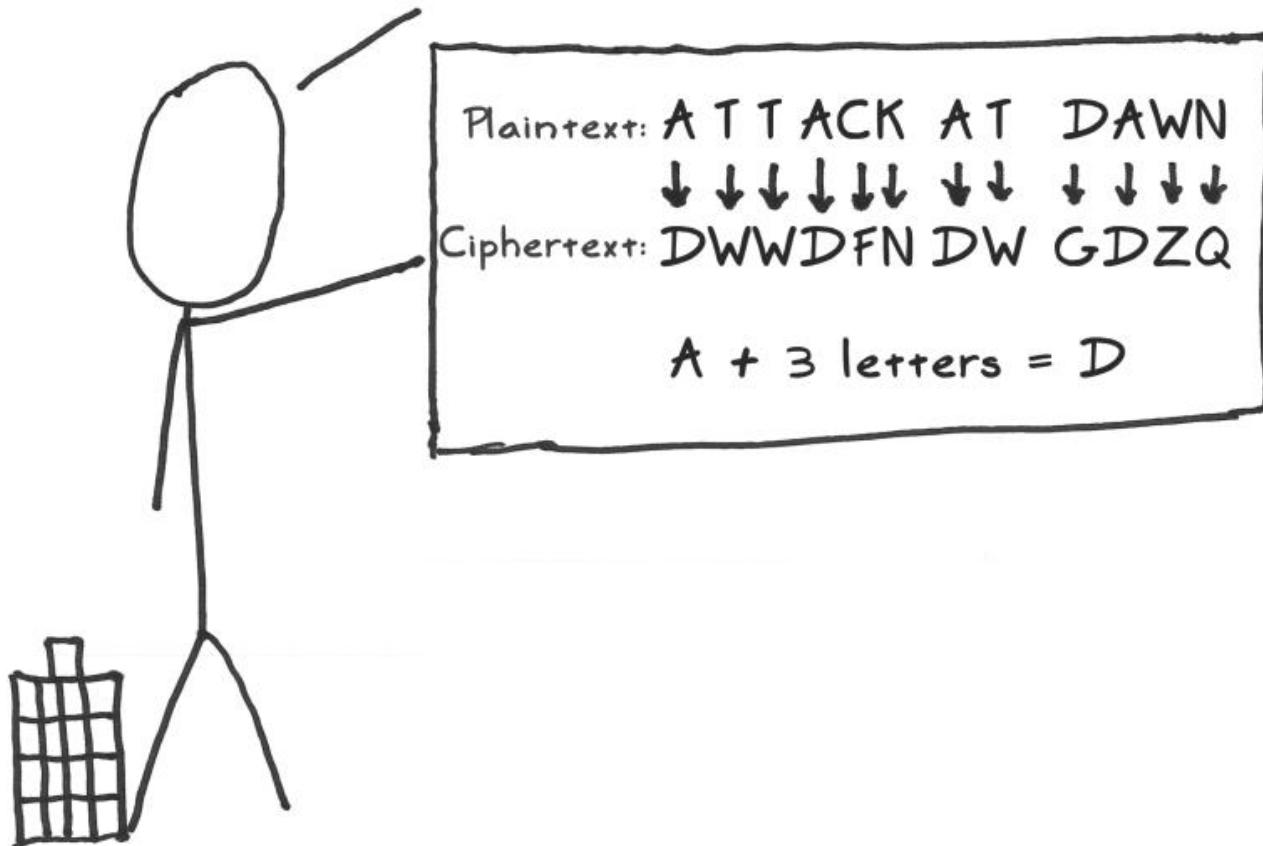
# Act 2: Crypto Basics

Great question! You only need to know 3 big ideas to understand crypto.



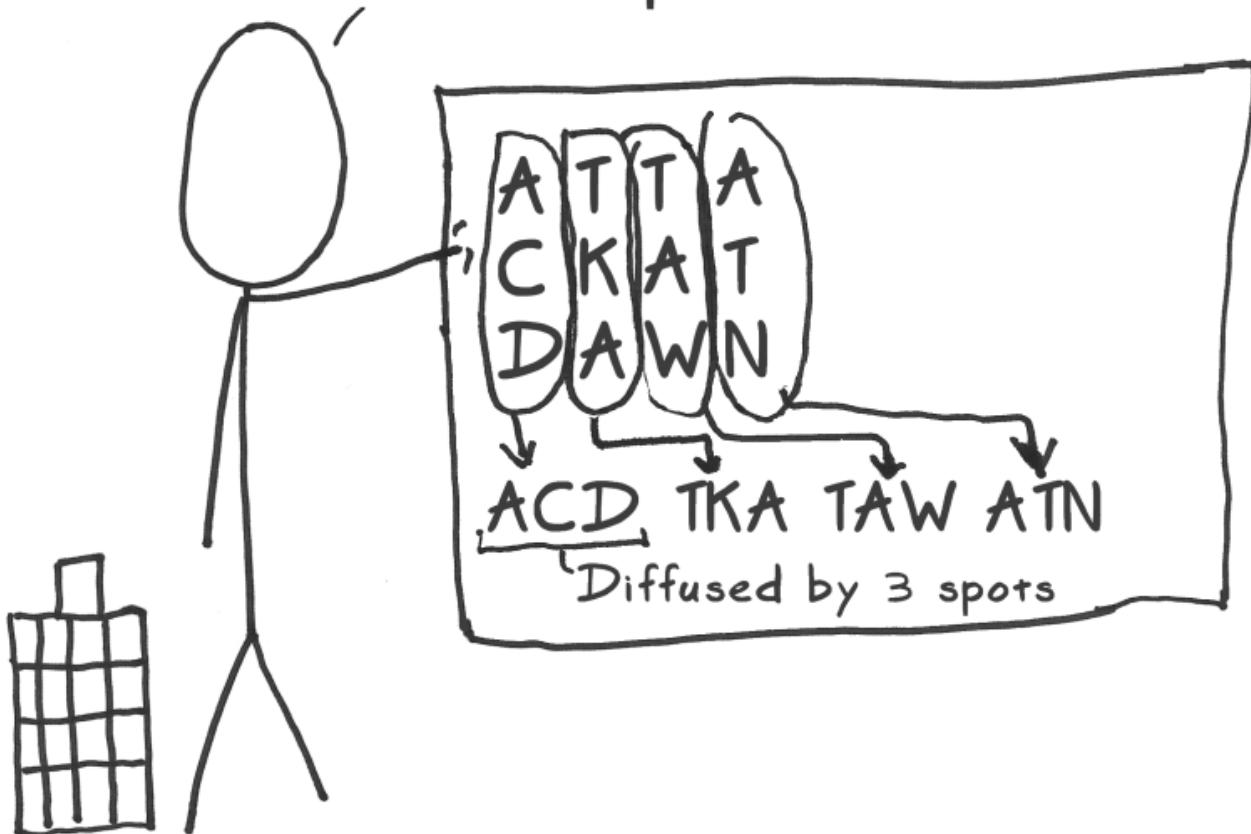
## Big Idea #1: Confusion

It's a good idea to obscure the relationship between your real message and your 'encrypted' message. An example of this "confusion" is the trusty ol' Caesar Cipher:



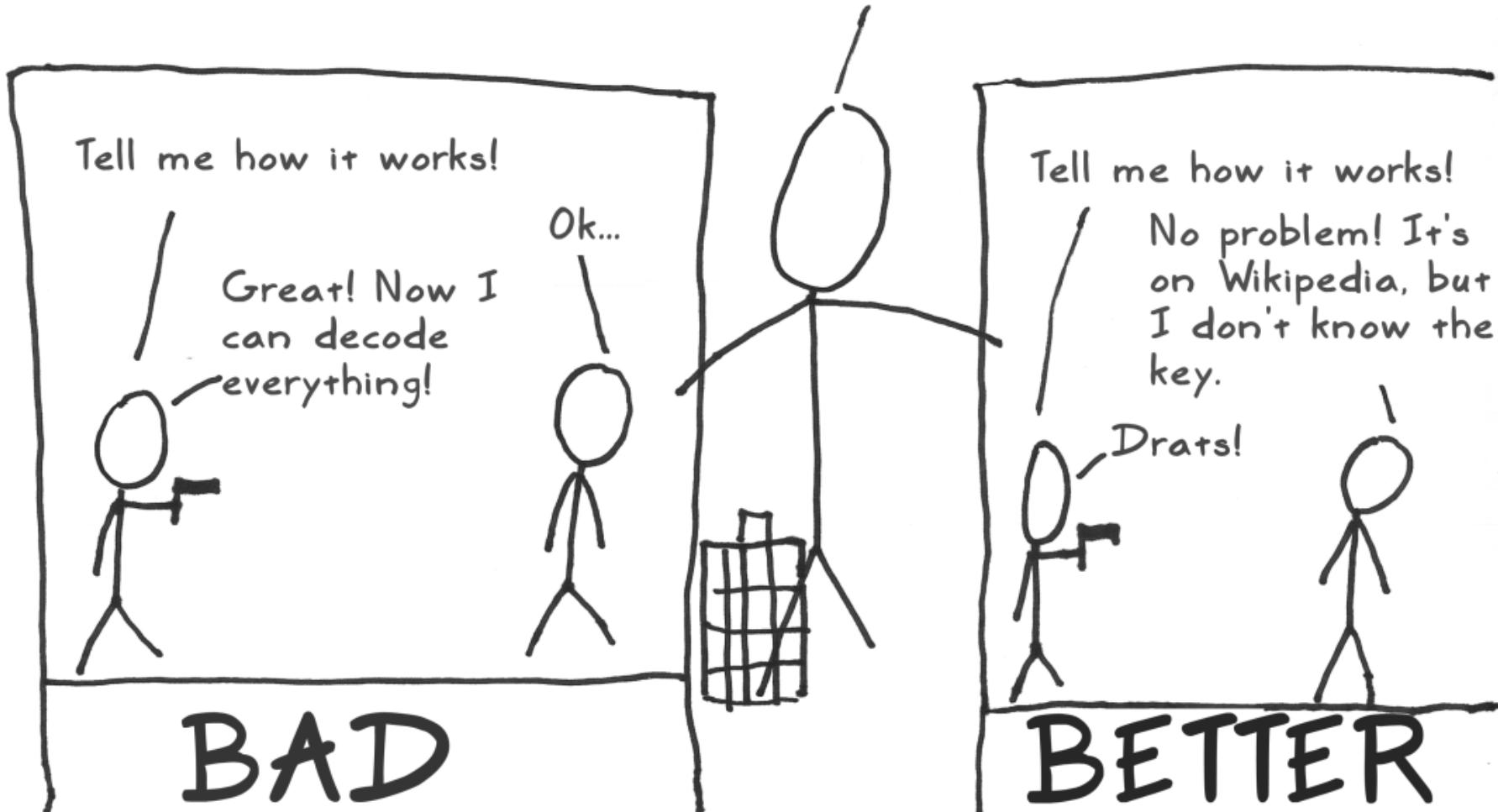
## Big Idea #2: Diffusion

It's also a good idea to spread out the message. An example of this "diffusion" is a simple column transposition:

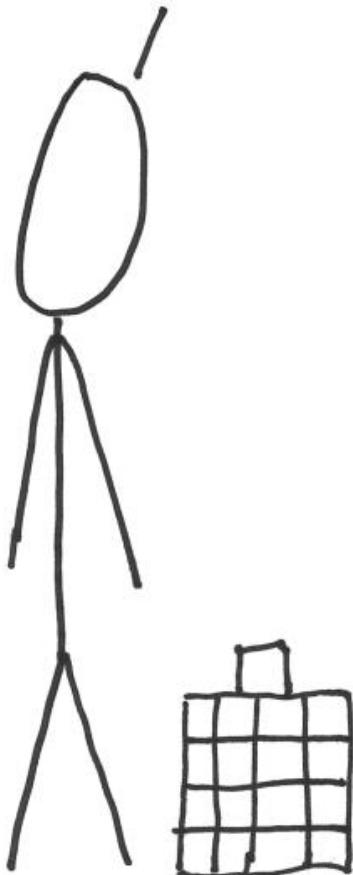


## Big Idea #3: Secrecy Only in the Key

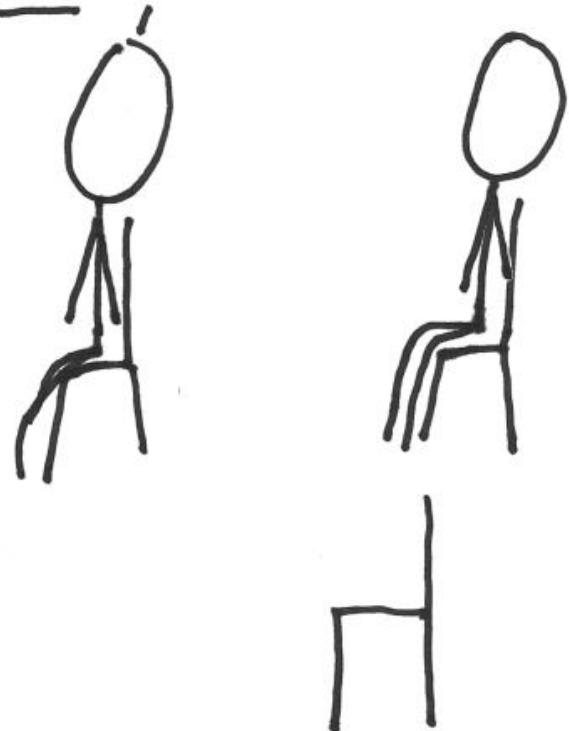
After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.



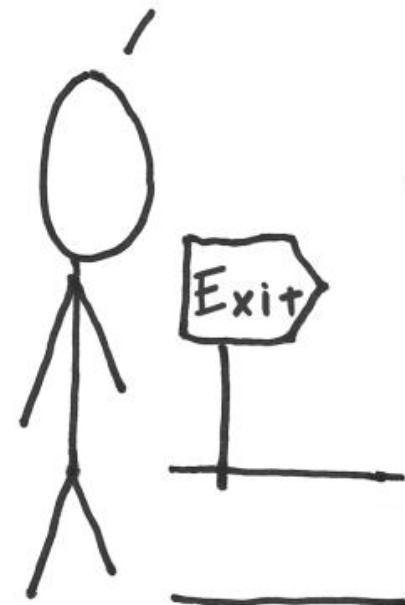
Does that answer  
your question?



That helps, but was  
too general. How do  
you work?



Details? I  
can't handle  
details!

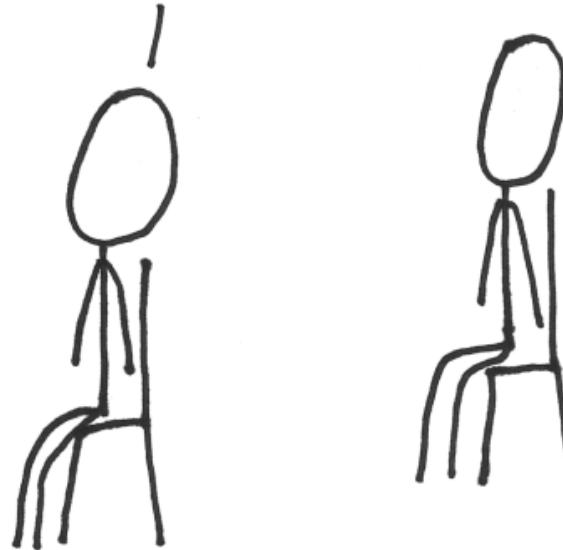


# Act 3: Details

I'd be happy to tell you  
how I work, but you have  
to sign this first.



Oh... what's that?



# Foot-Shooting Prevention Agreement

I, \_\_\_\_\_, promise that once  
Your Name

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.

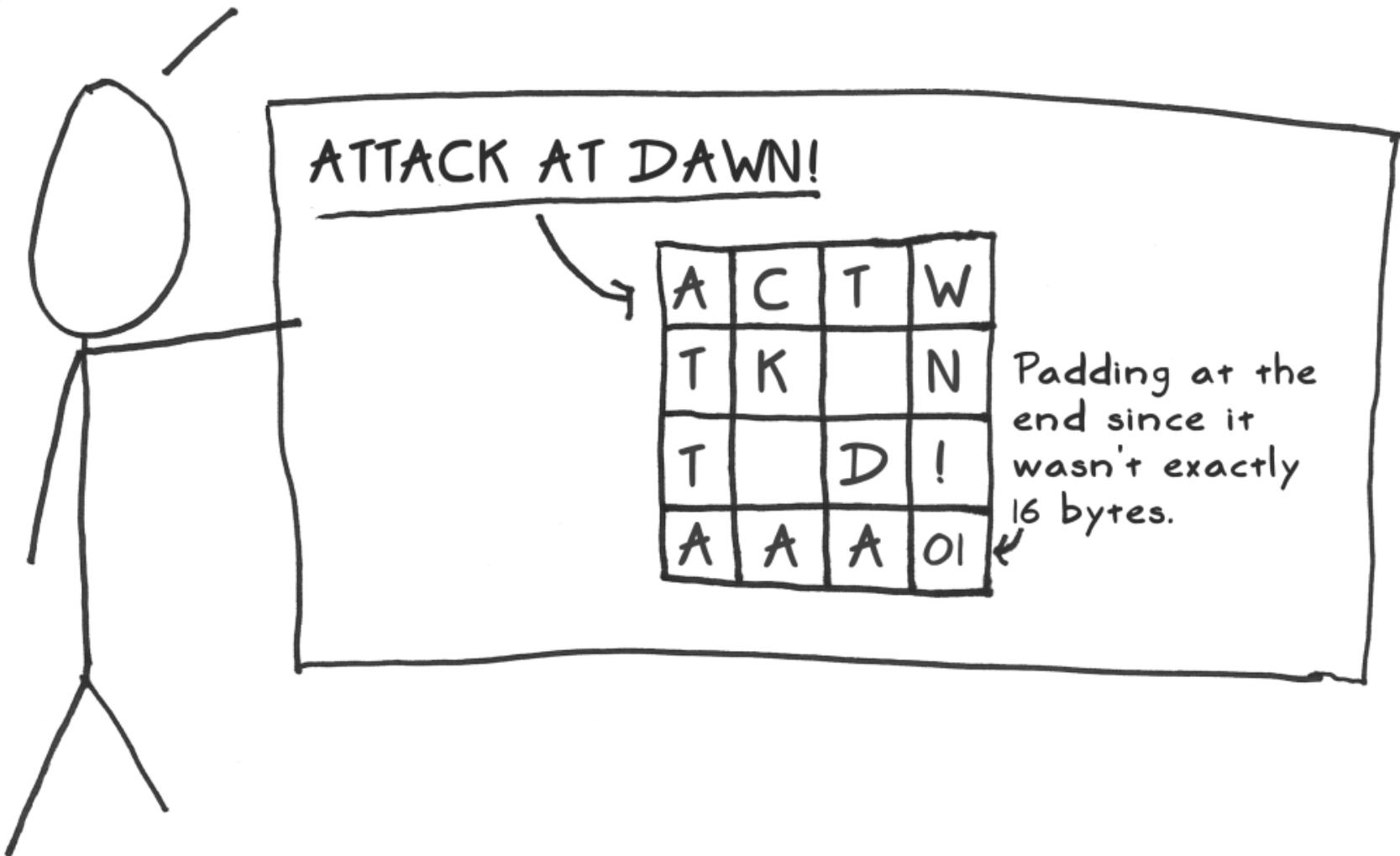
X

Signature

\_\_\_\_\_

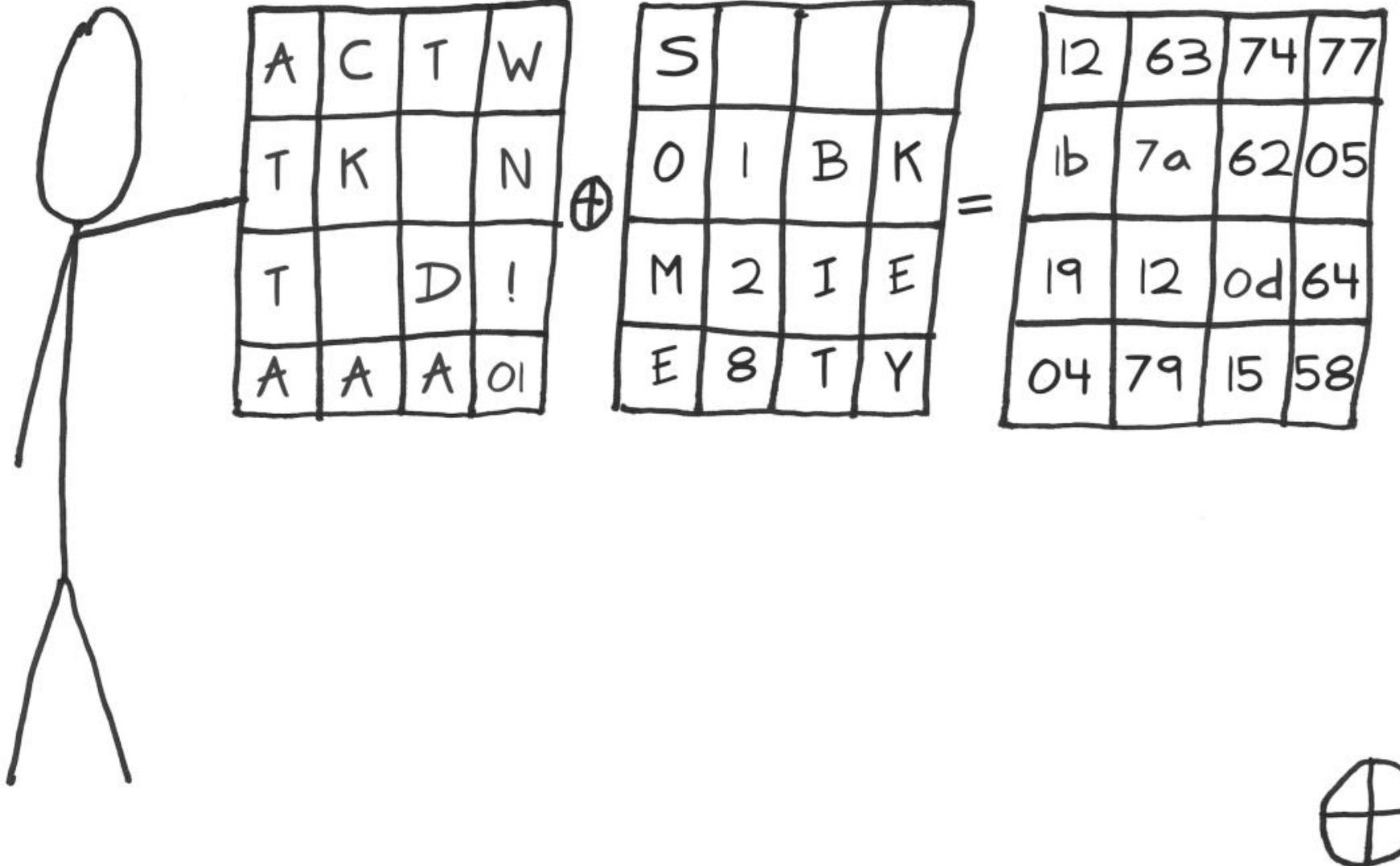
Date

I take your data and load it  
into this 4x4 square.\*



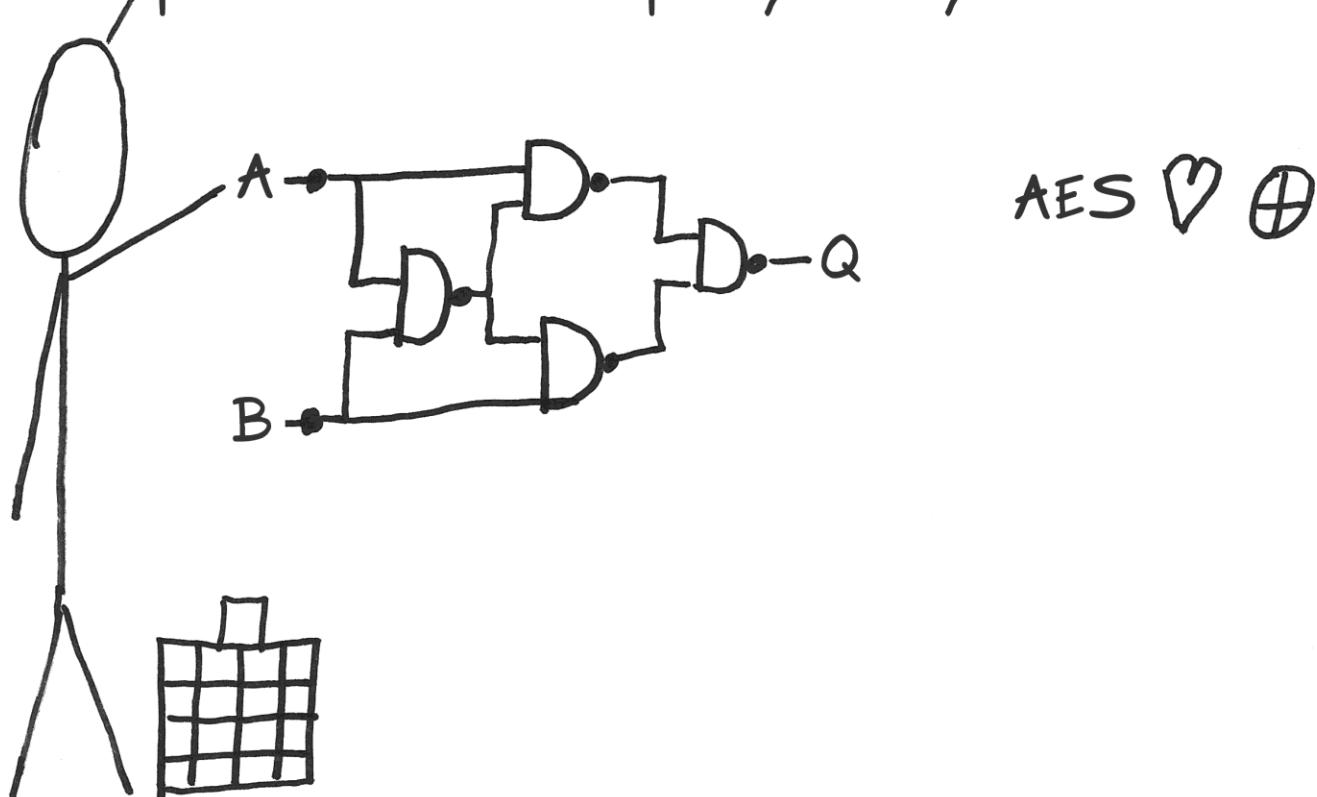
\* This is the "state matrix" that I carry with me at all times.

The initial round has me xor each input byte with the corresponding byte of the first round key.



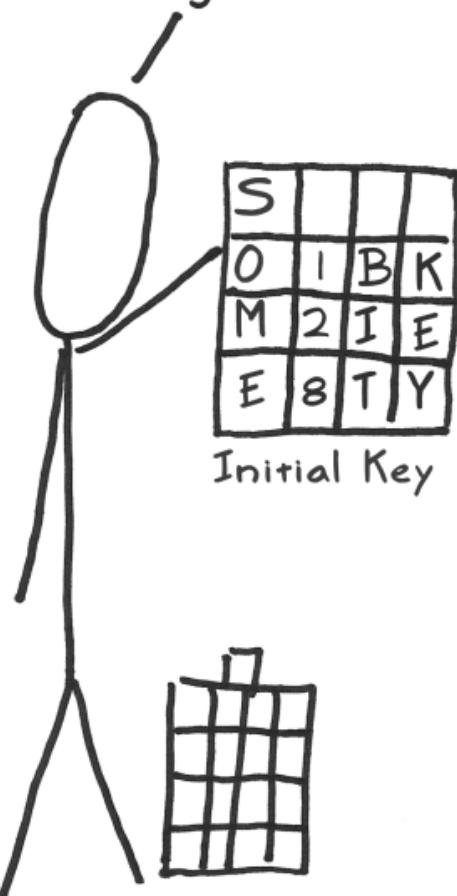
# A Tribute to XOR

There's a simple reason why I use xor to apply the key and in other spots: it's fast and cheap - a quick bit flipper. It uses minimal hardware and can be done in parallel since no pesky "carry" bits are needed.



# Key Expansion: Part 1

I need lots of keys for use in later rounds. I derive all of them from the initial key using a simple mixing technique that's really fast. Despite its critics,\* it's good enough.



S			
O	1	B	K
M	2	I	E
E	8	T	Y

Initial Key

e1	c1	e1	c1
21	10	52	19
86	b4	fd	b8
f2	ca	9e	c7

#1

...

ae	a6	a0	d4
97	d8	a6	c5
4d	7d	7a	d9
ef	ed	05	06

#9

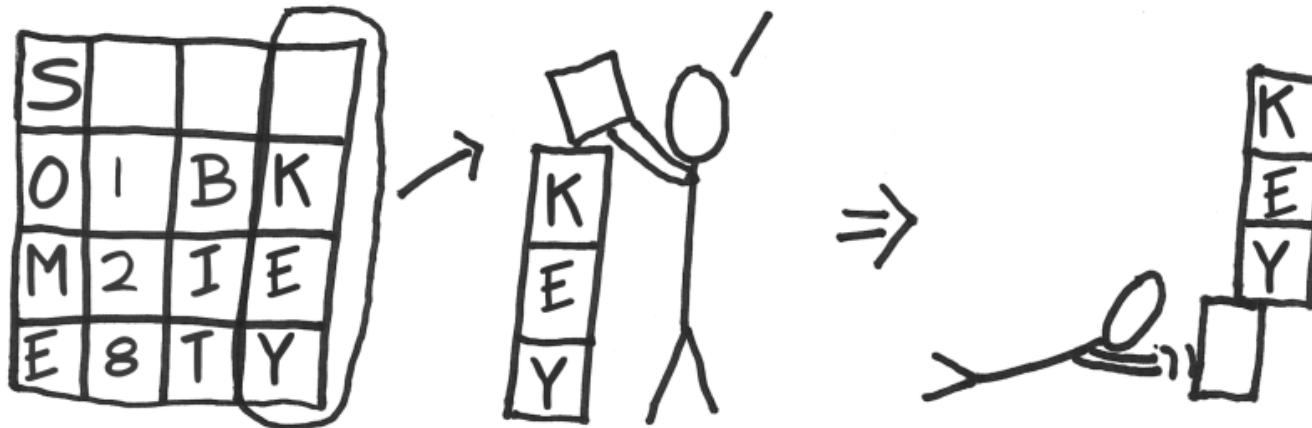
3e	98	38	ec
a2	7a	dc	19
22	5f	25	fc
a7	4a	4f	49

#10

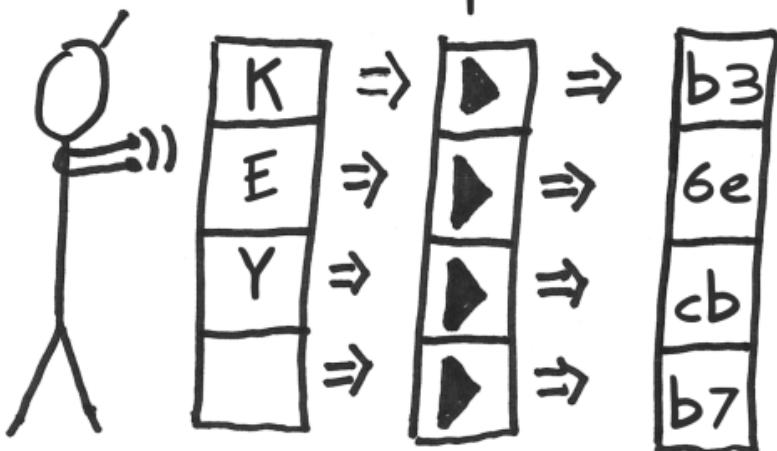
\* By far, most complaints against AES's design focus on this simplicity.

## Key Expansion: Part 2a

- ① I take the last column of the previous round key and move the top byte to the bottom:

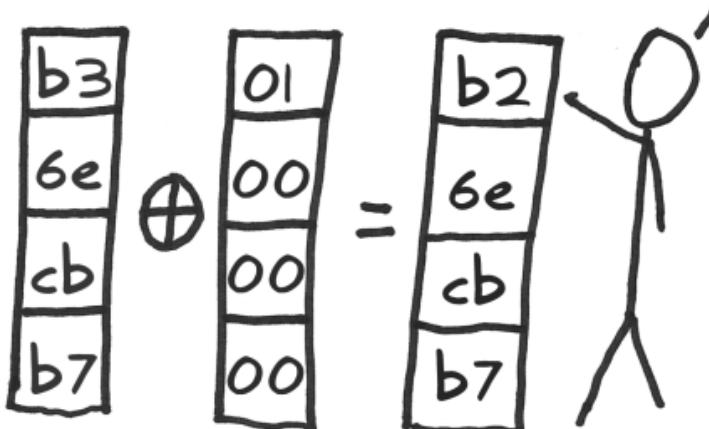


- ② Next, I run each byte through a substitution box that will map it to something else:

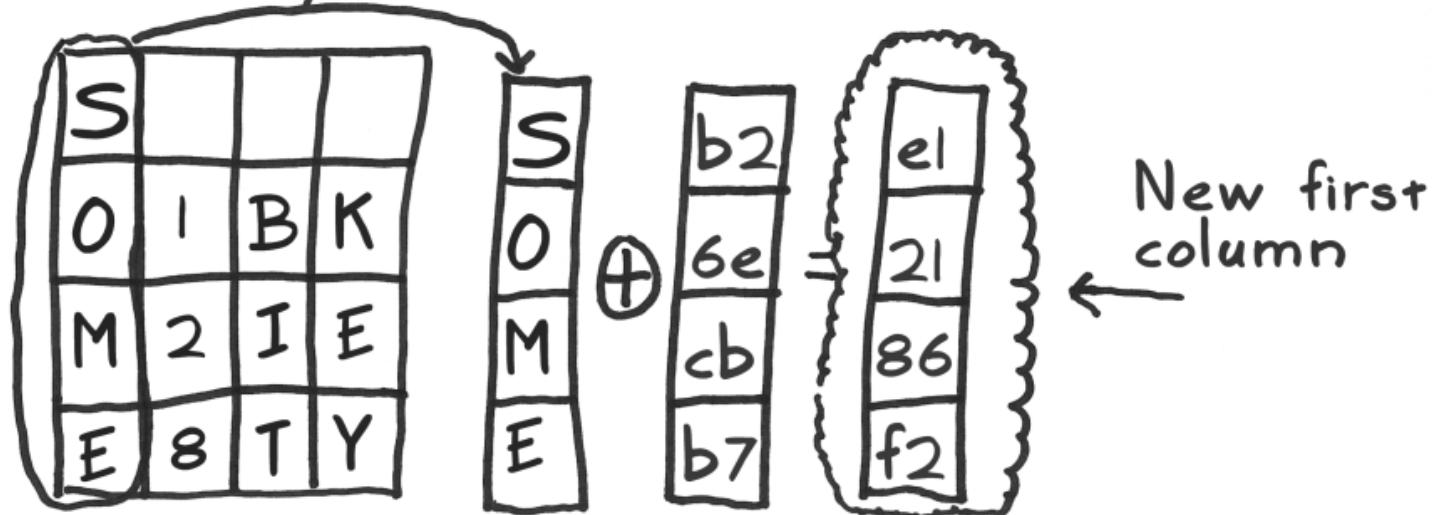


## Key Expansion: Part 2b

- ③ I then xor the column with a "round constant" that is different for each round.

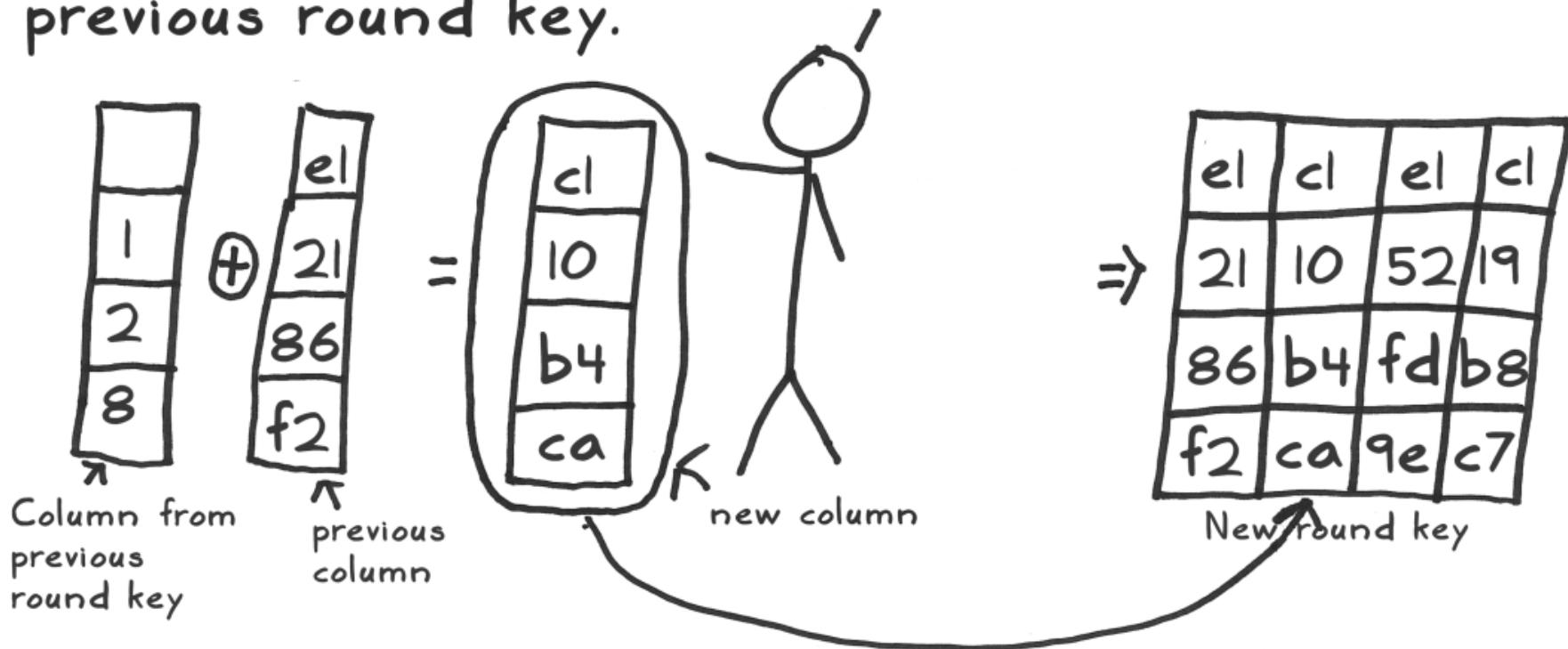


- ④ Finally, I xor it with the first column of the previous round key:



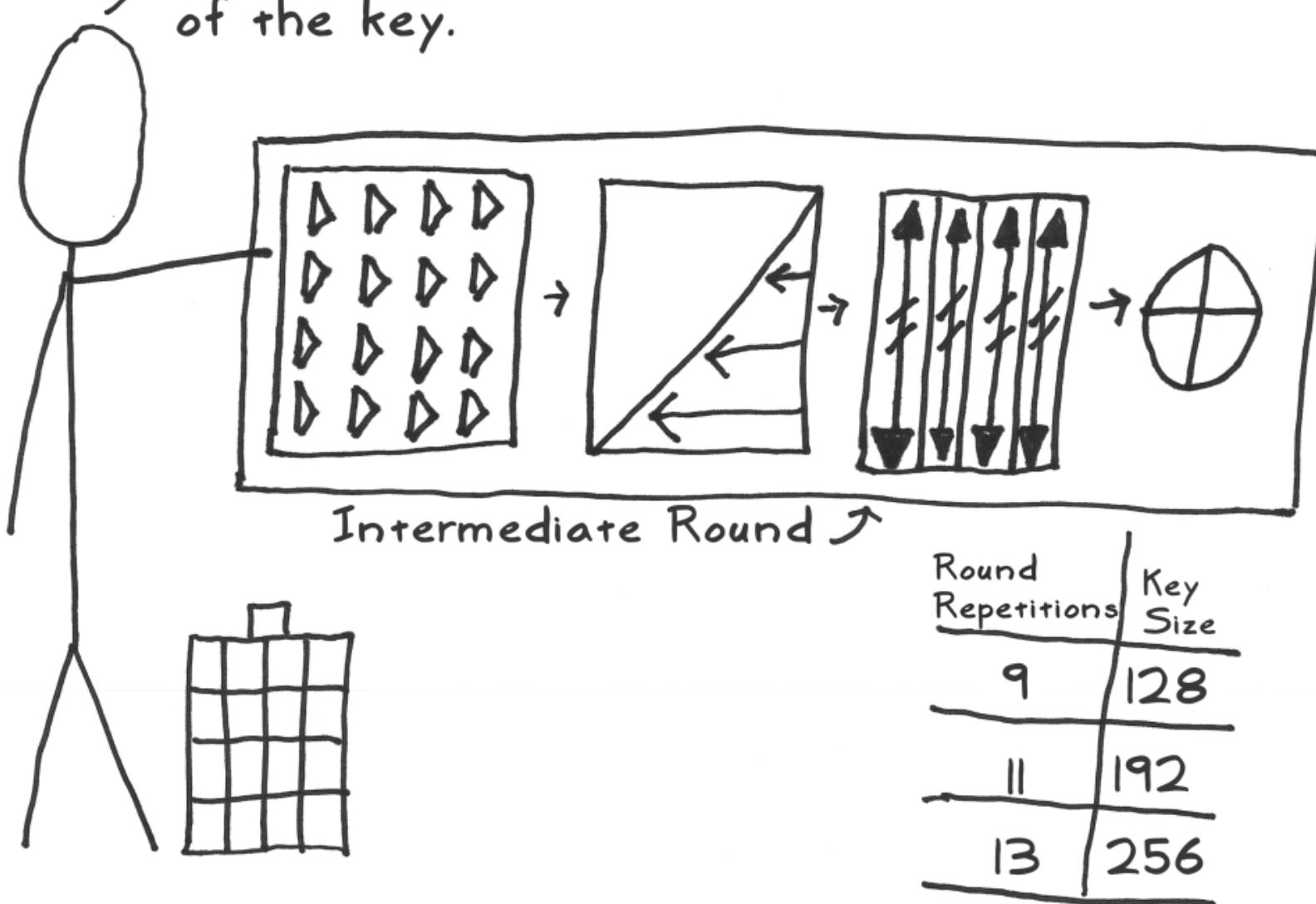
## Key Expansion: Part 3

The other columns are super-easy,\* I just xor the previous column with the same column of the previous round key.



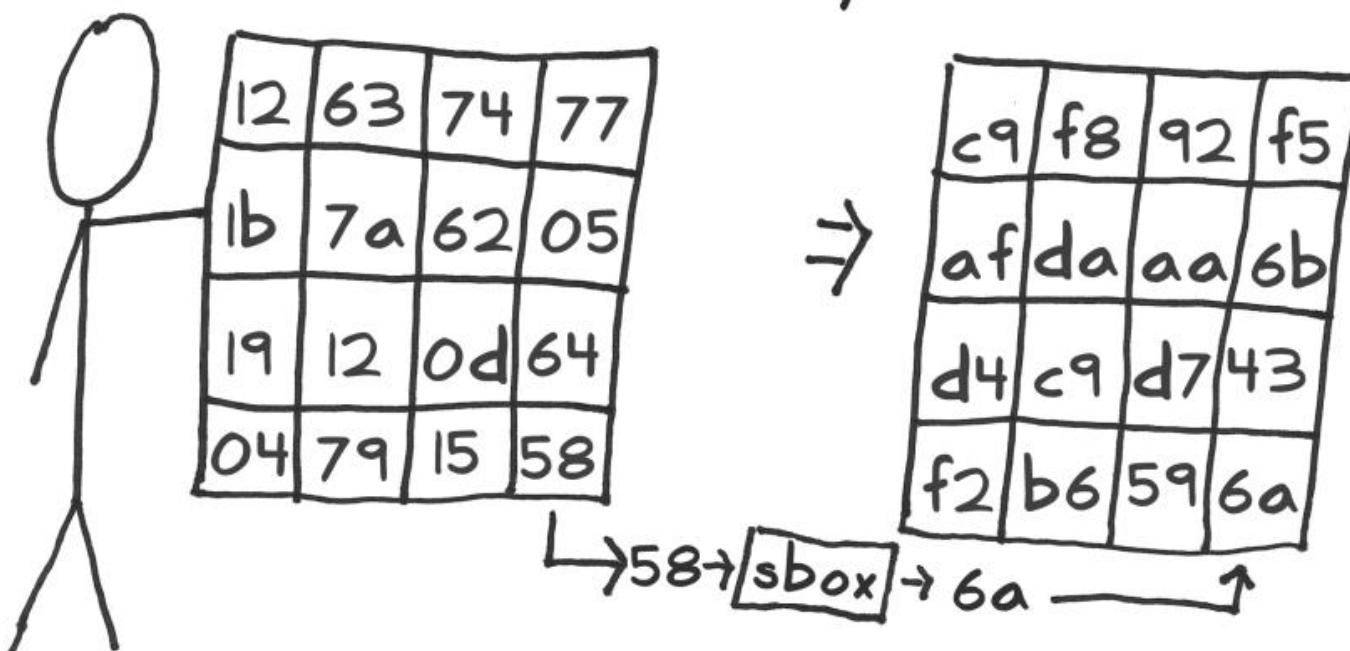
\* Note that 256 bit keys are slightly more complicated.

Next, I start the intermediate rounds. A round is just a series of steps I repeat several times. The number of repetitions depends on the size of the key.

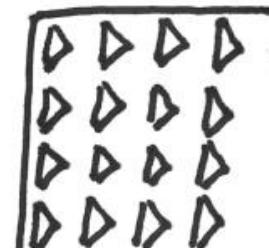


## Applying Confusion: Substitute Bytes

I use confusion (Big Idea #1) to obscure the relationship of each byte. I put each byte into a substitution box (sbox), which will map it to a different byte:

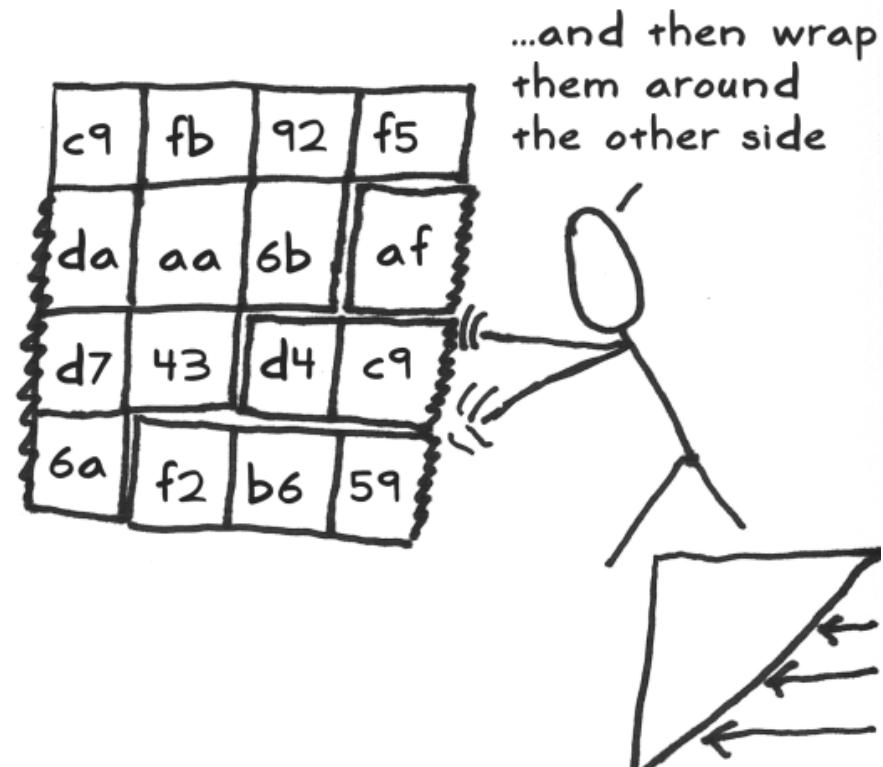
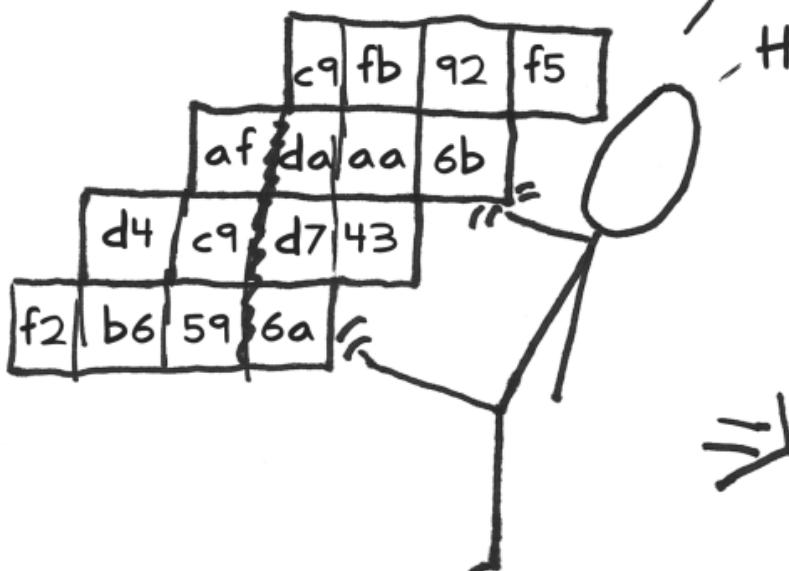


Denotes  
"confusion"



# Applying Diffusion, Part 1: Shift Rows

Next I shift the rows to the left



Denotes  
"permutation"



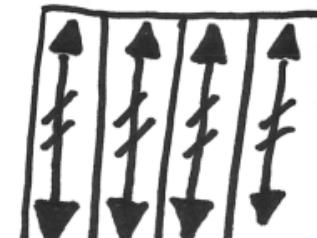
# Applying Diffusion, Part 2: Mix Columns

c9	fb	92	f5
40	a9	6b	af
d7	43	c4	c9
6a	f2	b6	59

I take each column and mix up the bits in it.



41	b9	e0	8b
6e	83	95	a9
18	da	8b	38
99	00	65	do



# Applying Key Secrecy: Add Round Key

At the end of each round, I apply the next round key with an xor:

41	b9	e0	8b
6e	83	95	a9
18	da	8b	38
99	00	65	d0



e1	c1	e1	c1
21	10	52	19
86	b4	fd	b8
f2	ca	9e	c7

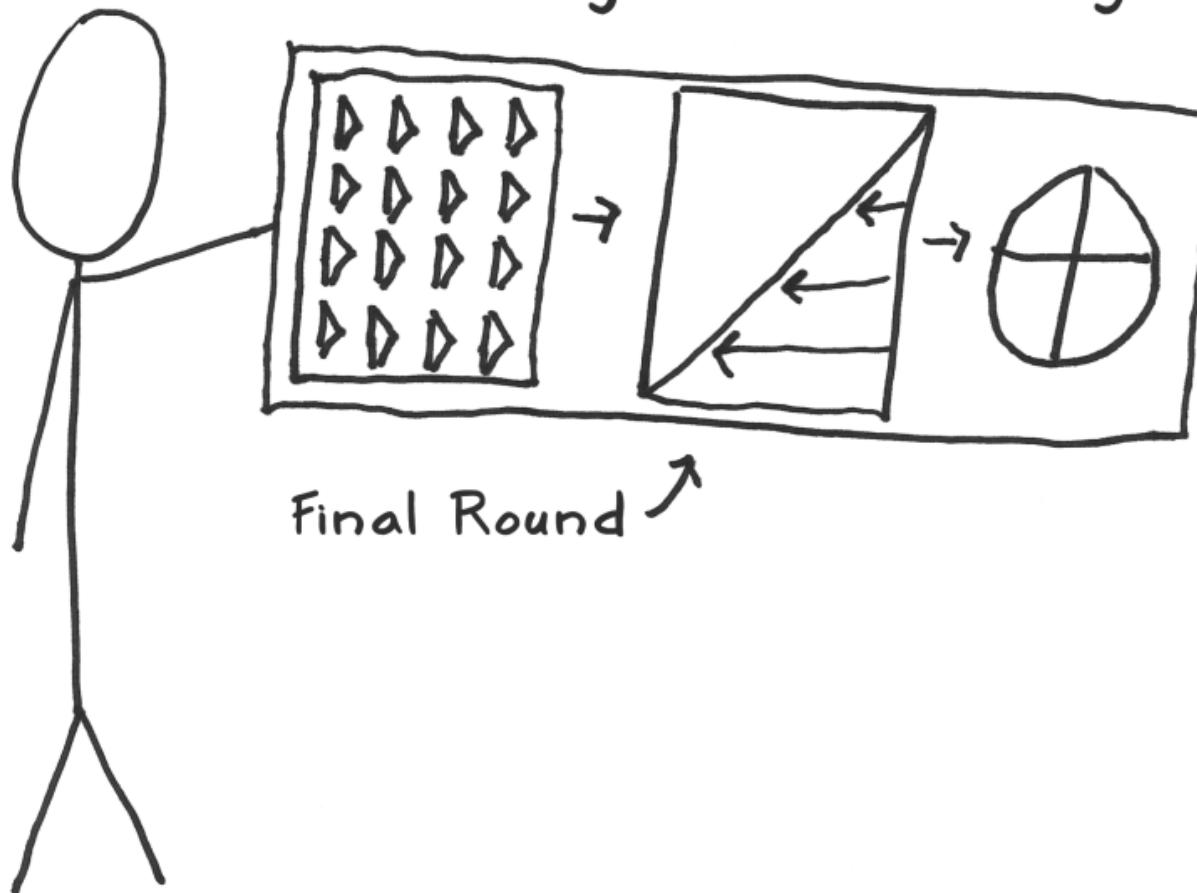


a0	78	01	4a
4f	93	c7	b0
9e	6e	76	80
6b	ca	fb	17

$$d0 \oplus c7 = 17$$

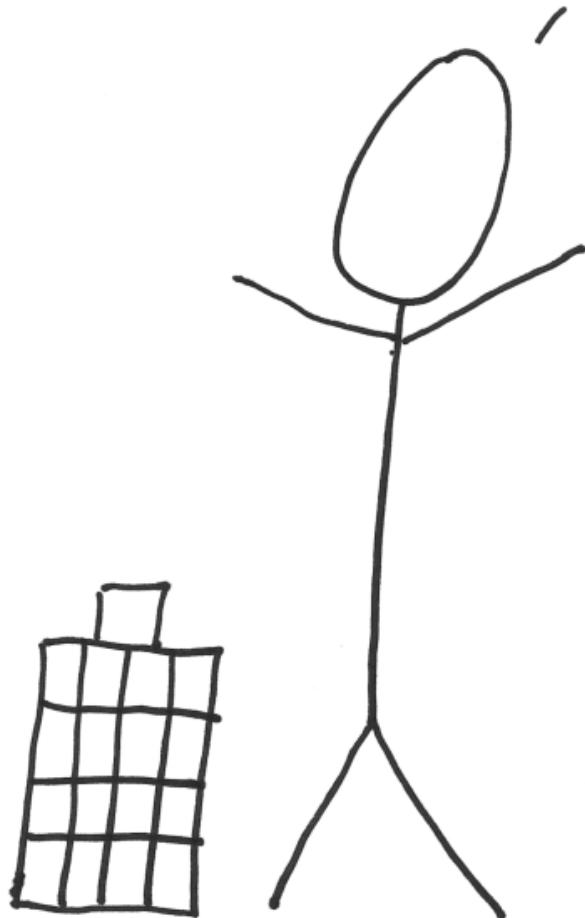


In the final round, I skip the "Mix Columns" step since it wouldn't increase security\* and would just slow things down:

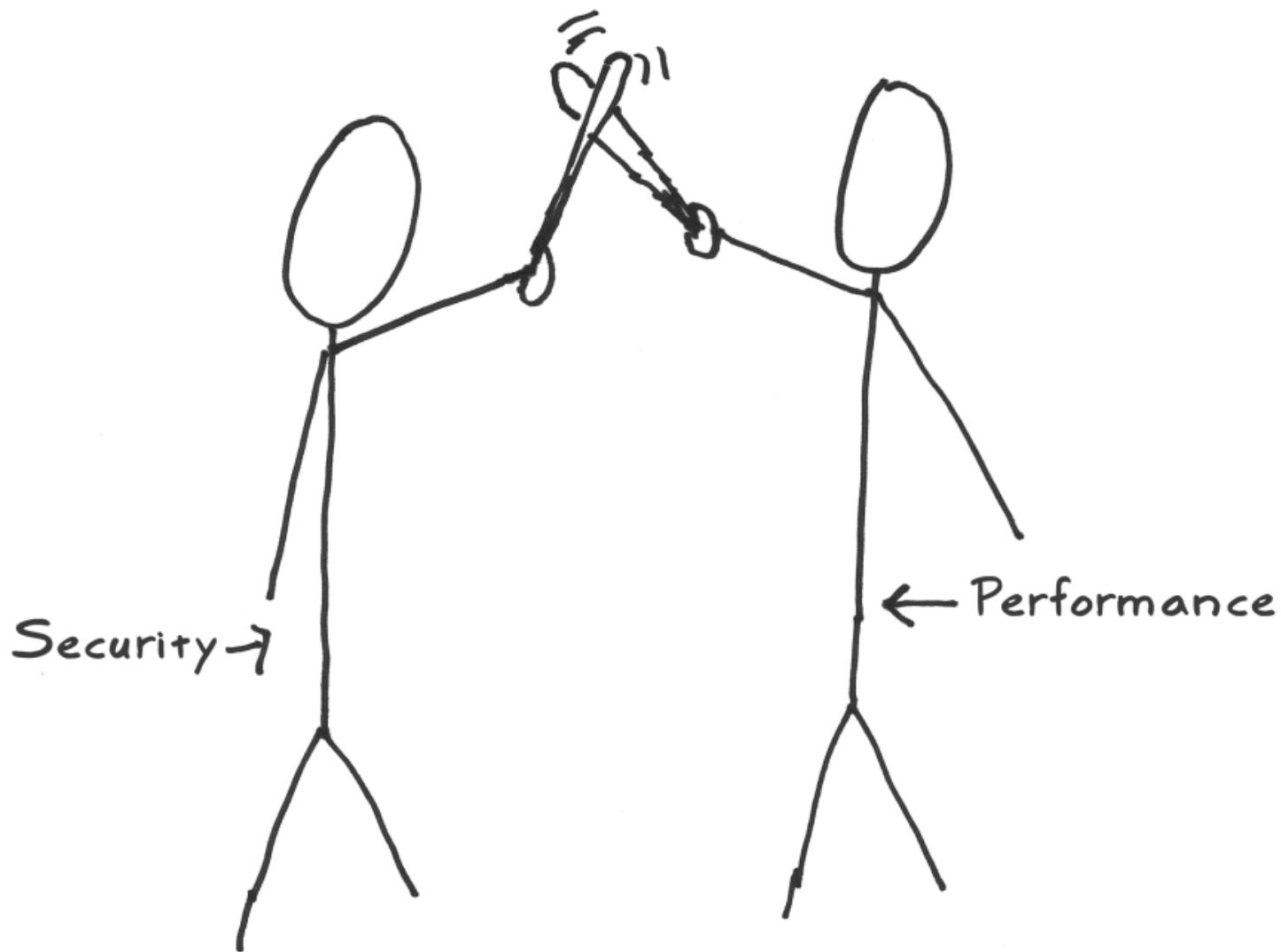


\*The diffusion it would provide wouldn't go to the next round.

...and that's it. Each round I do makes the bits more confused and diffused. It also has the key impact them. The more rounds, the merrier!

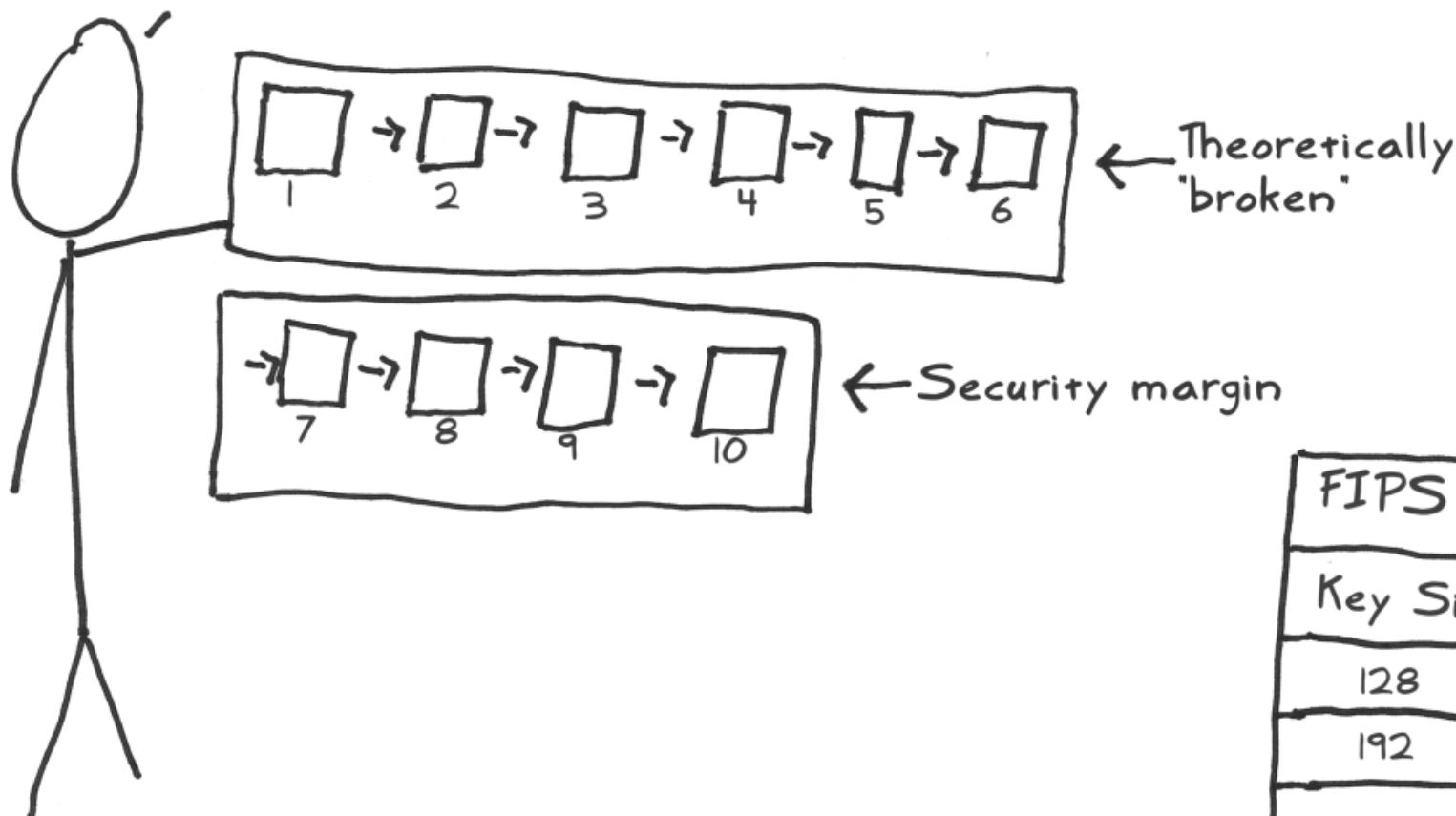


Determining the number of rounds always involves several tradeoffs.



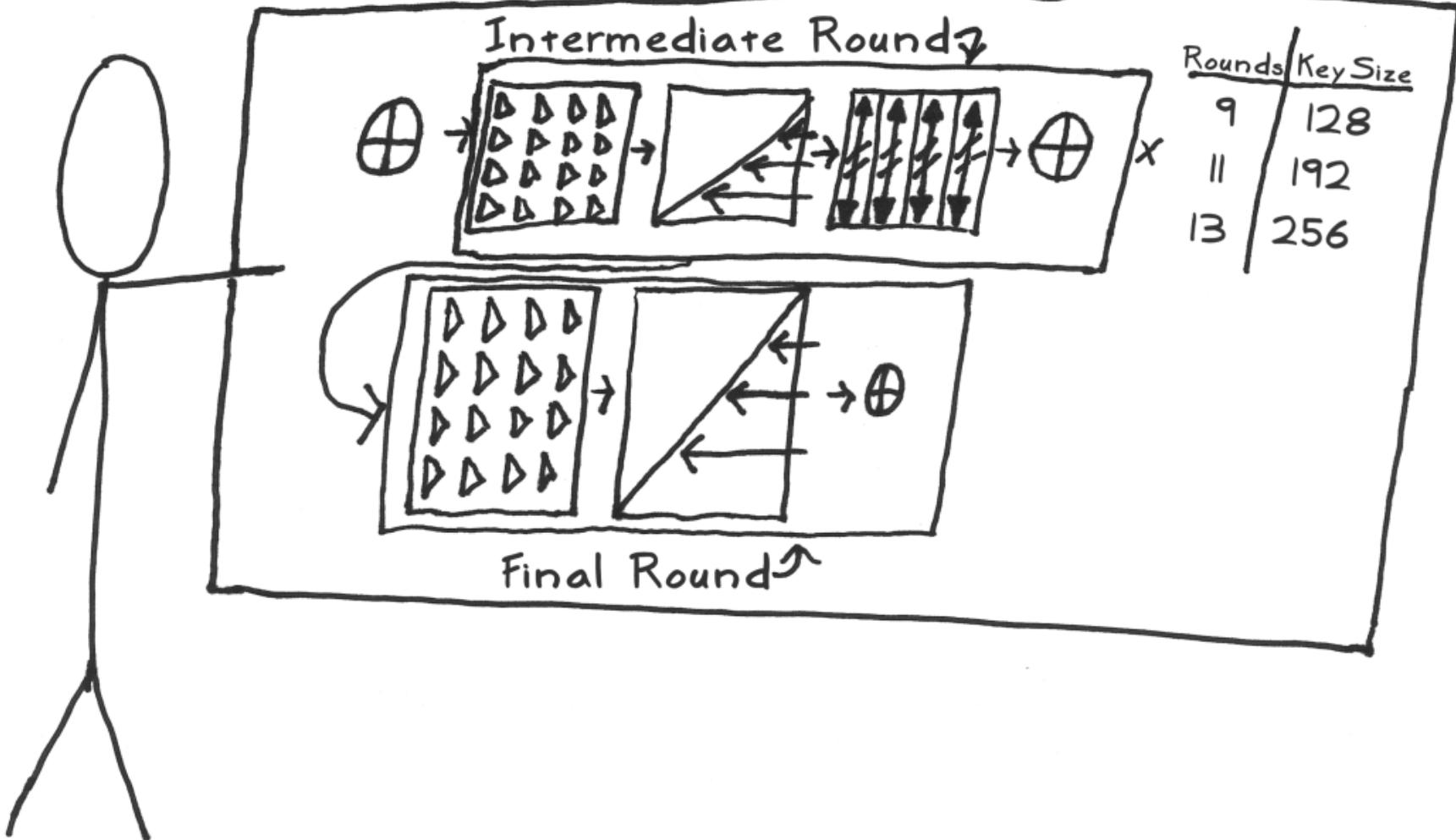
"Security always comes at a cost to performance" - Vincent Rijmen

When I was being developed, a clever guy was able to find a shortcut path through 6 rounds. That's not good! If you look carefully, you'll see that each bit of a round's output depends on every bit from two rounds ago. To increase this diffusion "avalanche," I added 4 extra rounds. This is my "security margin."

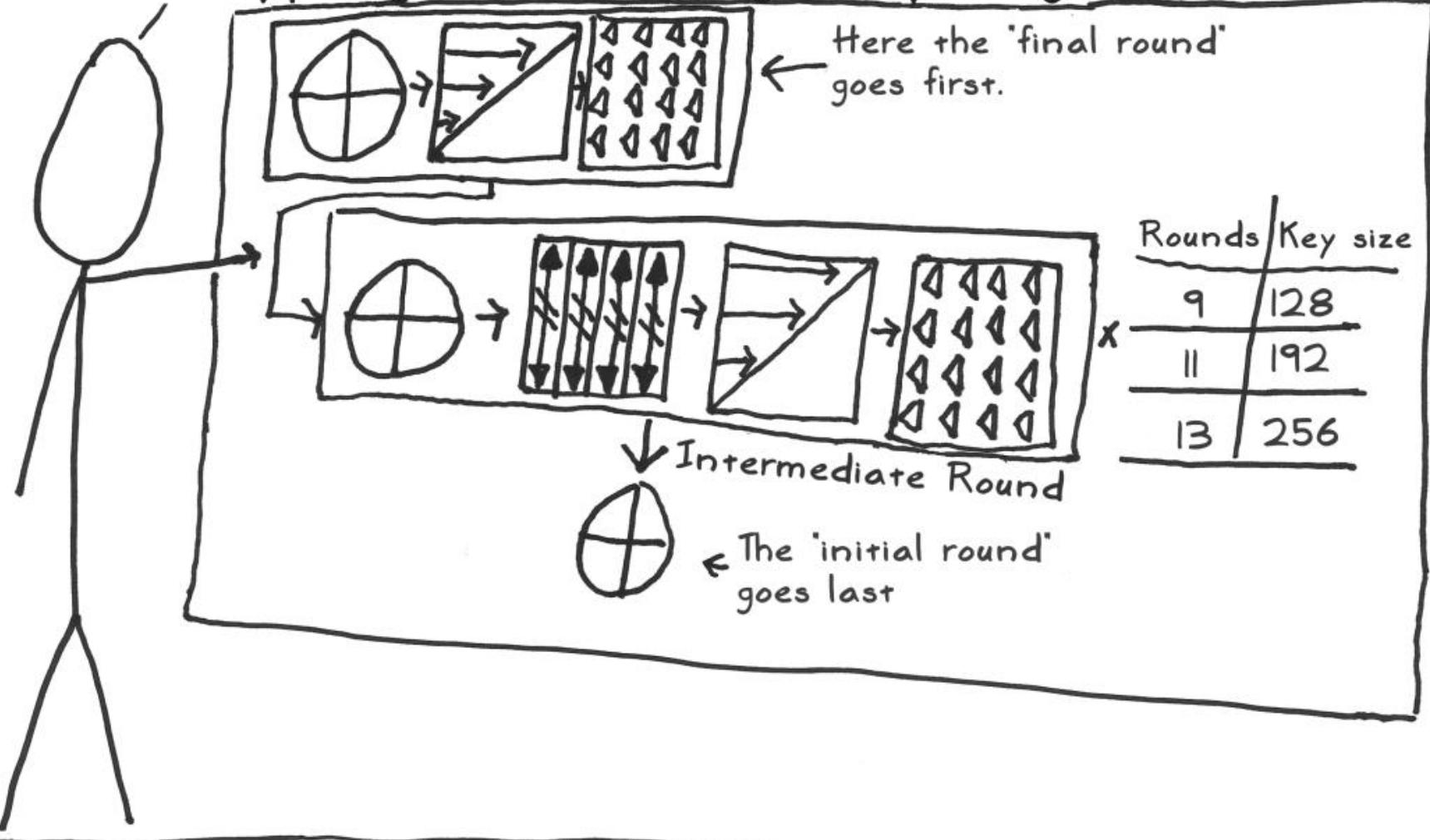


FIPS 197 Spec	
Key Size	Rounds
128	10
192	12
256	14

So in pictures, we have this:



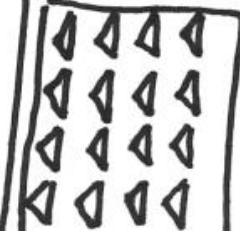
# Decrypting means doing everything in reverse



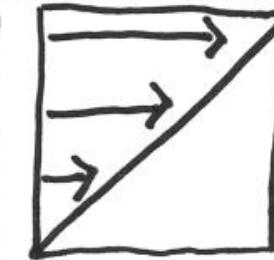
Add Round Key Inverse



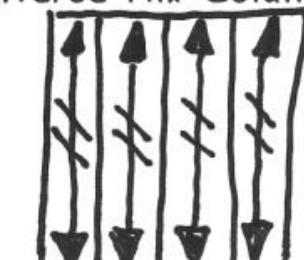
Inverse Substitute Bytes



Inverse Shift Rows

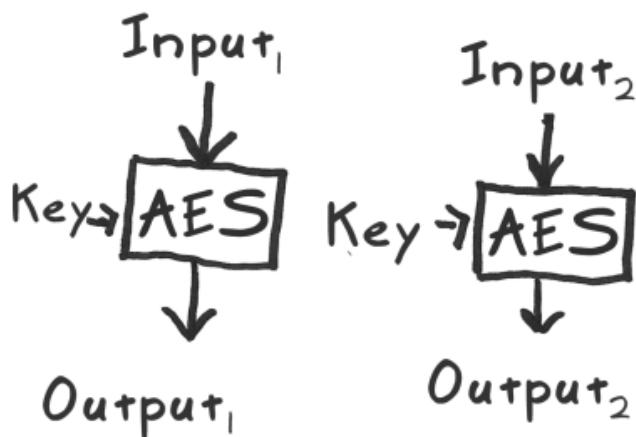


Inverse Mix Columns



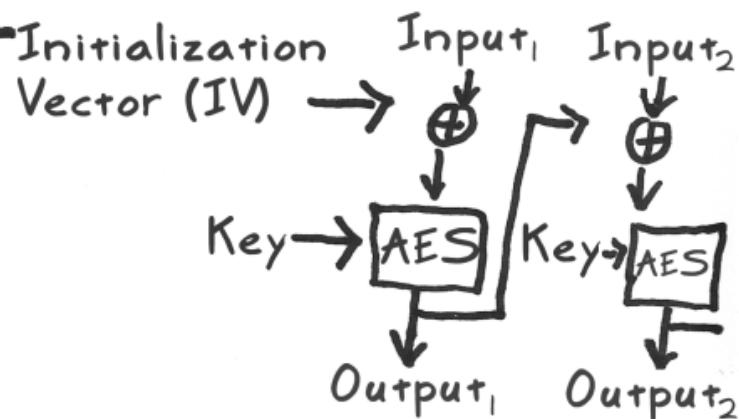
One last tidbit: I shouldn't be used as-is, but rather as a building block to a decent "mode."

### Electronic Codebook Mode (ECB)



BAD!

### Cipher-block Chaining (CBC)

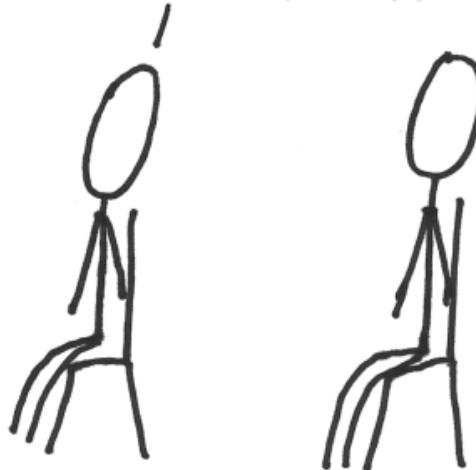


Better

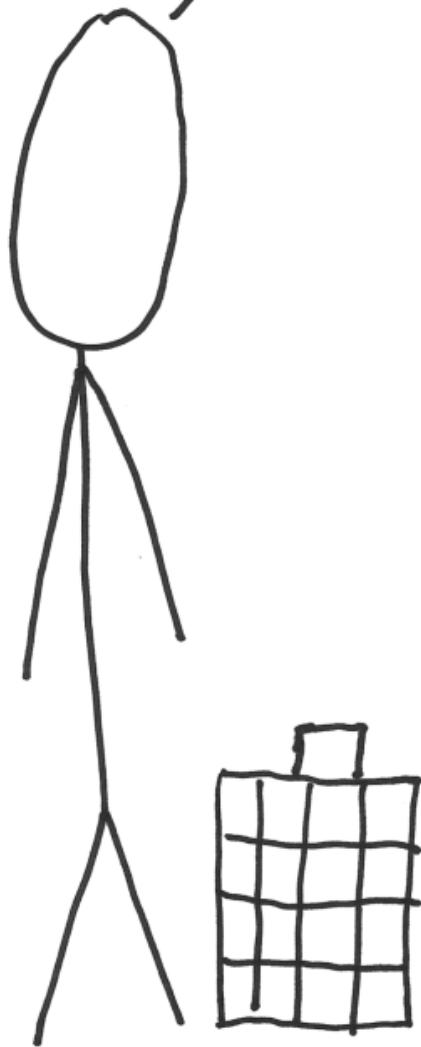
Make sense? Did that  
answer your question?



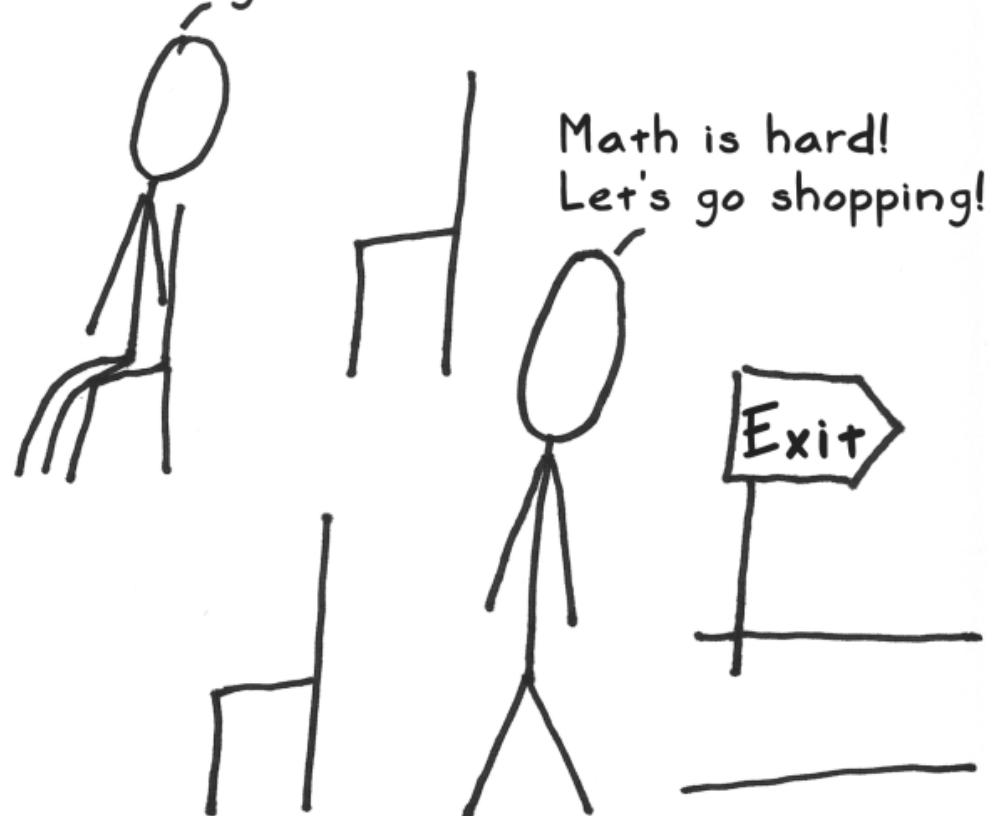
Almost...except you just  
waved your hands and  
used weird analogies.  
What really happens?



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!

# Act 4: Math!

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

Come on  
class, what's  
the answer?

$$X + X = ?$$



I know!  
It's  $2x$

I should  
copy off  
him...

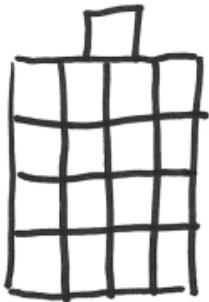
You

Will Ashley  
go out  
with me?

# Reviewing the Basics...



$$\begin{aligned} & \text{square} && \text{multiplication} && \text{polynomial} \\ & (x+1)^2 = (x+1) \cdot (x+1) = x^2 + x + x + 1 = x^2 + 2x + 1 && && \\ & \text{the unknown} && \text{addition} && \text{degree} \\ & && && \text{coefficient} \end{aligned}$$



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

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

$$= x^2 \oplus 1 \quad \text{The 'new' add*}$$

Small coefficients

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

$$\begin{aligned} &= 0 \oplus x^2 \\ &= x^2 \end{aligned}$$

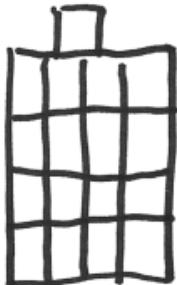
\*Nifty Fact: In the new way, addition is the same as subtraction (e.g.  $x \oplus x = x - x = 0$ )

Remember how multiplication could make things grow fast?



$$\begin{aligned} & (x^7 + x^5 + x^3 + x) \cdot (x^6 + x^4 + x^2 + 1) \\ &= x^{7+6} + x^{7+4} + x^{7+2} + x^{7+0} + x^{5+6} + x^{5+4} + x^{5+2} + x^{5+0} \\ &\quad + x^{3+6} + x^{3+4} + x^{3+2} + x^{3+0} + x^{1+6} + x^{1+4} + x^{1+2} + x^{1+0} \\ &= x^{13} + x^{11} + x^9 + x^7 + x^{10} + x^9 + x^7 + x^5 + x^9 + x^7 + x^5 + x^3 + x^7 + x^5 + x^3 + x \\ &= x^{13} + x^{11} + x^{10} + x^9 + x^9 + x^7 + x^7 + x^7 + x^7 + x^5 + x^5 + x^5 + x^3 + x^3 + x \\ &= x^{13} + 2x^{11} + 3x^9 + 4x^7 + 3x^5 + 2x^3 + x \end{aligned}$$

↗  
Big and yucky!

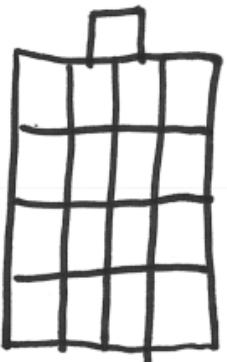


With the "new" addition, things are simpler, but the  $x^{13}$  is still too big. Let's make it so we can't go bigger than  $x^7$ . How can we do that?

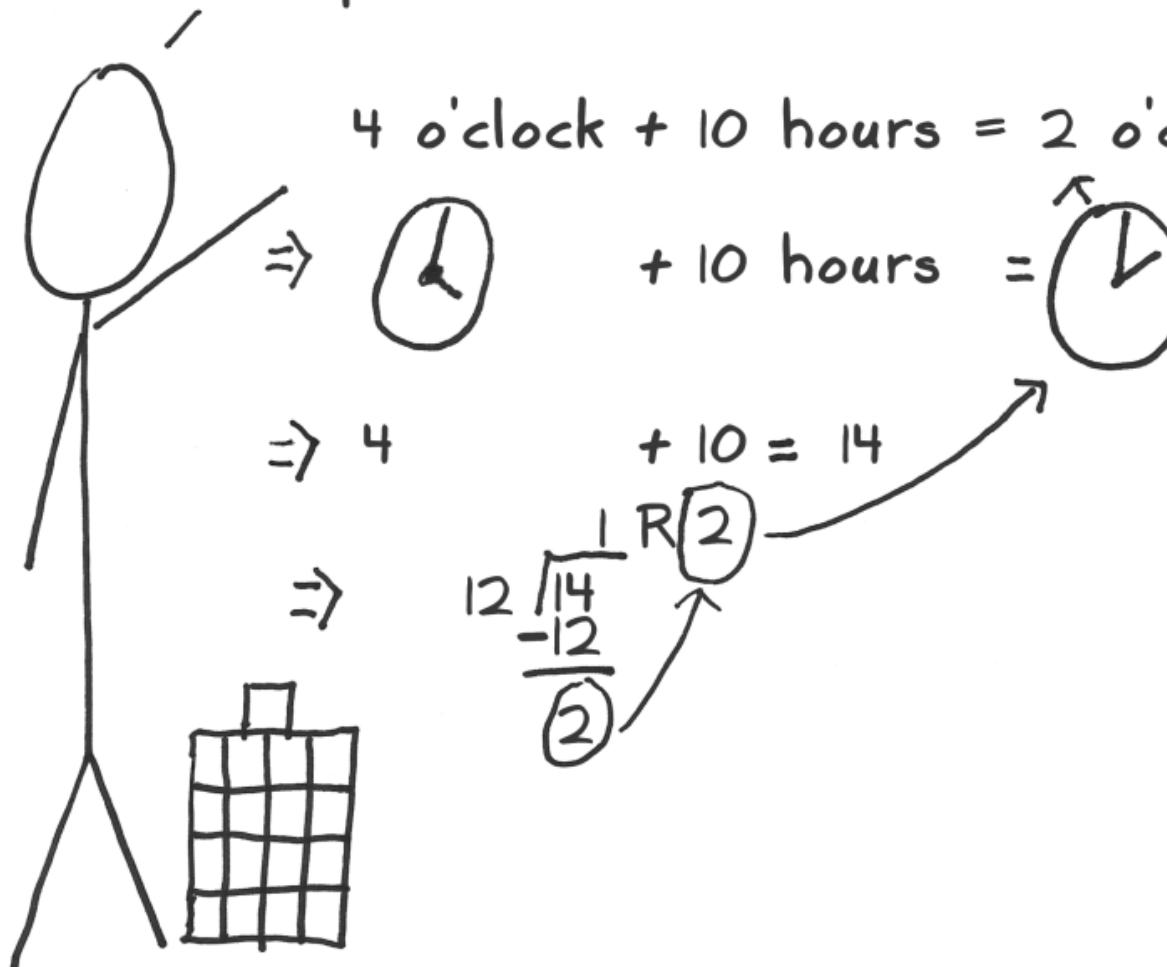
$$x^{13} + 2x^{11} + 3x^9 + 4x^7 + 3x^5 + 2x^3 + x$$

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

$$= x^{13} + x^9 + x^5 + x$$



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



\*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  $b_7 \dots b_0$ :

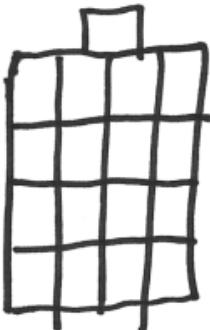
$$x \cdot b(x)$$

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

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



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



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

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


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

Remainder —

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

↑

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

↑

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:

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

$$\text{So... } \log(10 \cdot 100) = \log(10^1) + \log(10^2) \\ = 2 + 1 = 3$$

In reverse:

$$\begin{aligned}\log^{-1}(1) &= 10^1 = 10 \\ \log^{-1}(2) &= 10^2 = 100 \\ \log^{-1}(3) &= 10^3 = 1,000\end{aligned}$$



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

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



$$(x+1)^1 = x+1$$

$$(x+1)^2 = (x+1)(x+1) = x^2 + x + x + 1 = x^2 + 1$$

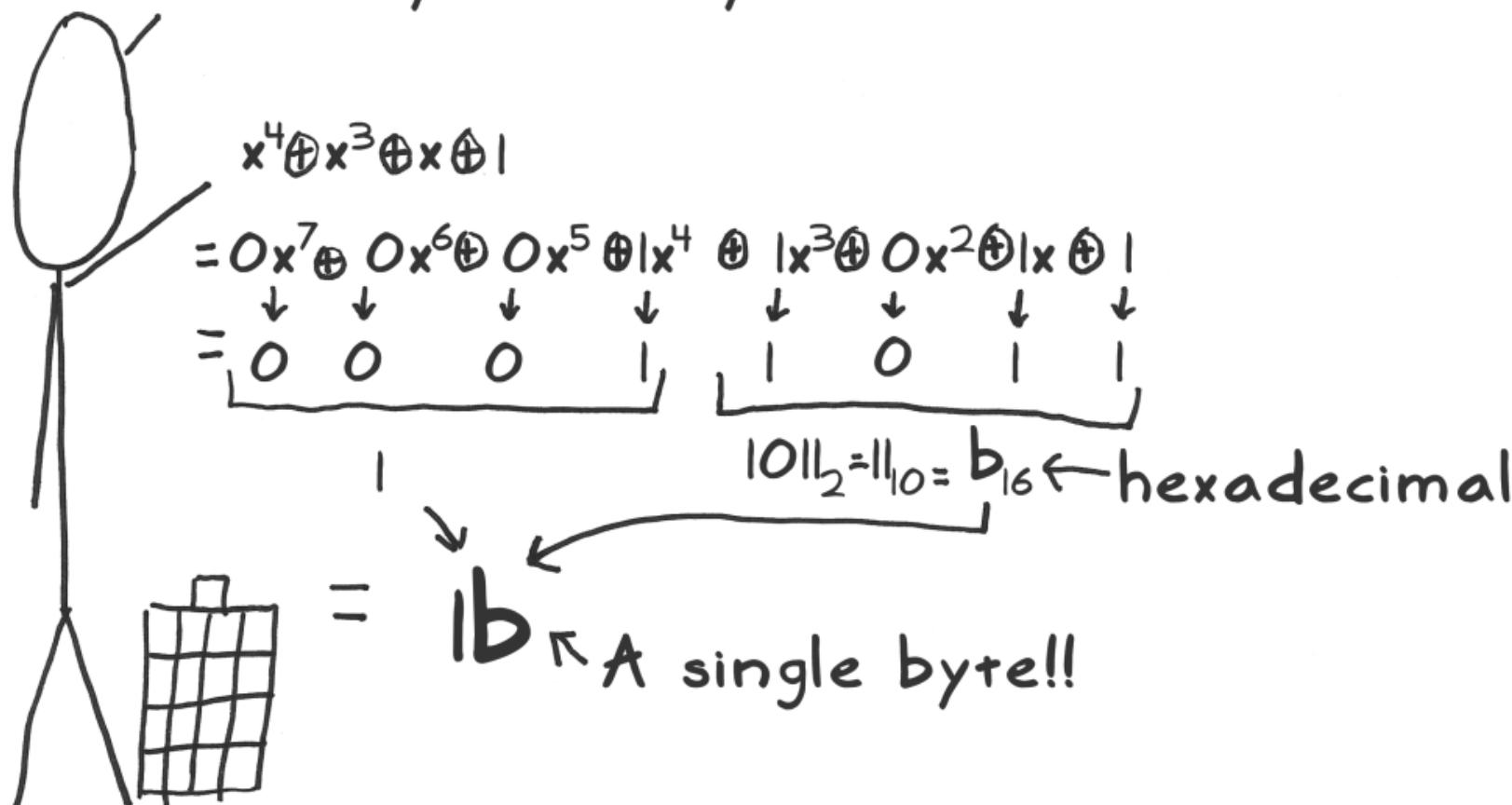
$$(x+1)^3 = (x+1)^2 \cdot (x+1) = x^3 + x^2 + x + 1$$

So...

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

\*If you keep multiplying by  $(x+1)$  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<sup>th</sup> 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.

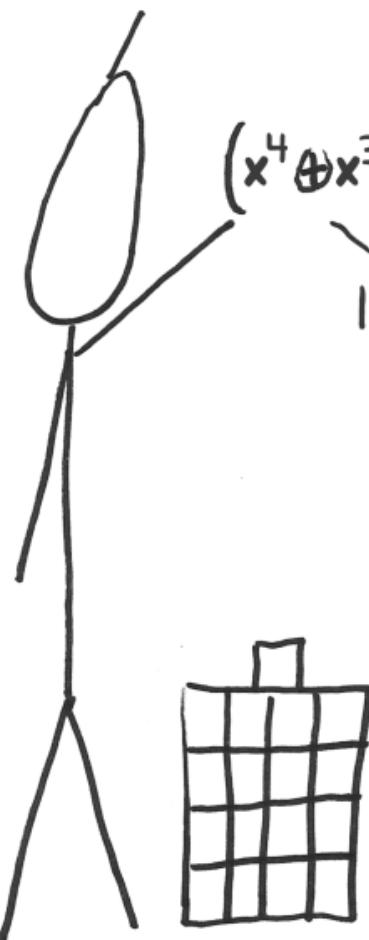
With bytes, polynomial addition becomes a simple xor. We can use our logarithm skills to make a table for speedy multiplication.\*


$$\begin{aligned} & (x^4 \oplus x^3 \oplus x \oplus 1) \oplus (x^7 \oplus x^5 \oplus x^3 \oplus x) \\ &= lb \quad \quad \quad \oplus \quad aa \quad \leftarrow \text{byte xor} \\ &= bl \\ &= x^7 \oplus x^5 \oplus x^4 \oplus x^1 \end{aligned}$$

$$\begin{aligned} & (x^4 \oplus x^3 \oplus x \oplus 1) \cdot (x^7 \oplus x^5 \oplus x^3 \oplus x) \\ &= lb \quad \cdot \quad aa \quad \begin{matrix} \downarrow & \downarrow \\ \text{logarithm} & \text{table lookup} \end{matrix} \\ \Rightarrow & \log(lb) + \log(aa) = c8 + 1f = e7 \\ & \qquad \qquad \qquad \begin{matrix} \downarrow & \downarrow \\ \text{inverse} & \text{table lookup} \end{matrix} \\ \Rightarrow & \log^{-1}(e7) = 8c \Rightarrow lb \cdot aa \\ &= x^7 \oplus x^3 \oplus x^2 \end{aligned}$$

\*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.

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

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

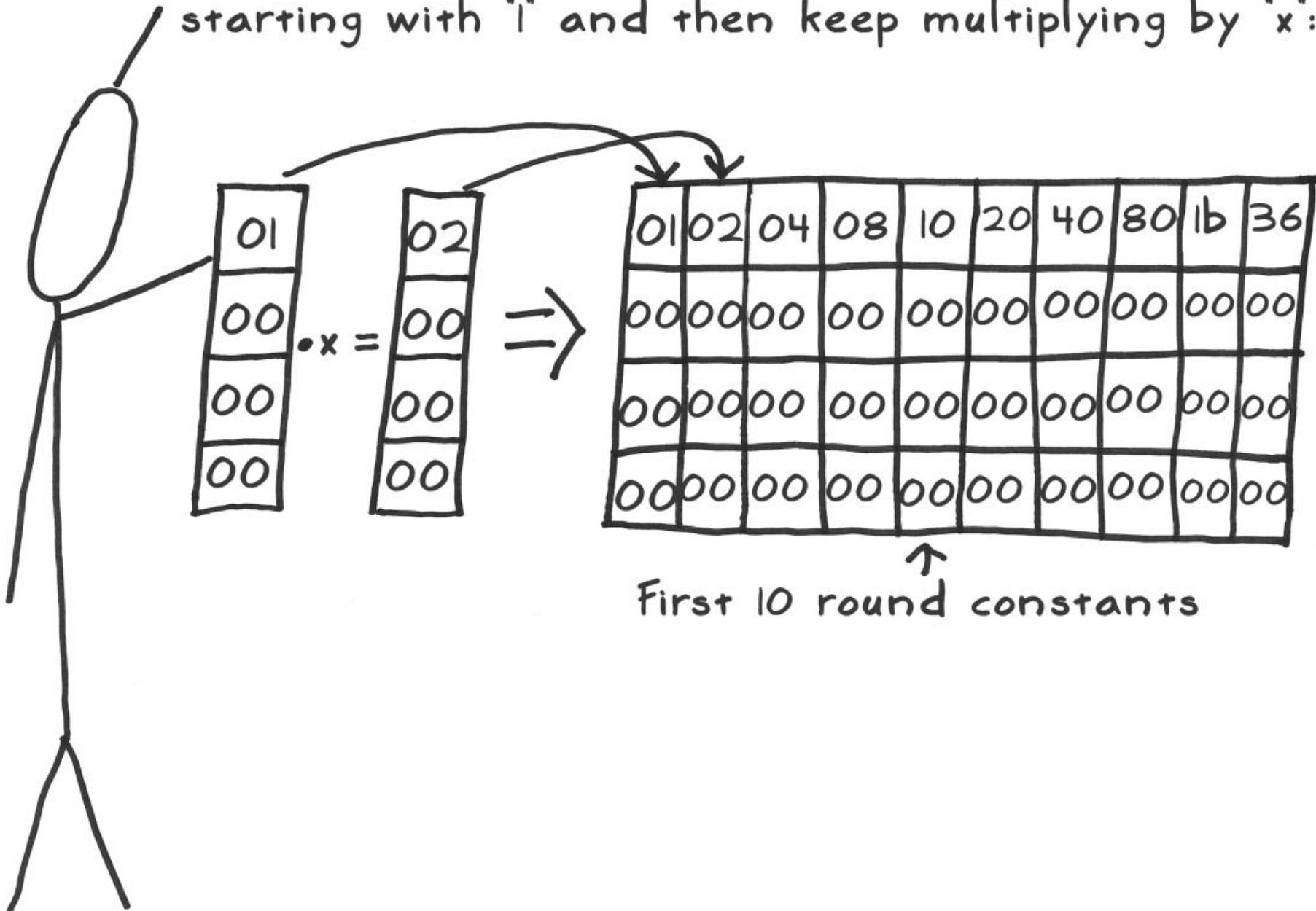
$$\text{sbox}[a] = f(g(a))$$

$$\text{sbox}[58] = f(g(58))$$

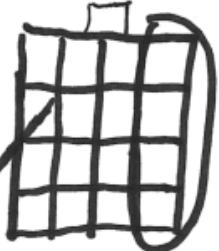
$$\text{sbox}[58] = f(18) = 6a$$

$$\uparrow$$
$$58 \cdot 18 = 01$$

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



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:



$$\begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

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

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

special polynomial

the column

$$03a_3 \cdot x^2 + (3a_2 + a_3)x + (3a_1 + a_2 + a_3)$$

$$= x^4 + 1 \overline{03a_3x^6 + 03a_2x^5 + 03a_1x^4 + 03a_0x^3 + 01a_3x^5 + 01a_2x^4 + 01a_1x^3 + 01a_0x^2 + 01a_3x^4 + 01a_2x^3 + 01a_1x^2 + 01a_0x + 02a_3x^3 + 02a_2x^2 + 02a_1x + 02a_0}$$

$$\oplus \overline{03a_3x^6 + 03a_3x^6}$$

$$3a_2x^5 + 3a_1x^4 + 3a_0x^3 + a_3x^5 + a_2x^4 + a_1x^3 + a_0x^2 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x + 2a_3x^3 + 2a_2x^2 + 2a_1x + 2a_0$$

$$\oplus \overline{3a_2x^5 + a_3x^5 + 3a_2x + a_3x}$$

$$3a_1x^4 + 3a_0x^3 + a_2x^4 + a_1x^3 + a_0x^2 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x + 2a_3x^3 + 2a_2x^2 + 2a_1x + 2a_0 + 3a_3x^2 + 3a_2x + a_3x$$

$$\oplus \overline{(3a_1 + a_2 + a_3)x^4 + (3a_1 + a_2 + a_3)}$$

$$(2a_3 + a_2 + a_1 + 3)x^3 + (3a_3 + 2a_2 + a_1 + a_0)x^2 + (a_3 + 3a_2 + 2a_1 + a_0)x + (a_3 + a_2 + 3a_1 + 2a_0)$$



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

Plaintext in  
4x4 grid

# AES Crib Sheet (Handy for memorizing)

Shift Rows Row Shift



Initial Round

General Math

$$11B = \text{AES Polynomial} = m(x)$$

$$X^8 + X^4 + X^3 + X + 1 \quad \text{Fast Multiply}$$

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

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

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

S-Box (SRD)



$$\text{SRD}[a] = f(g(a))$$

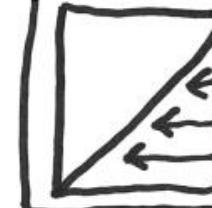
$$g(a) = a^{-1} \bmod m(x)$$

$f(a)$  Think  $53 \oplus 63^T$

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

11111000	$a_7$	0
01111100	$a_6$	1
00111110	$a_5$	0
00011111	$a_4$	0
10001111	$a_3$	0
11000111	$a_2$	1
11000111	$a_1$	1
11100011	$a_0$	0

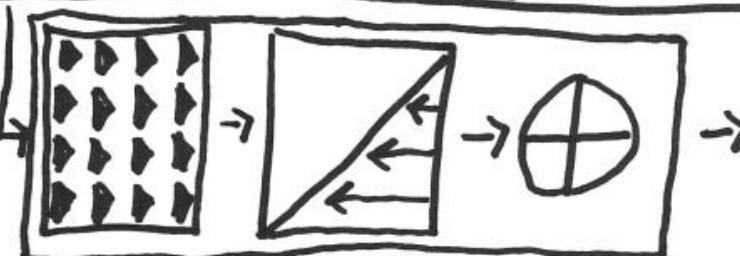
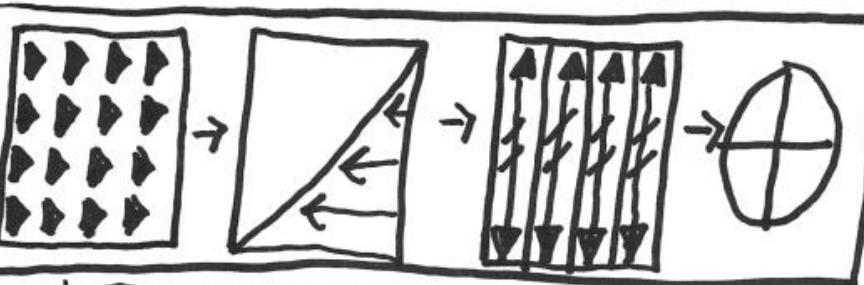
# AES Crib Sheet (Handy for memorizing)



Shift Rows Row Shift

Intermediate Rounds

#	Key
9	128
11	192
13	256



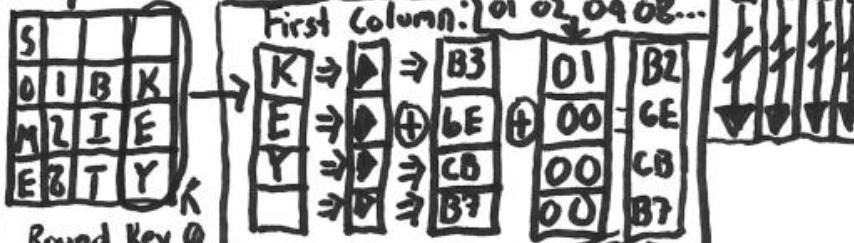
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

Ciphertext

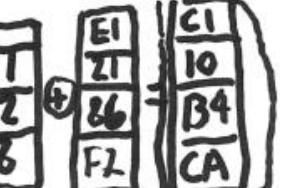
Final Round ↗

Key Expansion: Round Constants

First Column: 01 02 09 08 ...



Other Columns:

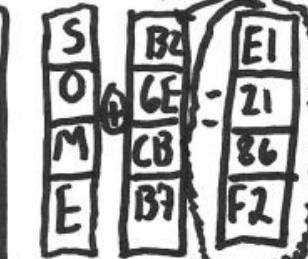


Prev Col  $\oplus$  Col from Previous round key

Mix Columns:

21132

2	1	1	3	2	$a_3$
2	1	1	3	2	$a_2$
1	3	2	1	3	$a_1$
1	1	3	2	1	$a_0$



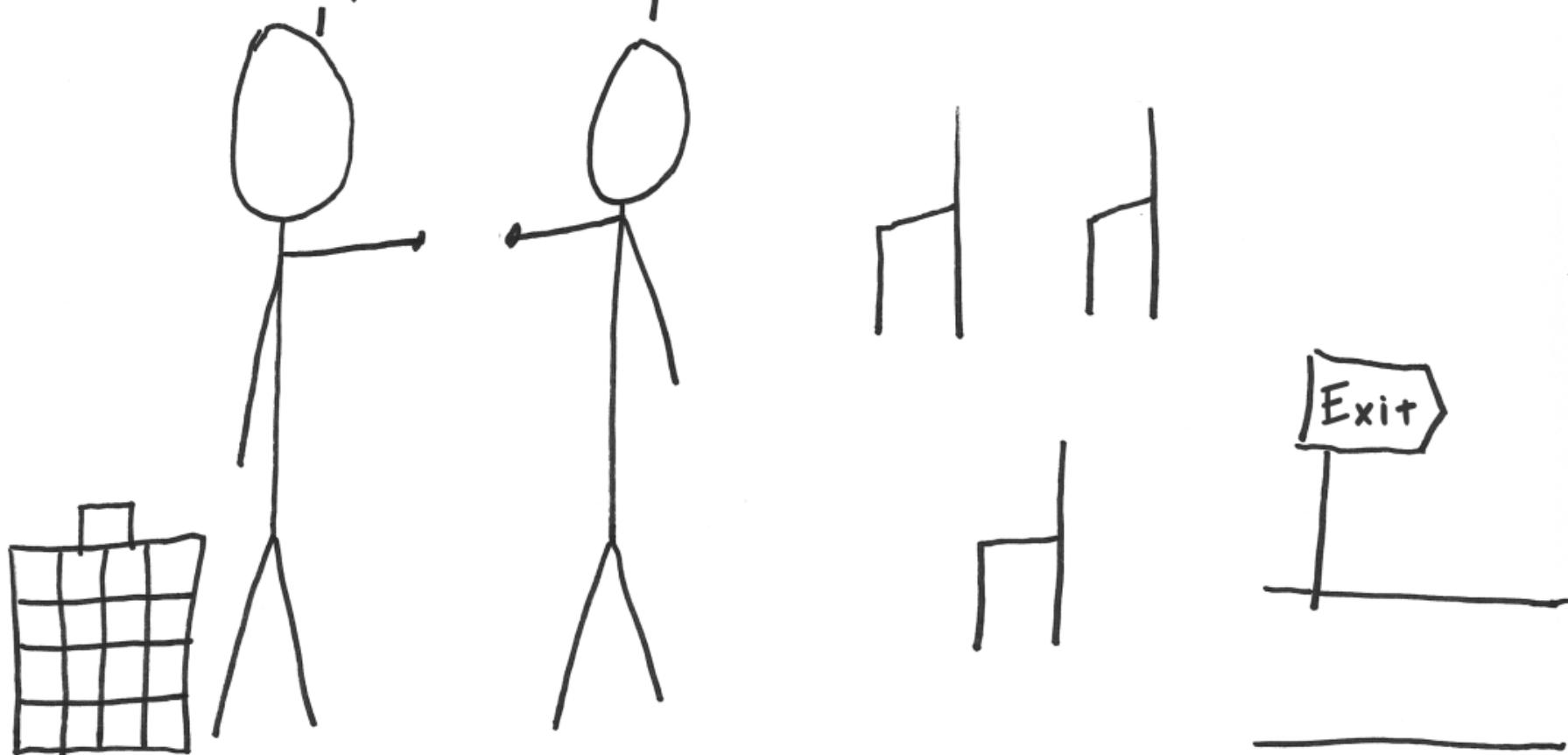
Inverse Mix

EBD9

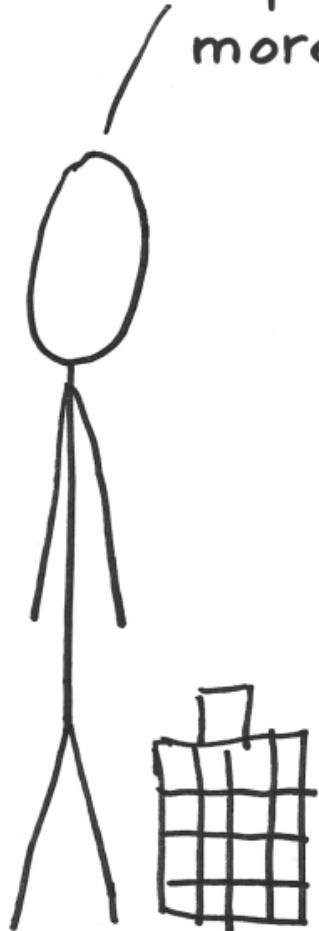
E	B	D	9	$a_3$
9	E	B	D	$a_2$
D	9	E	B	$a_1$
B	D	9	B	$a_0$

Whoa... I think I get it now. It's relatively simple once you grok the pieces. Thanks for explaining it. I gotta go now.

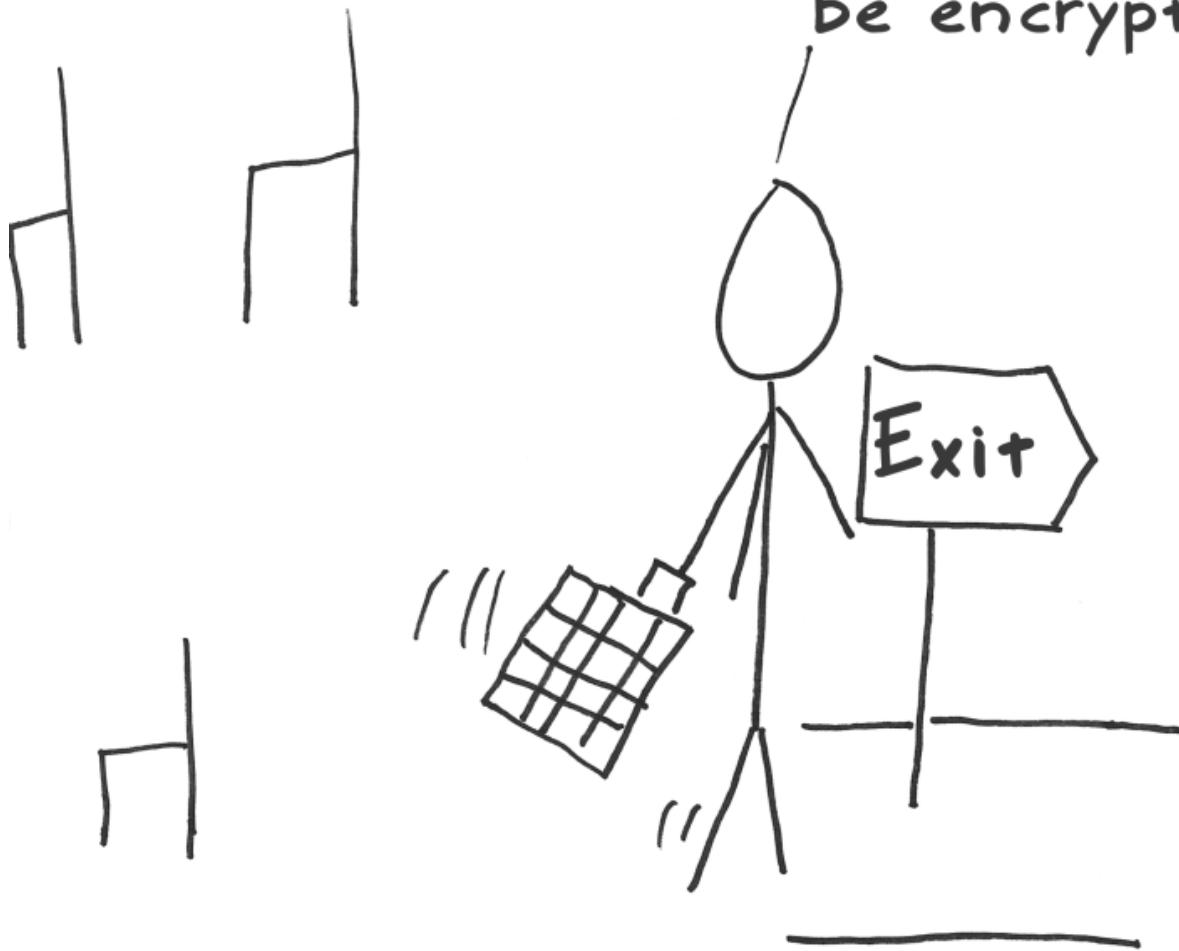
My pleasure.  
Come back anytime!

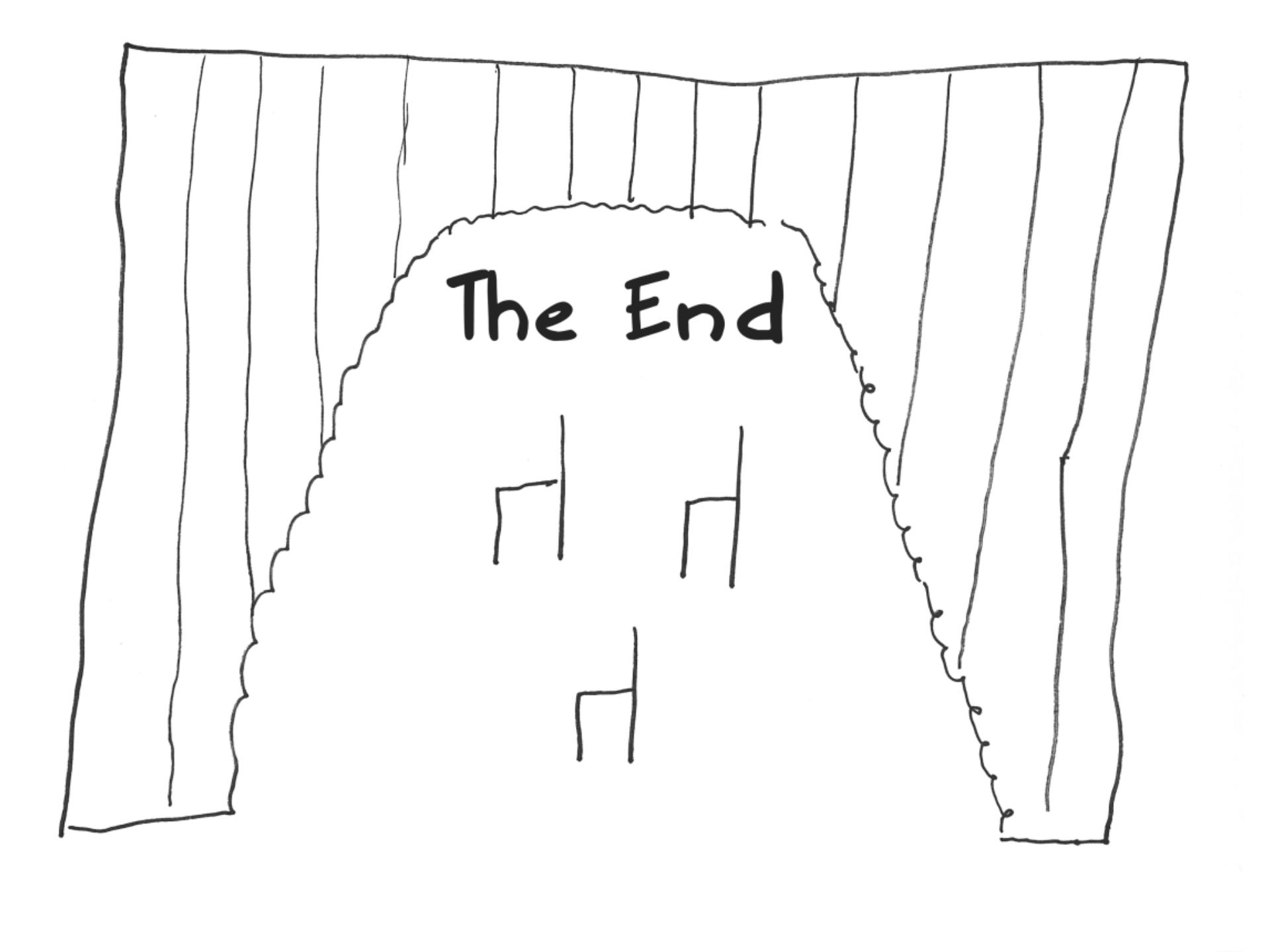


But there's so much more to talk about: my resistance to linear and differential cryptanalysis, my Wide Trail Strategy, impractical related-key attacks, and... so much more... but no one is left.



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





The End

