



University of  
Pittsburgh

# Applied Cryptography and Network Security

## CS 1653



Summer 2023

Sherif Khattab

[ksm73@pitt.edu](mailto:ksm73@pitt.edu)

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

# Announcements

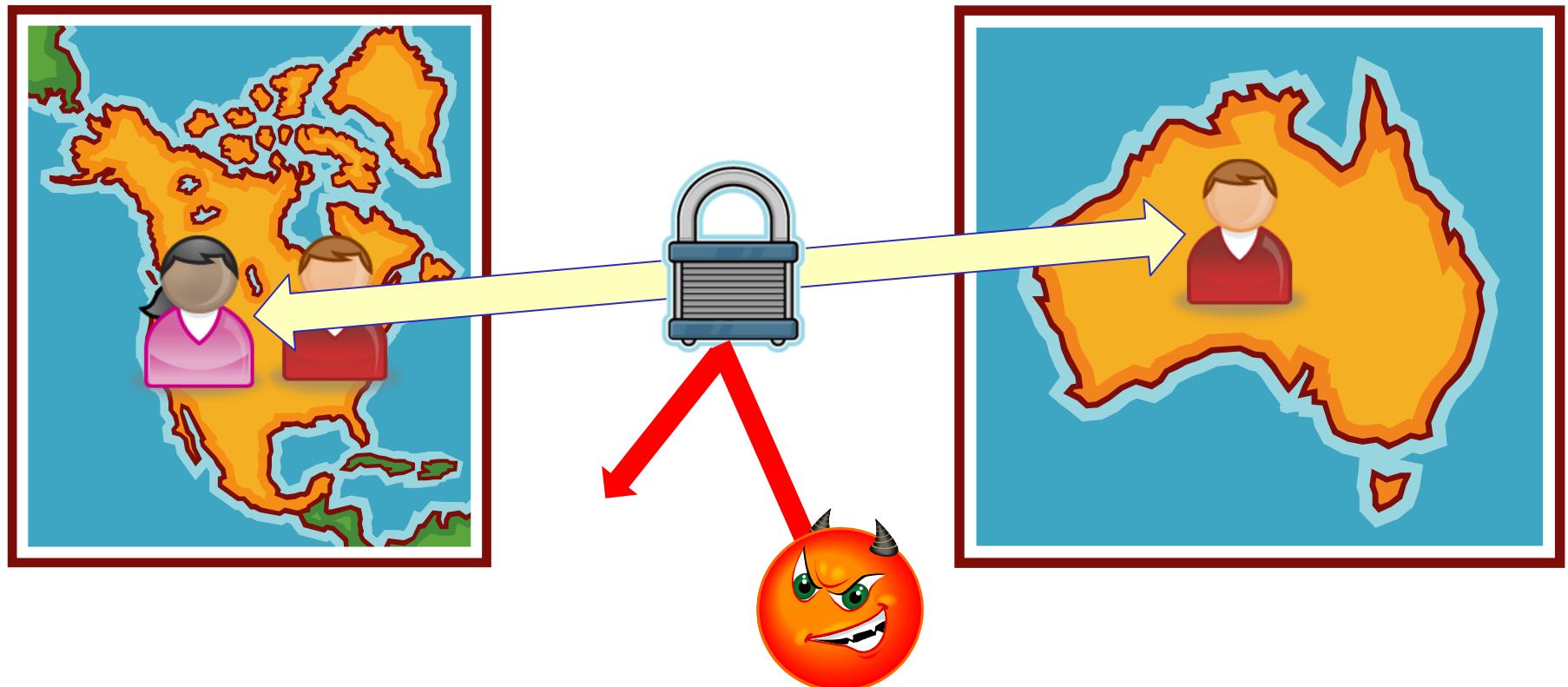
- Please skim over assigned reading
- Homework 1 due this Friday @ 11:59 pm
  - 3 attempts
- add 15 minutes to each lecture
  - updated lecture time: 1:30 pm – 3:30 pm
  - **feel free to leave at 3:15 if you have to**
- Makeup lectures
  - Friday 6/16 11:00-12:45
  - Friday 6/23 15:00-16:15
- **lectures are recorded**
- No Tophat questions during makeup time

# A Motivating Scenario

How can Alice and Bob communicate over an untrustworthy channel?

Need to ensure that:

1. Their conversations remain secret (**confidentiality**)
2. Modifications to any data sent can be detected (**integrity**)



# Recall our cryptographic model...

Formally, a cryptosystem can be represented as the 5-tuple  $(E, D, M, C, K)$

$M$  is a message space

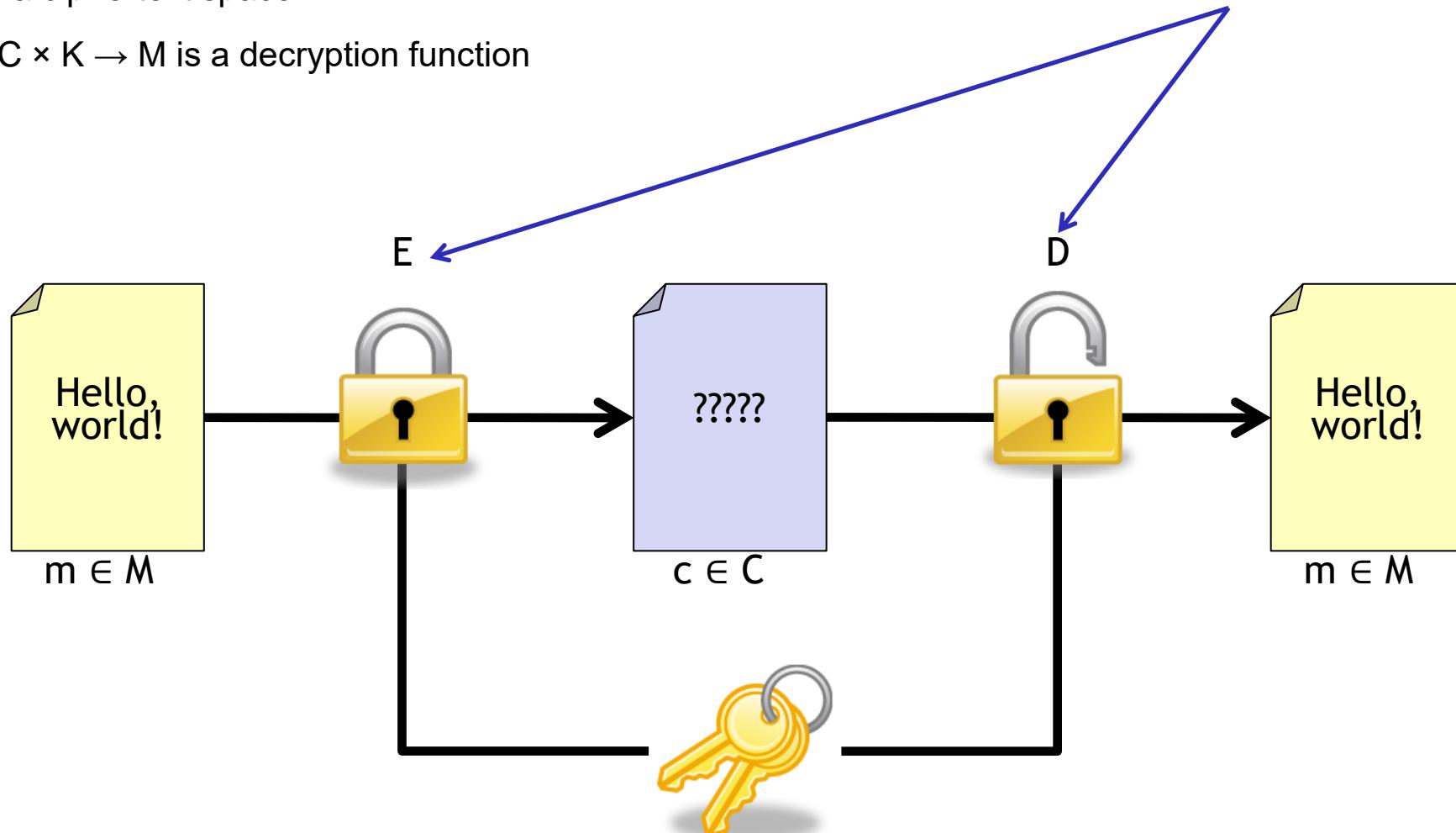
$K$  is a key space

$E : M \times K \rightarrow C$  is an encryption function

$C$  is a ciphertext space

$D : C \times K \rightarrow M$  is a decryption function

Our focus now is on **symmetric key** encryption



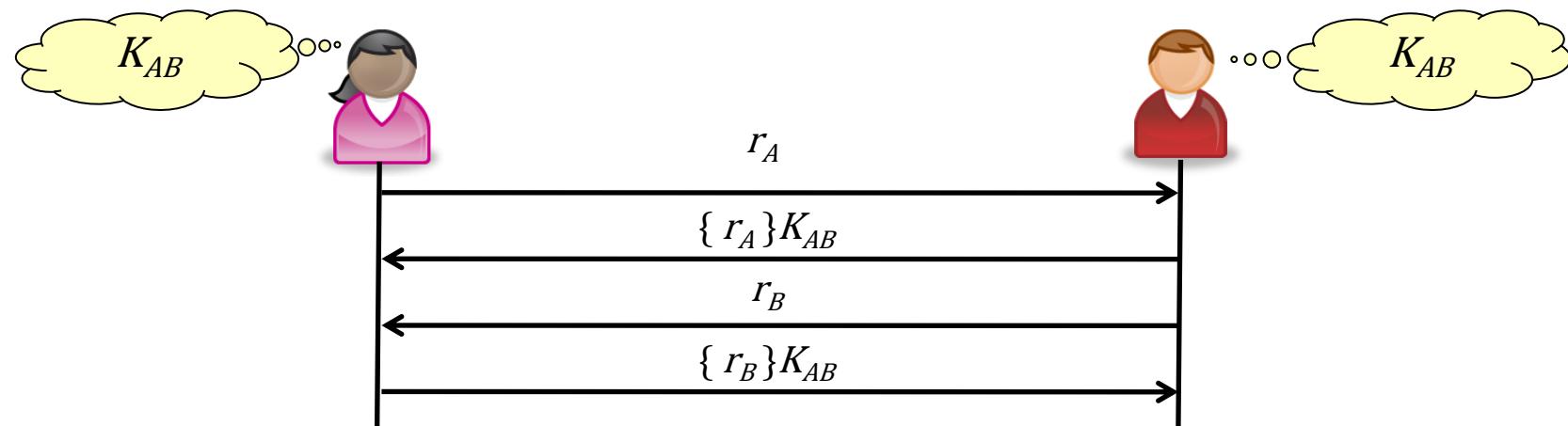
# Why study symmetric key cryptography?

Rather obvious good uses of symmetric key cryptography include:

- Transmitting data over insecure channels
  - SSL, SSH, etc.
- Securely storing sensitive data in untrusted places
  - Malicious administrators
  - Cloud computing
  - ...
- Integrity verification and tamper resistance

We'll go over these types of protocols in gory detail later ...

Authentication is (perhaps) a less obvious use of symmetric key crypto



Classical crypto algorithms are symmetric-key ciphers

Unfortunately, most of these ciphers offer essentially no protection in modern times

The exception is the one-time pad which offers perfect security from an information theory perspective

Namely, a **single ciphertext** of length  $n$  can decrypt to **any message** of length up to  $n$

More formally,  $H(m) = H(m \mid c)$

However, the large amount of key material required by the one-time pad is a hindrance to its use for many practical purposes

To transmit a message of length  $n$ , you need a key of length  $n$

If you have a secure channel to transmit  $n$  bits of key, why not use it to transmit  $n$  bits of message instead?

Modern cryptography: fixed-length key to encipher variable-length data

In an ideal world, we would like to have the **perfect** security guarantees of the one-time pad, without the hassle of requiring our key length to equal our message length

This is **very** difficult!

However, modern cryptographers have developed many algorithms that give **good** security using very **small keys**

Today, we'll study two classes of symmetric key algorithms

**Stream ciphers:** RC4, SEAL, etc.

**Block ciphers:** DES, TDES, AES, Blowfish, etc.

# Stream ciphers work like a one-time pad

The secrecy of a stream cipher rests entirely on PRNG “randomness”

PRNG = Pseudo-Random Number Generator

Often, we see stream ciphers used in communications **hardware**

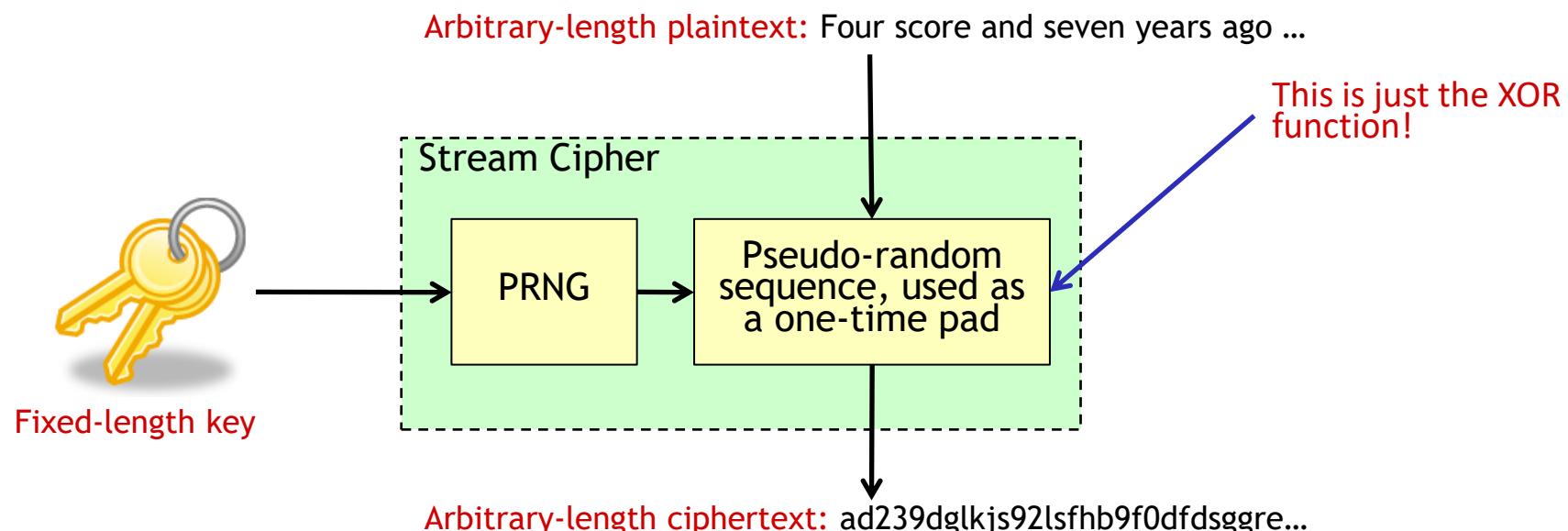
Single-bit transmission error effects only single bit of plaintext

Low transmission delays

Key stream can (sometimes) be **pre-generated** and buffered

Encryption is just an XOR

No buffering of data to be transmitted



# Two types of stream ciphers

In a **synchronous** stream cipher, the key stream is generated independently of the ciphertext

## **Advantages**

- Does not propagate bit flip errors

- Robust against **replay attack**

- Key stream can be pre-generated

## **Disadvantages**

- May need to change keys often **if periodicity of PRNG is low**

- Vulnerable to **insertion/drop attack**

# Two types of stream ciphers

In a **self-synchronizing** stream cipher, the key stream is also a function of the most recent  $n$  ciphertext bits

## ***Advantages***

Decryption key stream automatically synchronized with encryption key stream after correctly receiving  $n$  ciphertext bits → Robust against insertion/drop attack

Less frequent key changes, since key stream is a function of key and ciphertext

## ***Disadvantages***

Vulnerable to **replay attack**

Single-bit flips propagate up to  $n$  bits

Key stream cannot be pre-generated

# Block ciphers are more commonly used in software

Block ciphers achieve two properties identified by **Claude Shannon**

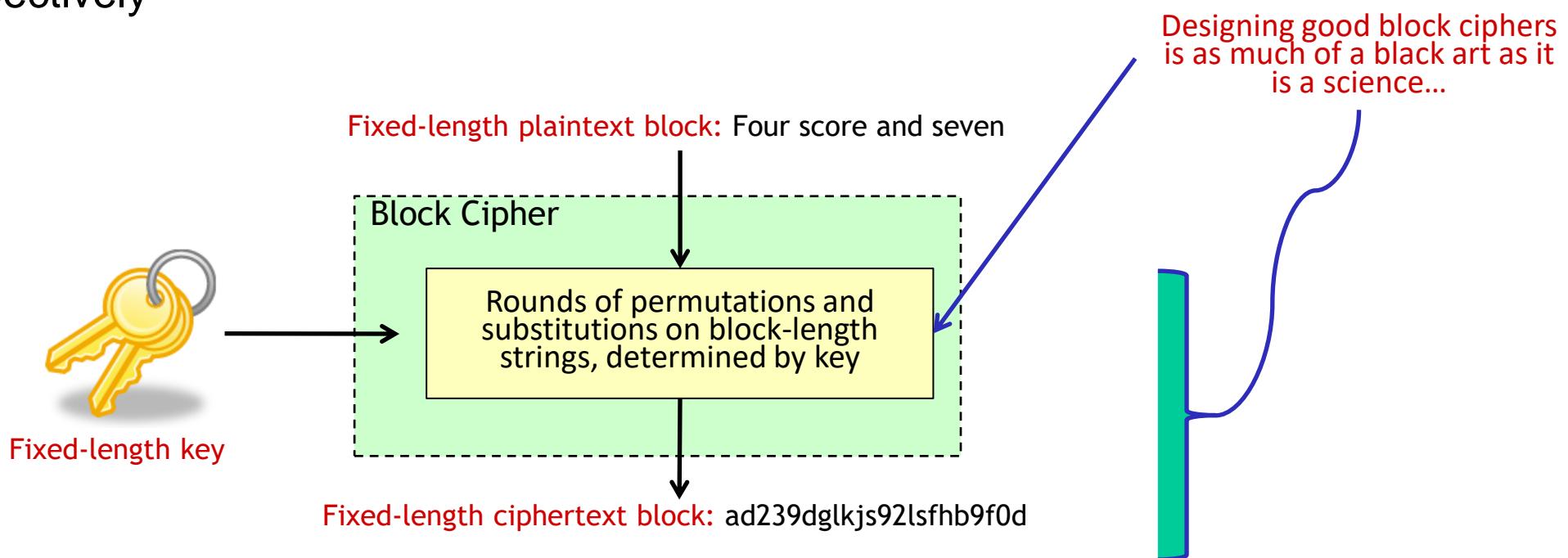
**Goal:** hide/obscure patterns in plaintext and key

Confusion: each bit in **ciphertext** depends on several parts of **key**

Diffusion: one bit change in **plaintext** changes several bits in **ciphertext** and vice versa

Change one bit → at least half of the other bits change

Confusion and Diffusion achieved using **substitution and permutation/shuffling**, respectively



# Block ciphers are more commonly used in software

Block ciphers operate on **fixed-length** blocks of plaintext

Typical block lengths: 40, 56, 64, 80, 128, 192 and 256 bits

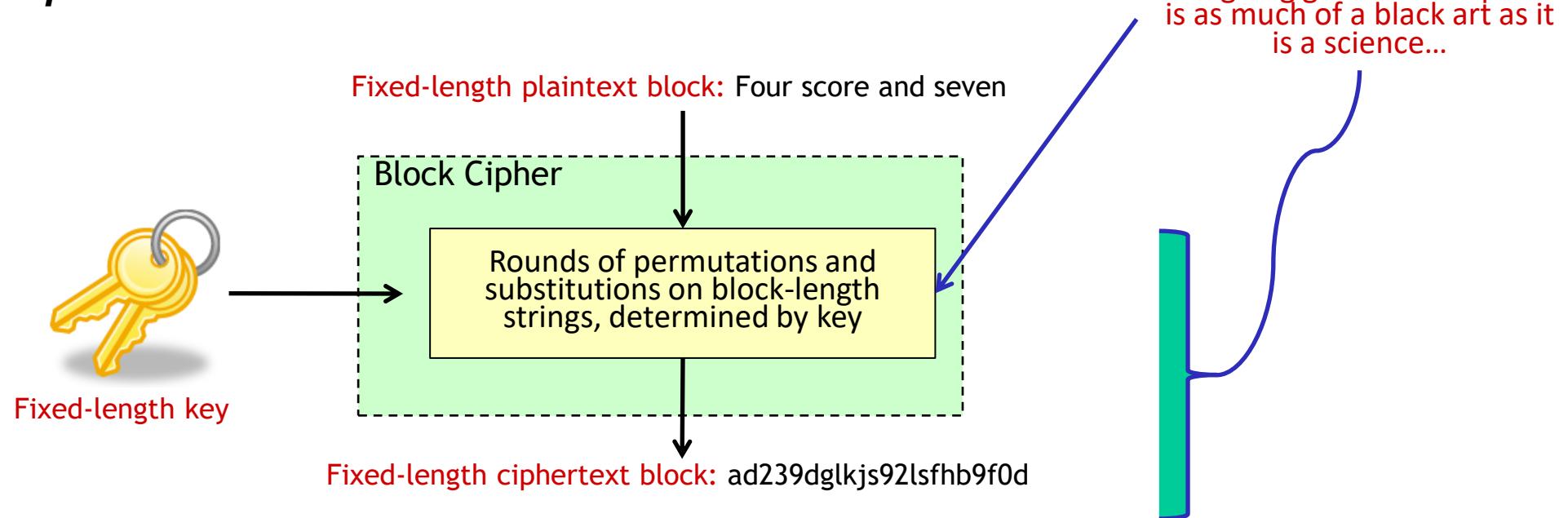
Often, block ciphers apply several rounds of a simpler function

Most block ciphers can be categorized as **Feistel networks**

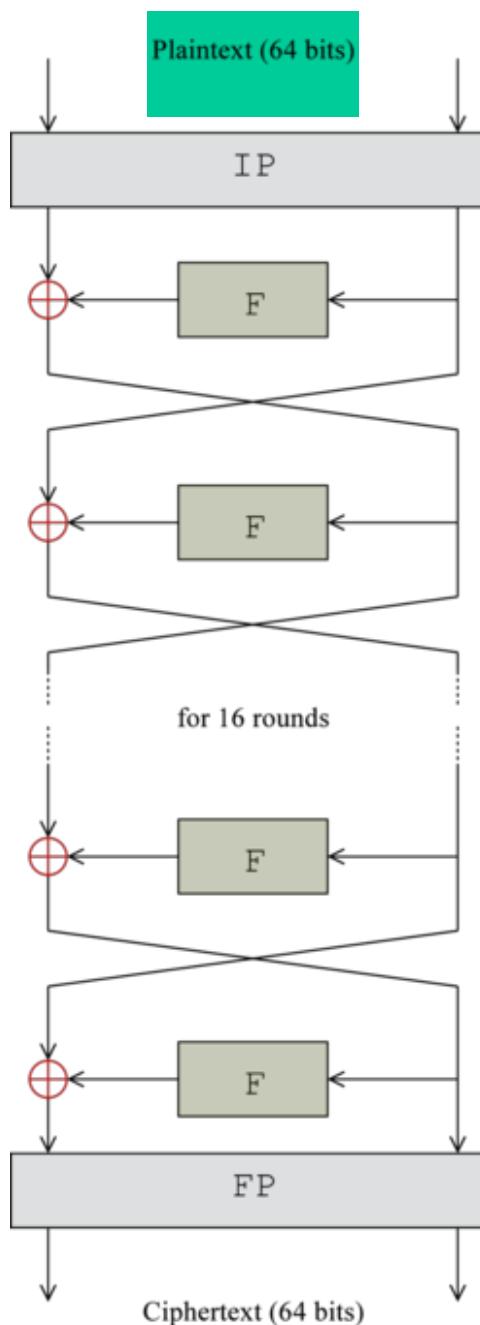
Bit shuffling, non-linear substitution, and linear mixing (XOR)

Confusion and diffusion *a la* Claude Shannon

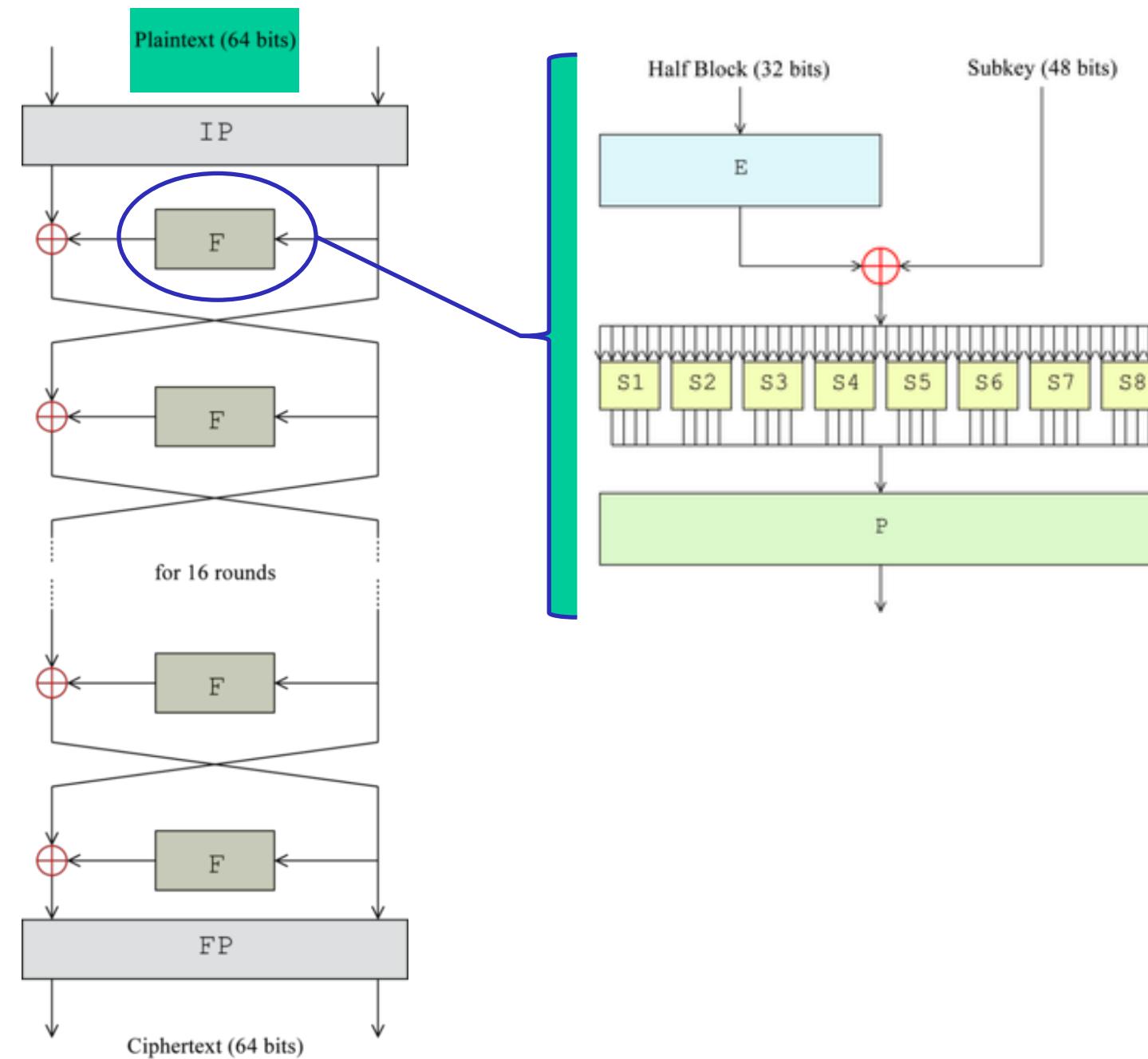
**Example:** DES is a Feistel network that uses 16 rounds



# Example: Data Encryption Standard (DES)

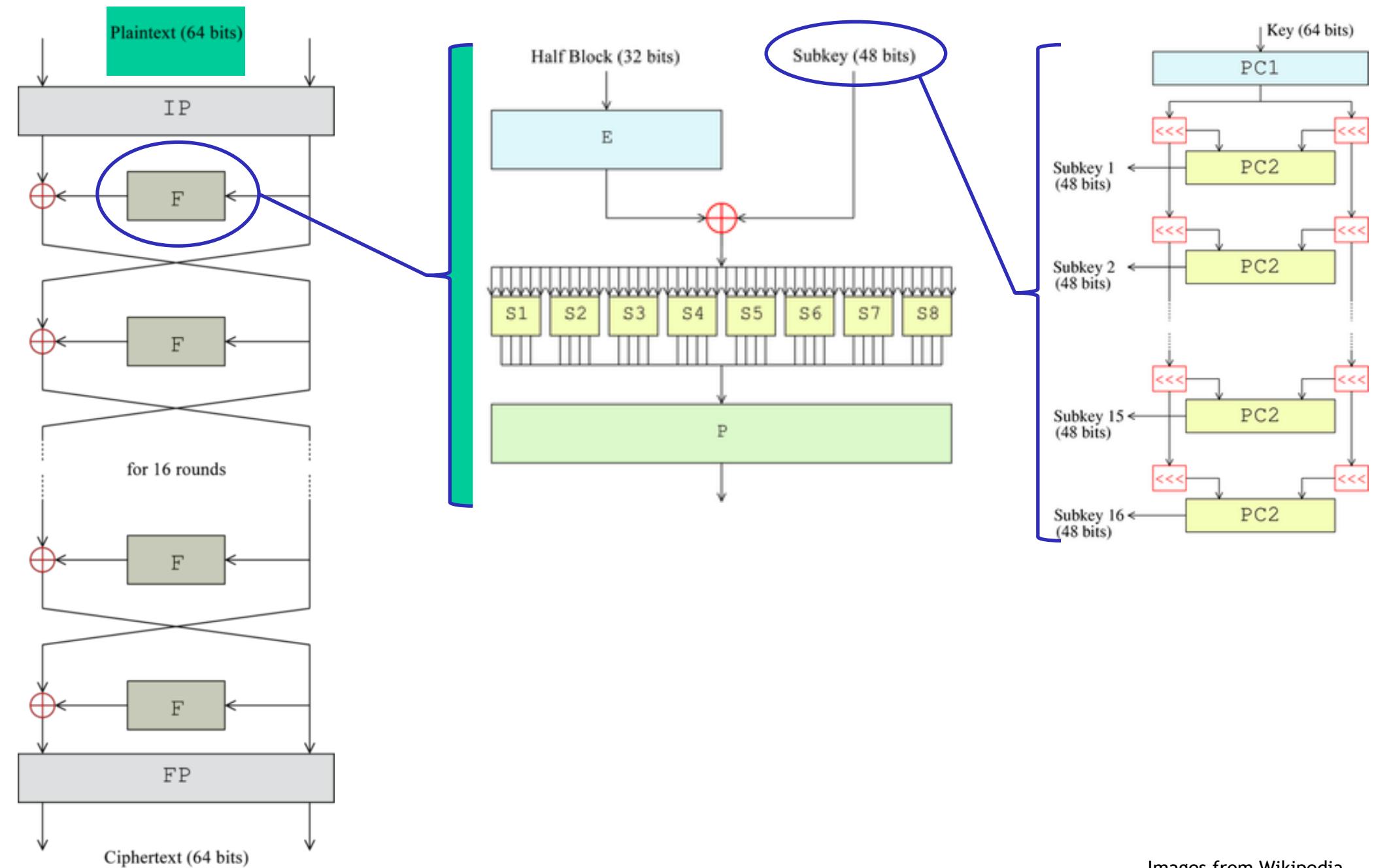


# Example: DES



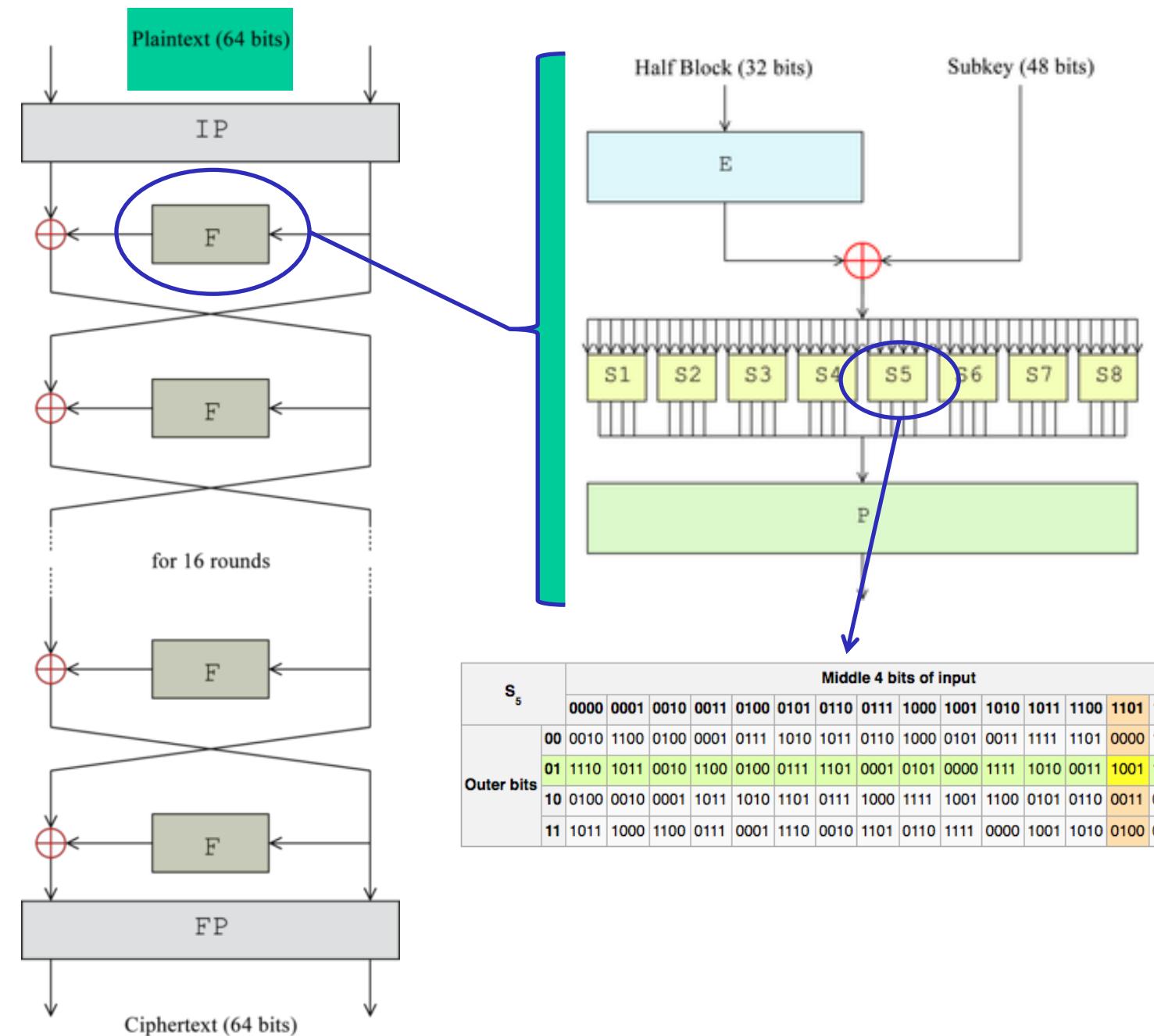
Images from Wikipedia

# Example: DES



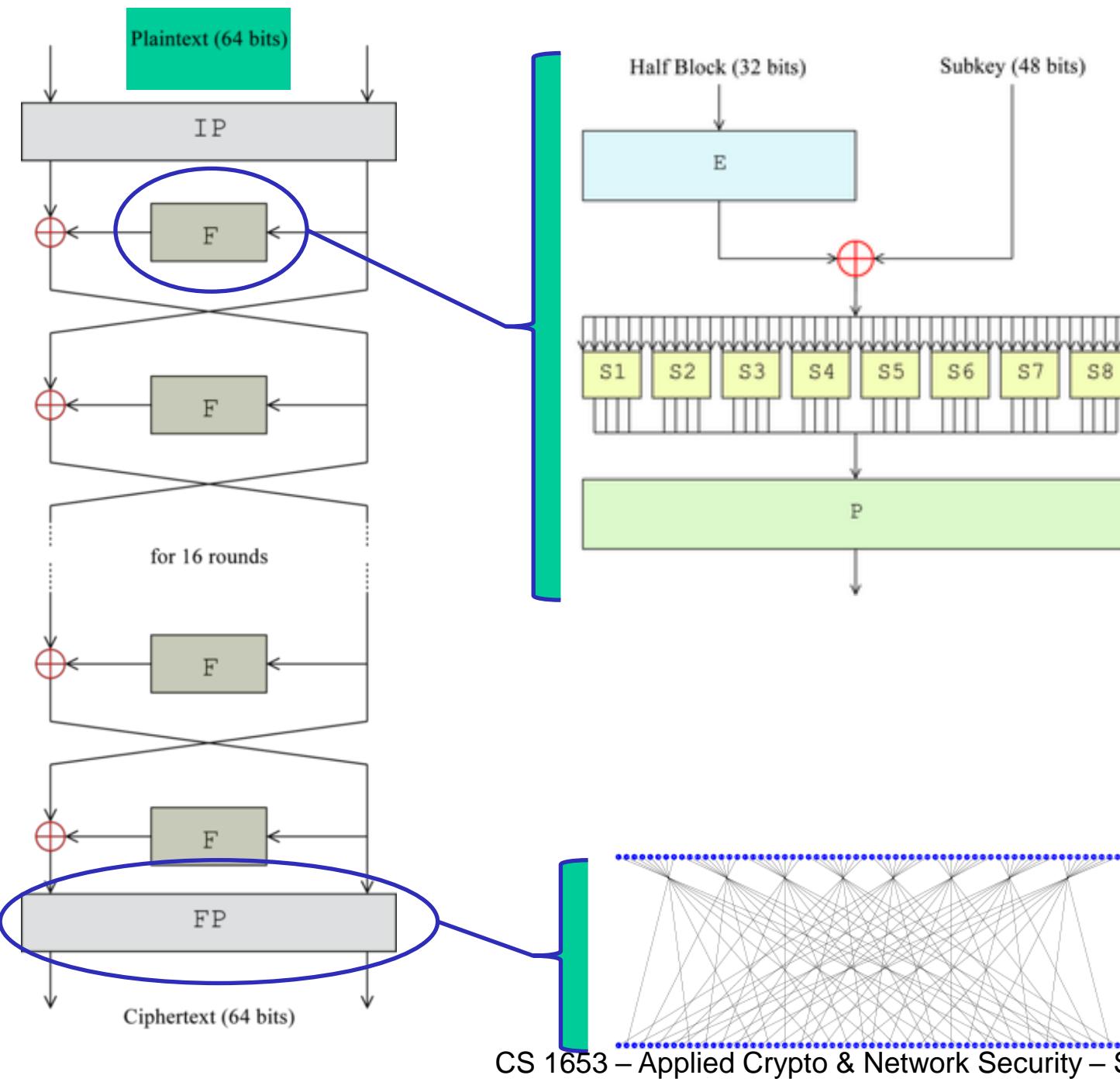
Images from Wikipedia

# Example: DES



Images from Wikipedia

# Example: DES



Images from Wikipedia

# Example: Advanced Encryption Standard (AES)

Develop a (brief) history of modern cryptography

Learn about the AES standardization process

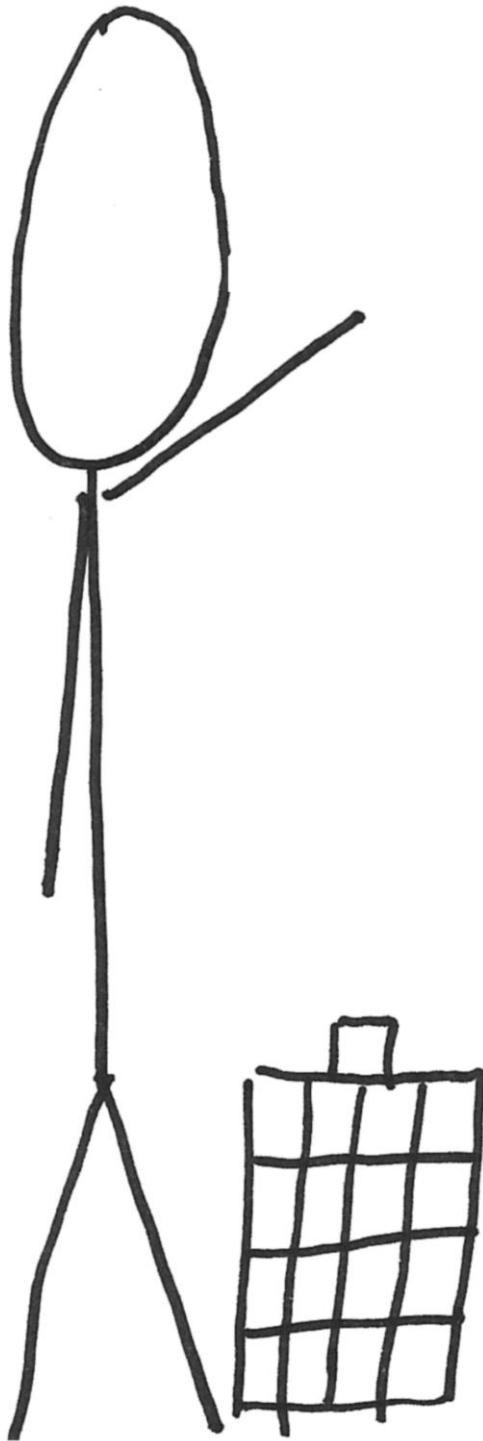
Understand AES:

**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

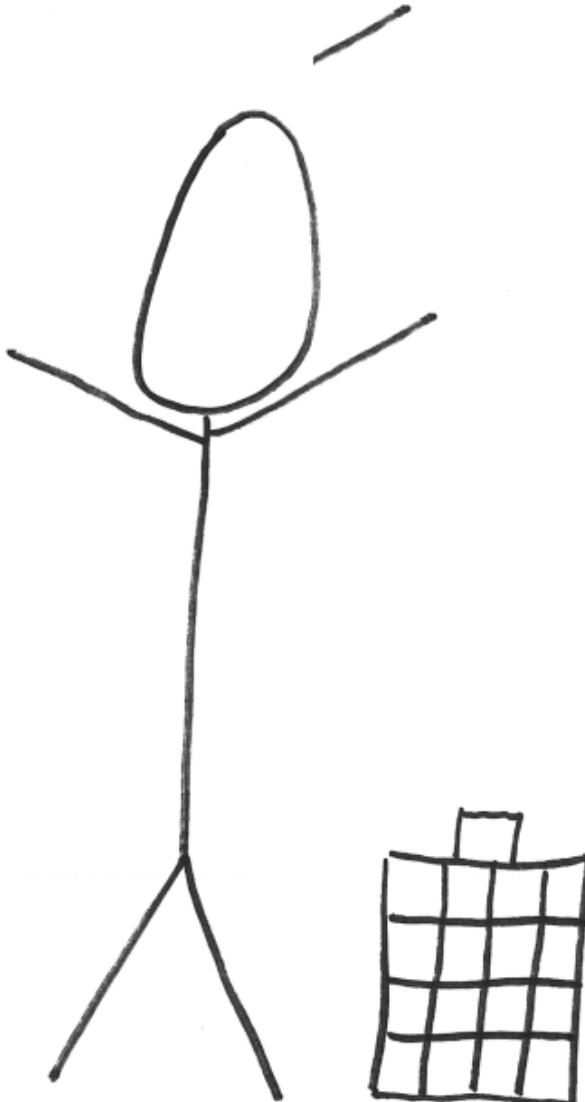
# A Stick Figure Guide to the Advanced Encryption Standard (AES)



<http://www.moserware.com/>

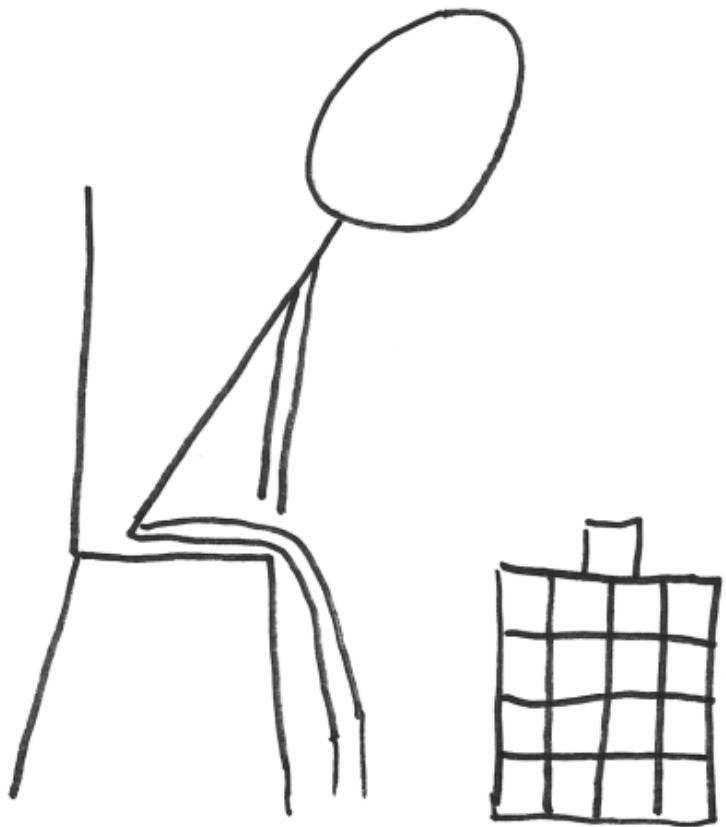
# **Act 1: Once Upon a Time...**

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!

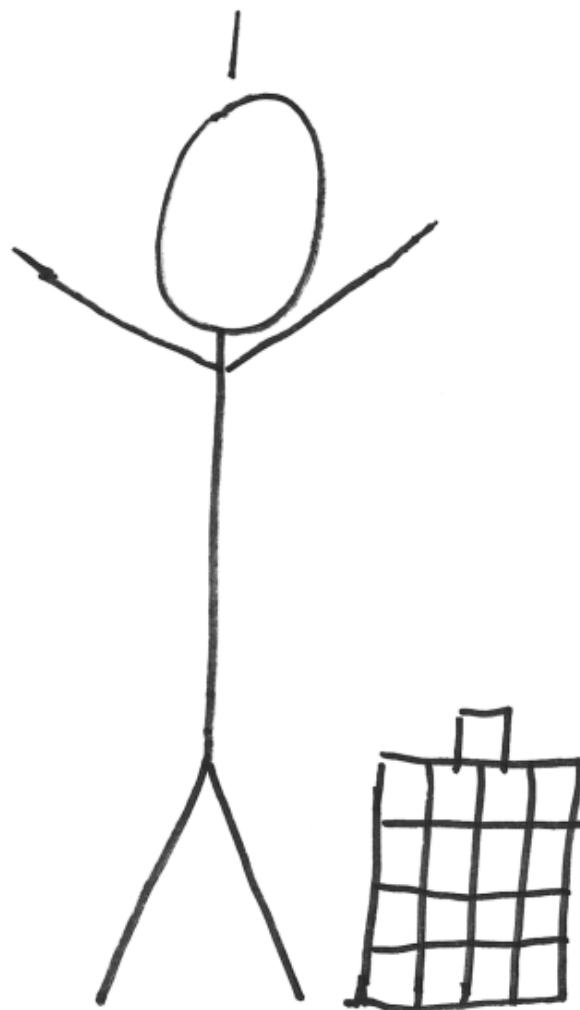


\* 1 petabyte ≈ a lot

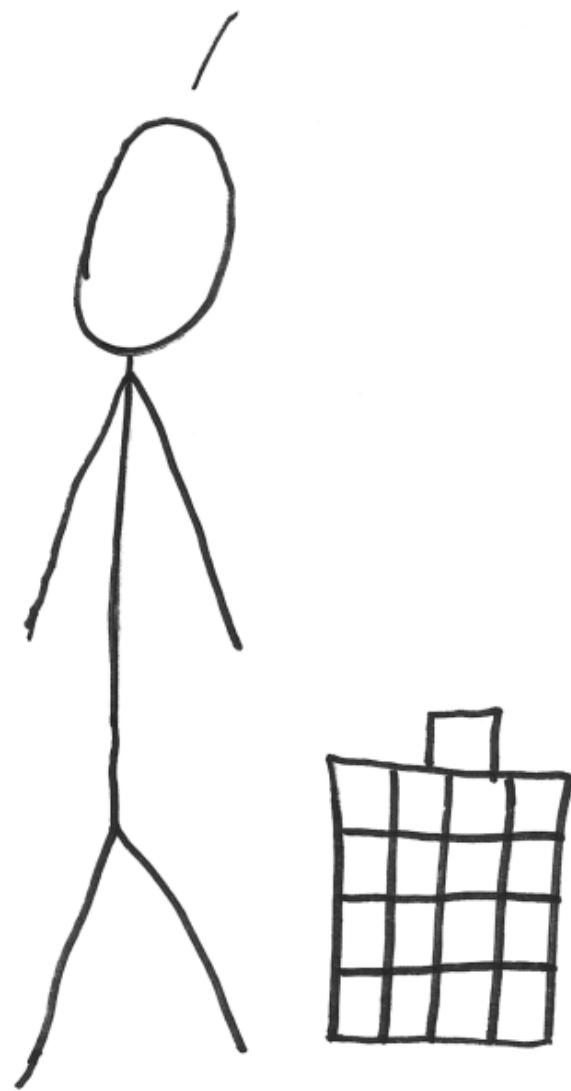
... and still no one seems to care  
about me or my story.



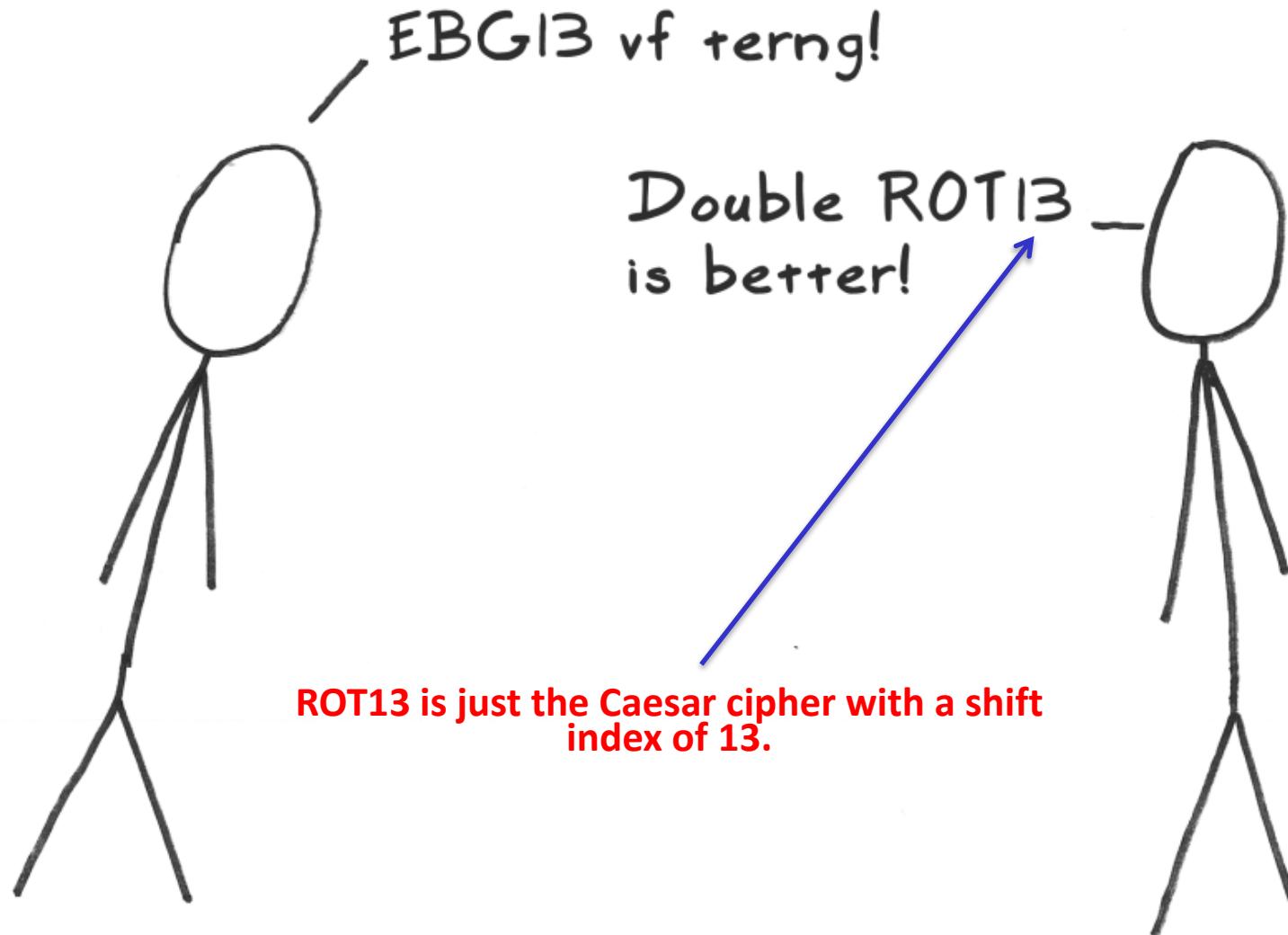
I've got a better-than-Cinderella  
story as I made my way to become  
king of the block cipher world.



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



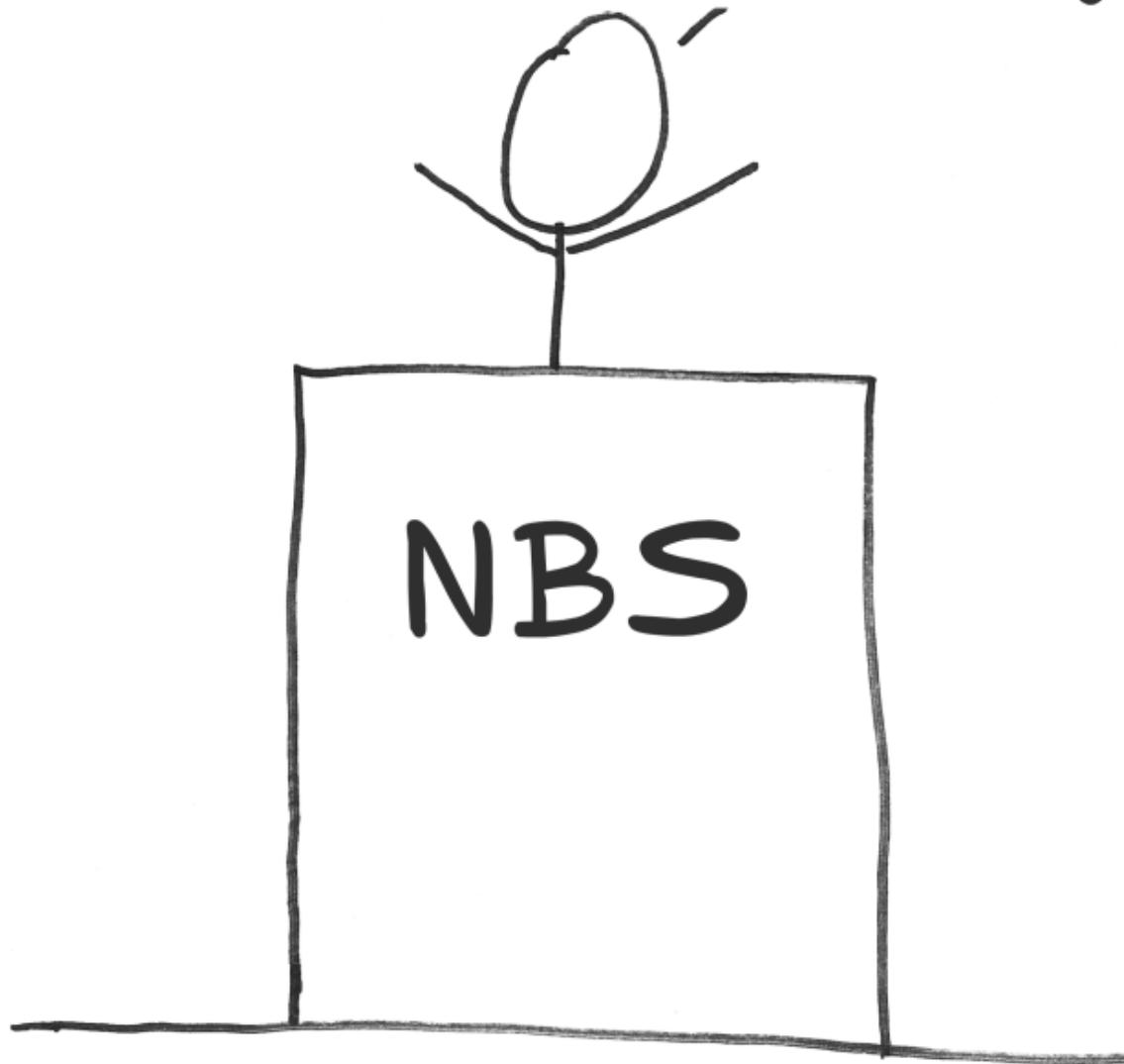
Once upon a time,\* there was no good way for people outside secret agencies to judge good crypto.



\* ~ pre-1975 for the general public

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

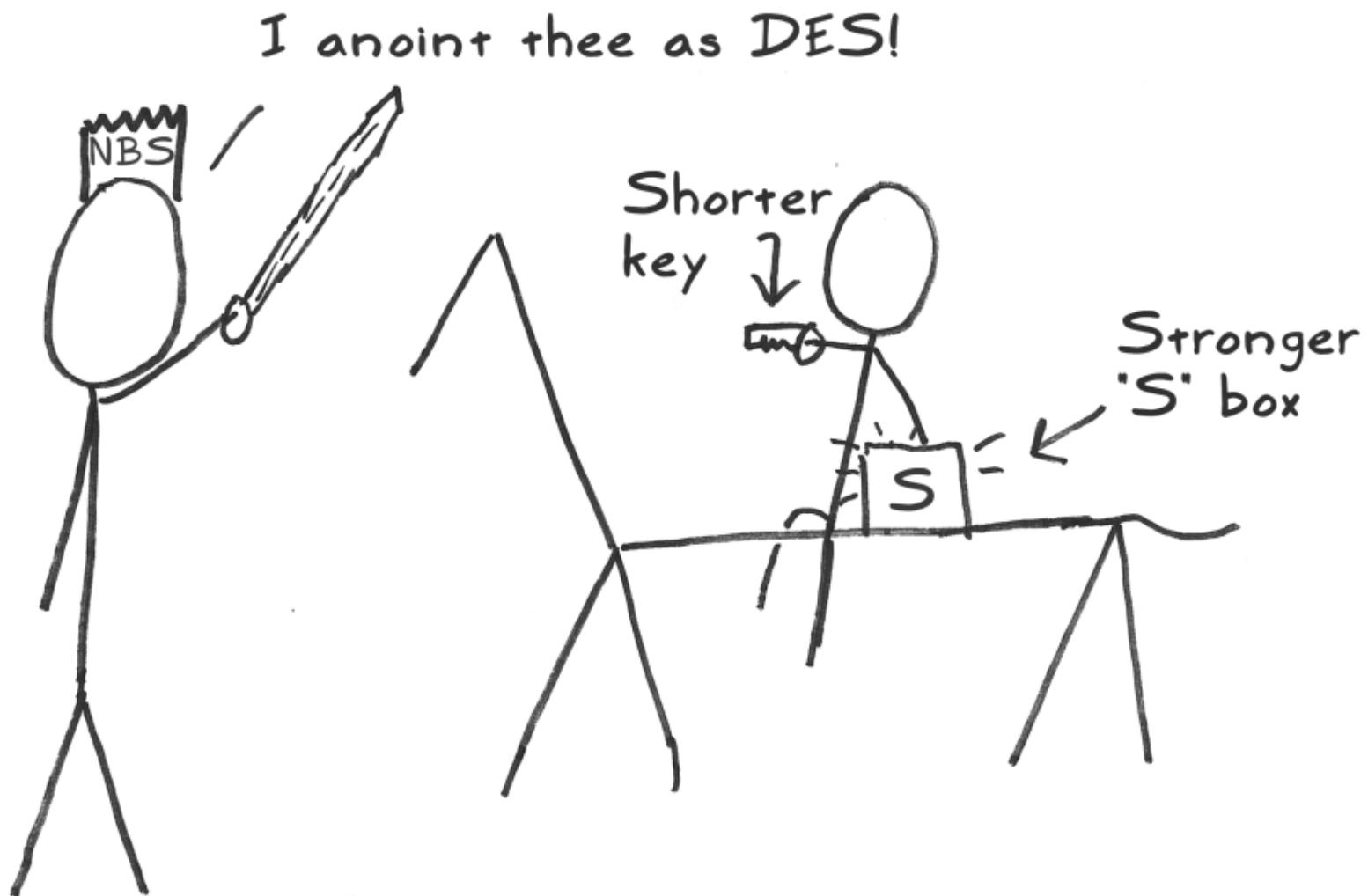
We need a good cipher!



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).



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."

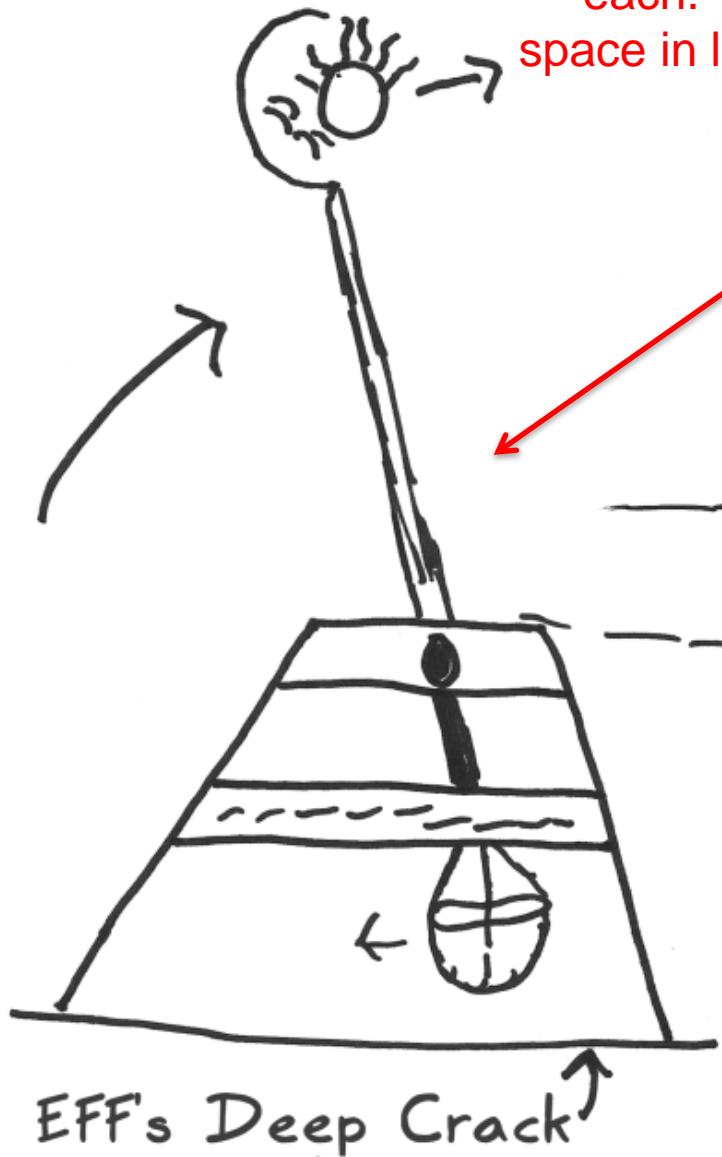


Check out that Feistel network!



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

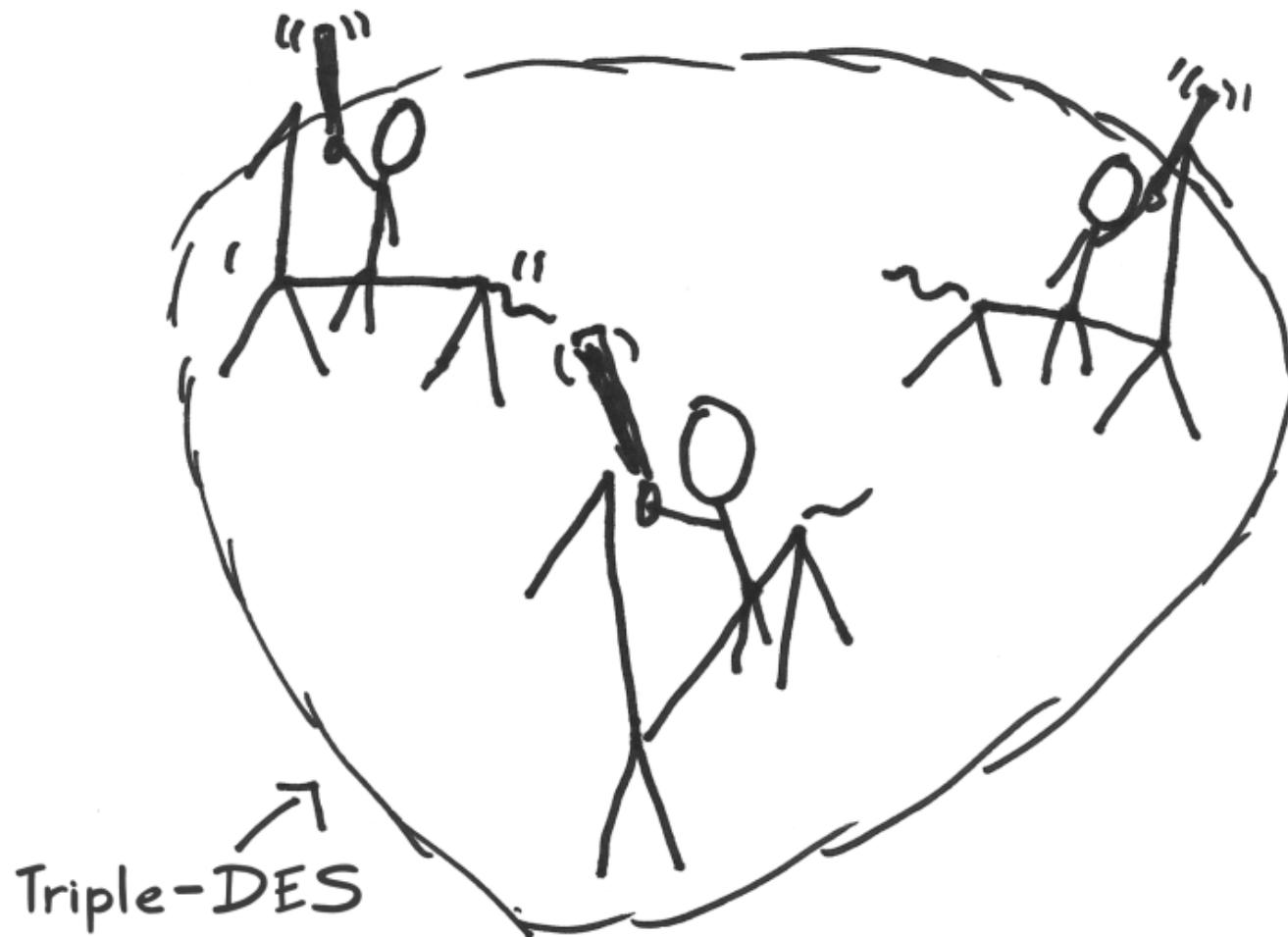
Built by the EFF in 1998 for less than \$250,000. 29 circuit boards with 64 chips in each. Could brute force entire DES key space in less than 9 days. Broke certain keys within 24 hours.



Distributed client  
(kind of like SETI@Home).  
Broke one DES key in 39 days.



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.



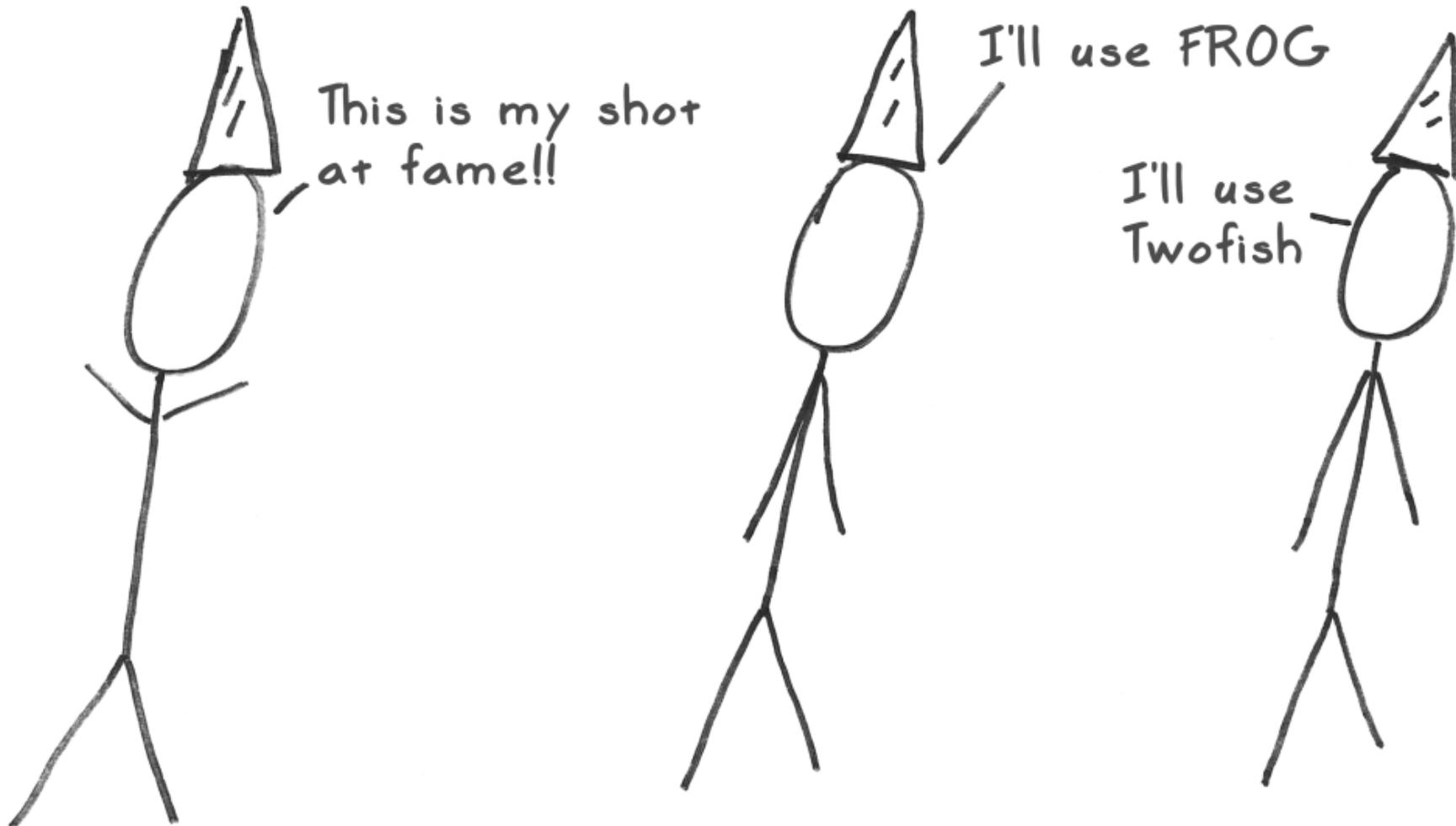
Another decree went out\*...



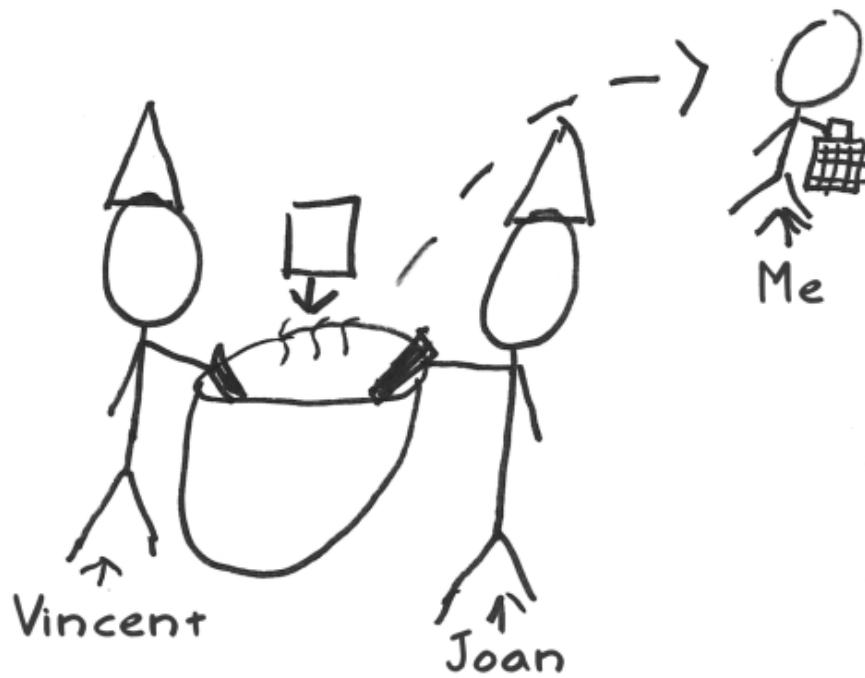
We need something at least as strong as Triple-DES, but it has to be fast and flexible.

\* ~ early 1997

This call rallied the crypto wizards  
to develop something better.

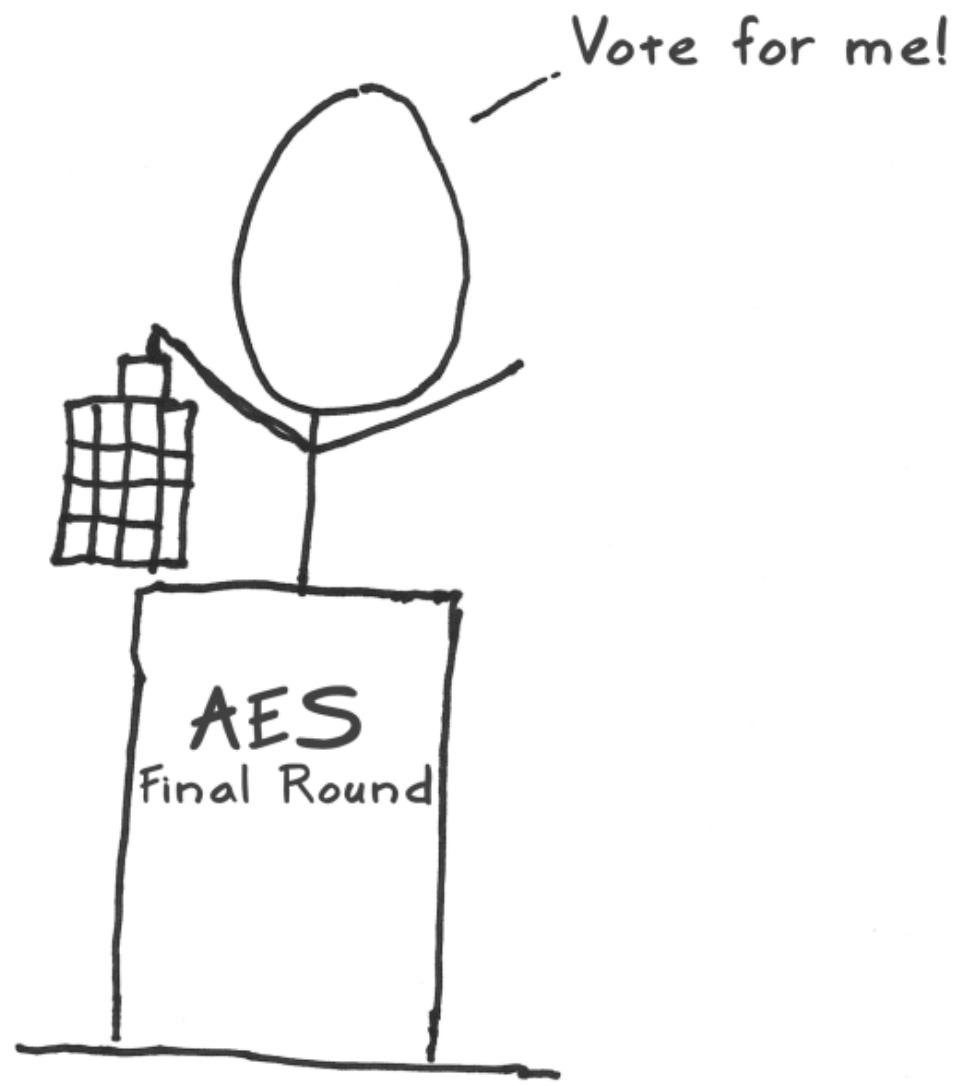


My creators, Vincent Rijmen and Joan Daemen, were among these crypto wizards. They combined their last names to give me my birth name: Rijndael.\*



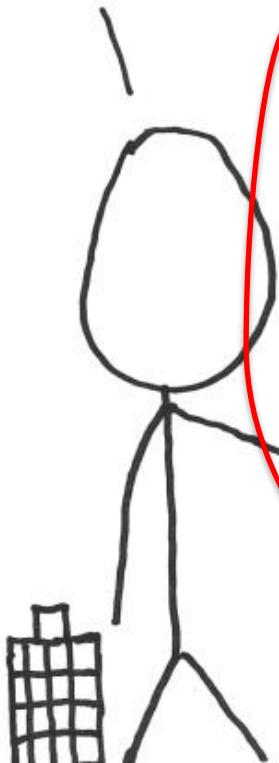
\* That's pronounced "Rhine Dahl" for the non-Belgians out there.

Everyone got together to vote and...



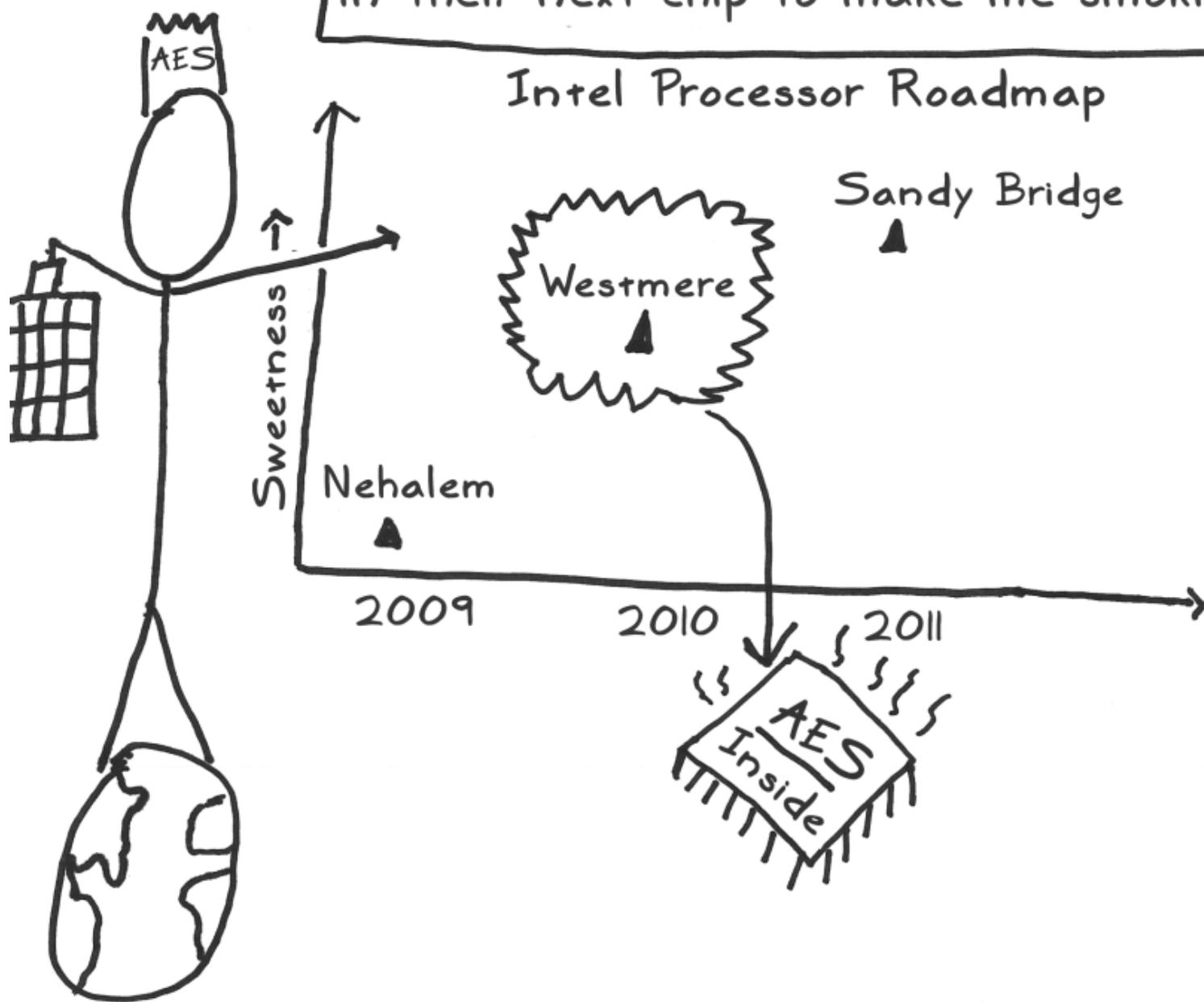
	Rijndael	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	16	14	13	10	9

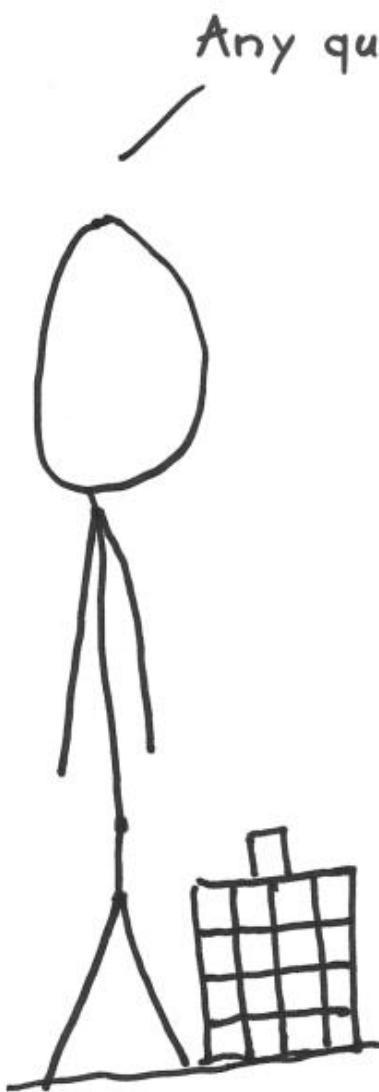
I won!!



Note the importance of performance...

...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!

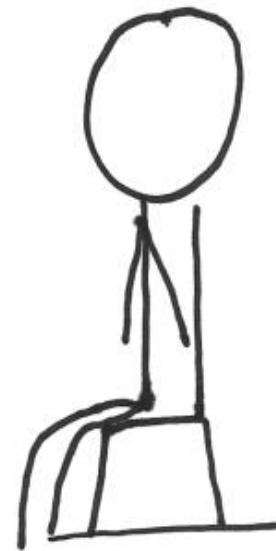




Any questions?



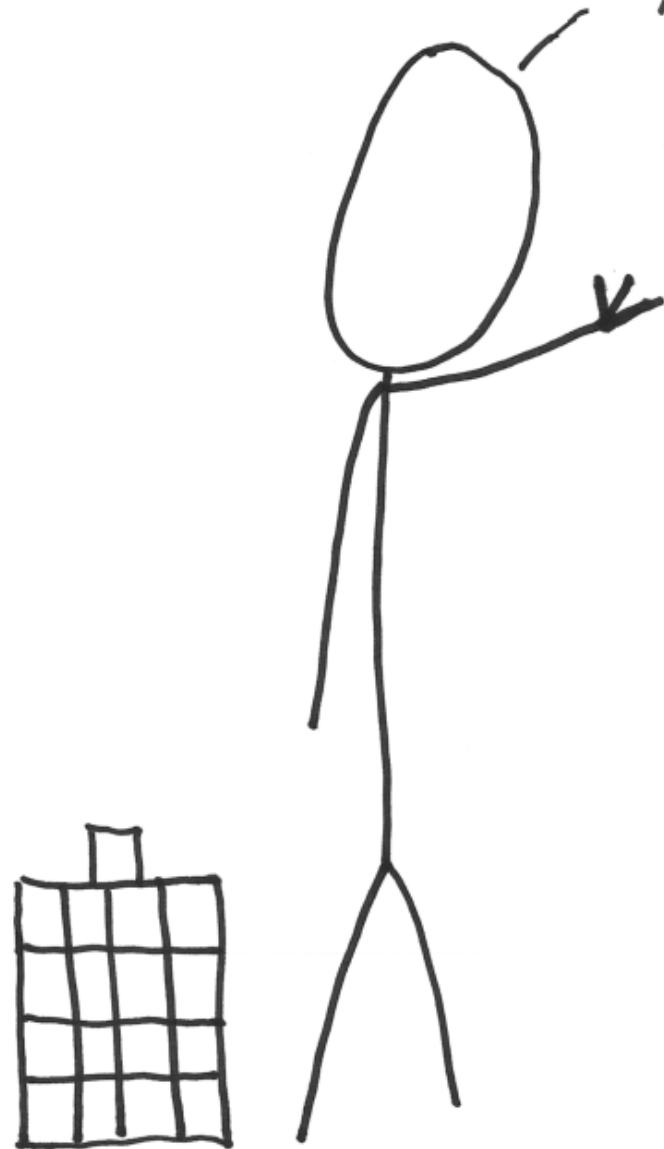
Nice story and  
all, but how does  
crypto work?



Weird. I'm out...

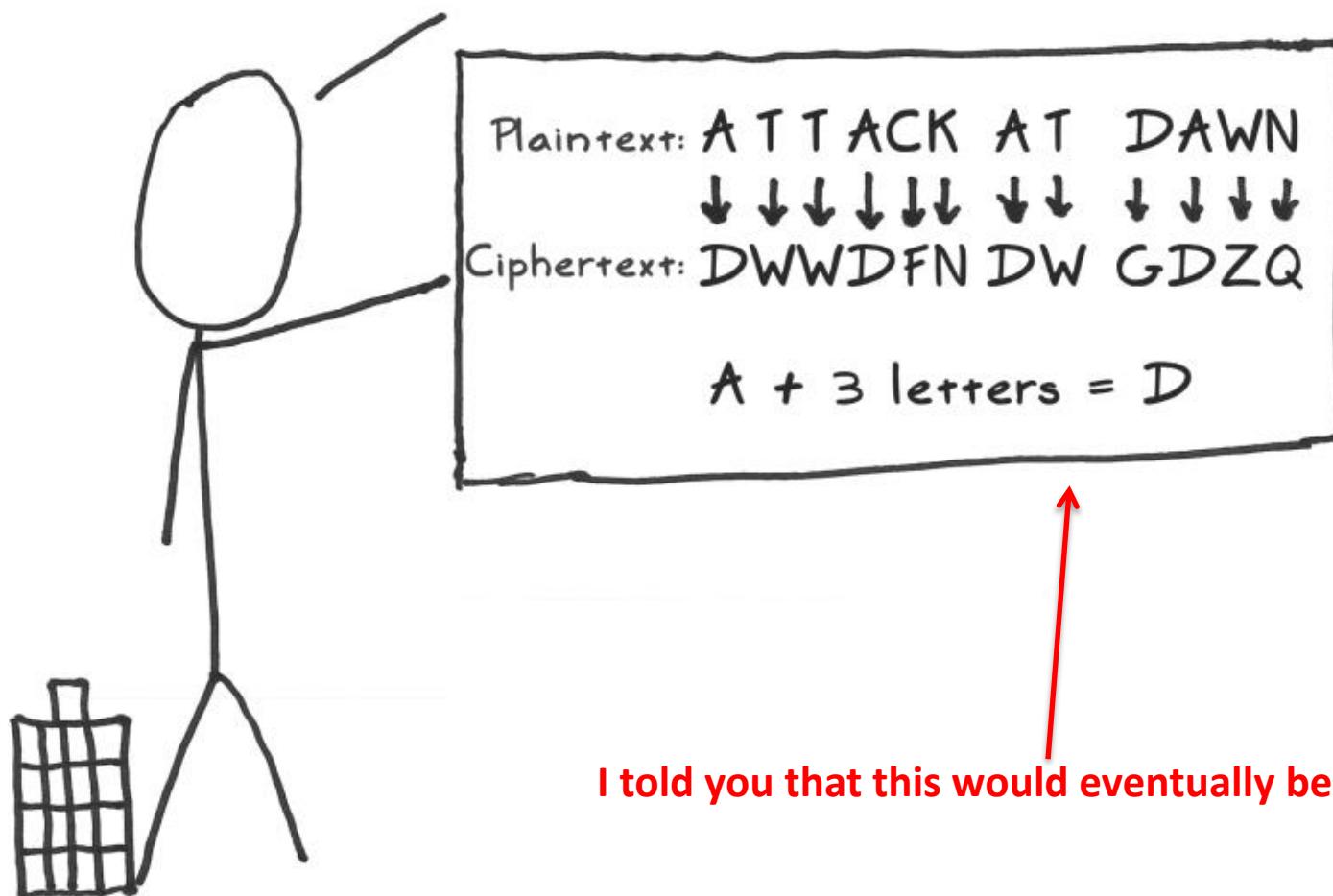
# Act 2: Crypto Basics

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:

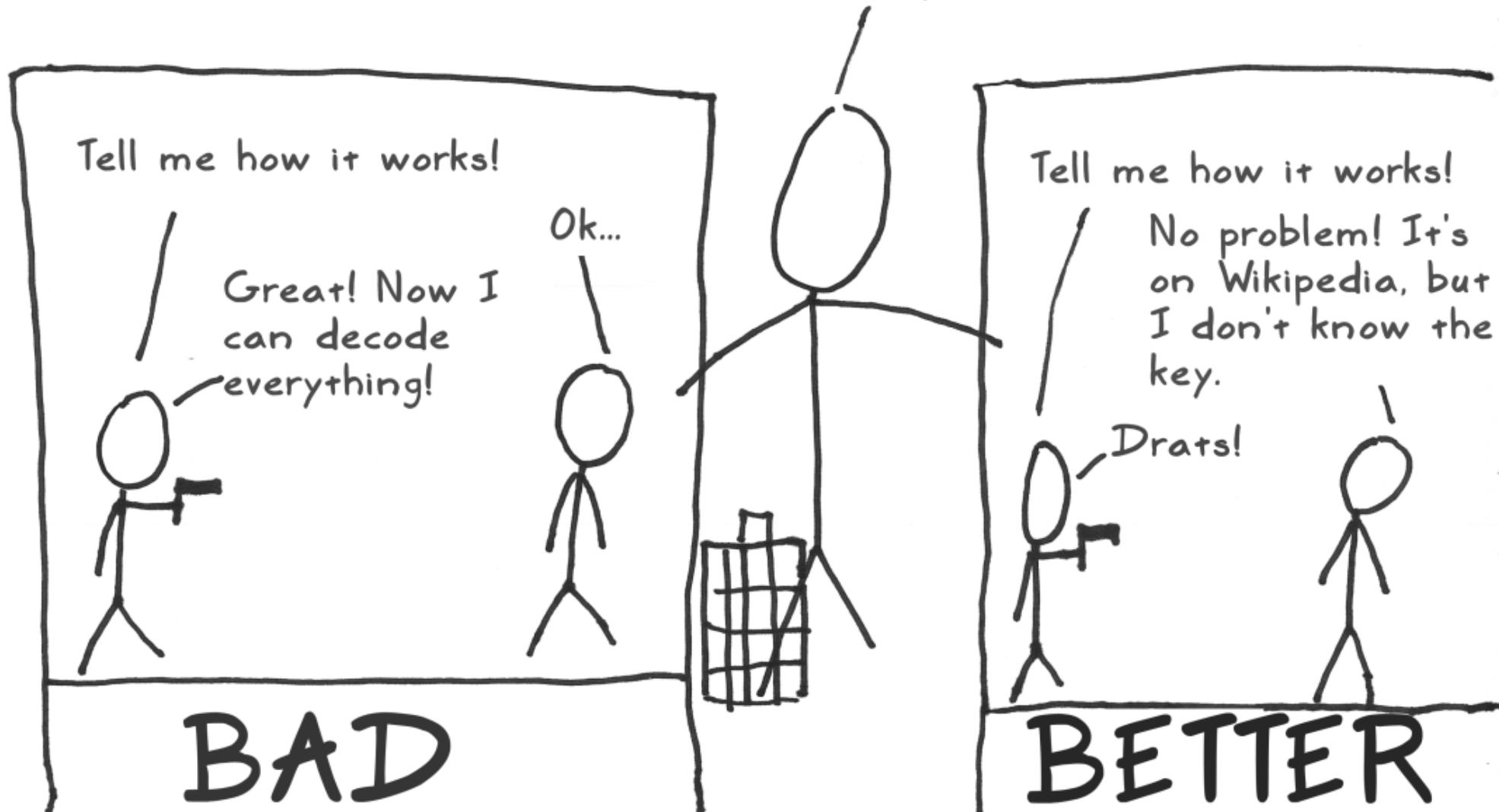


This is the "rail fence" cipher that we talked about last week

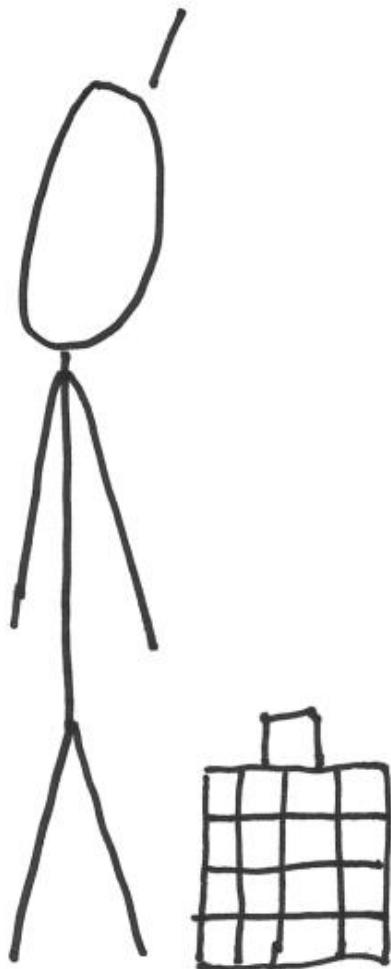
# Big Idea #3: Secrecy Only in the Key

Kerchoffs' 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.



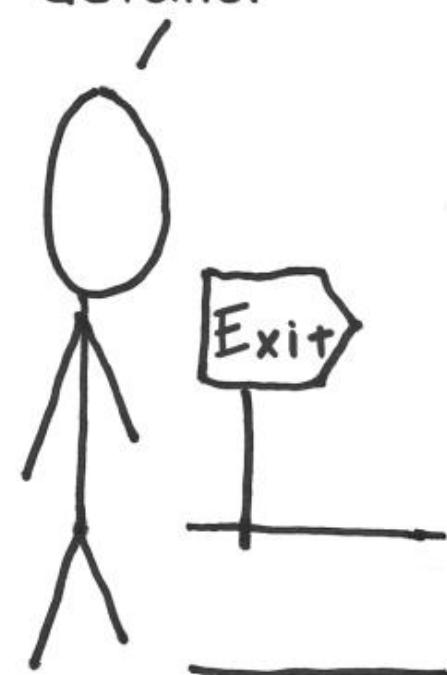
Does that answer  
your question?



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



Details? I  
can't handle  
details!



# Act 3: AES Basics

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.

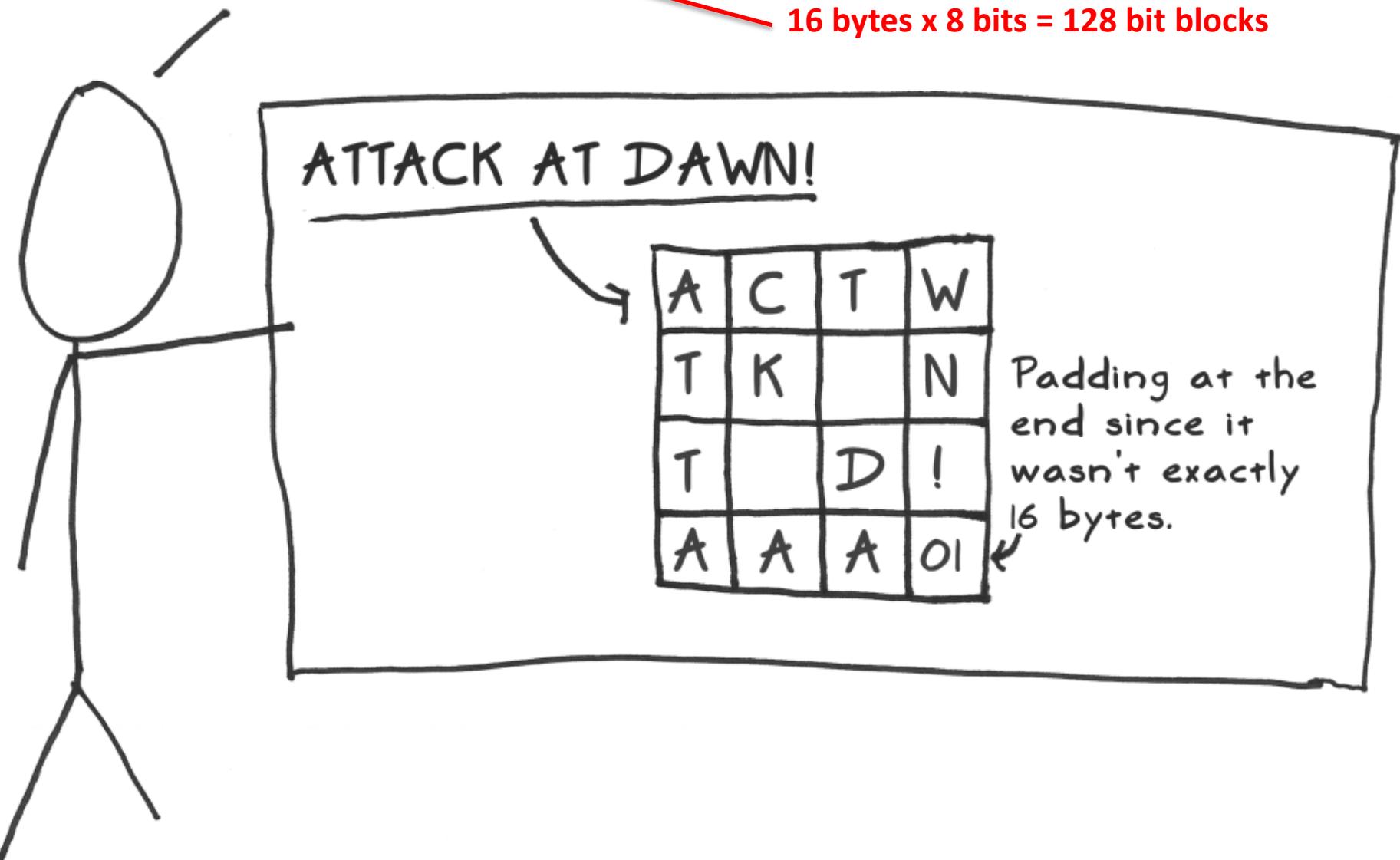


Signature

Date

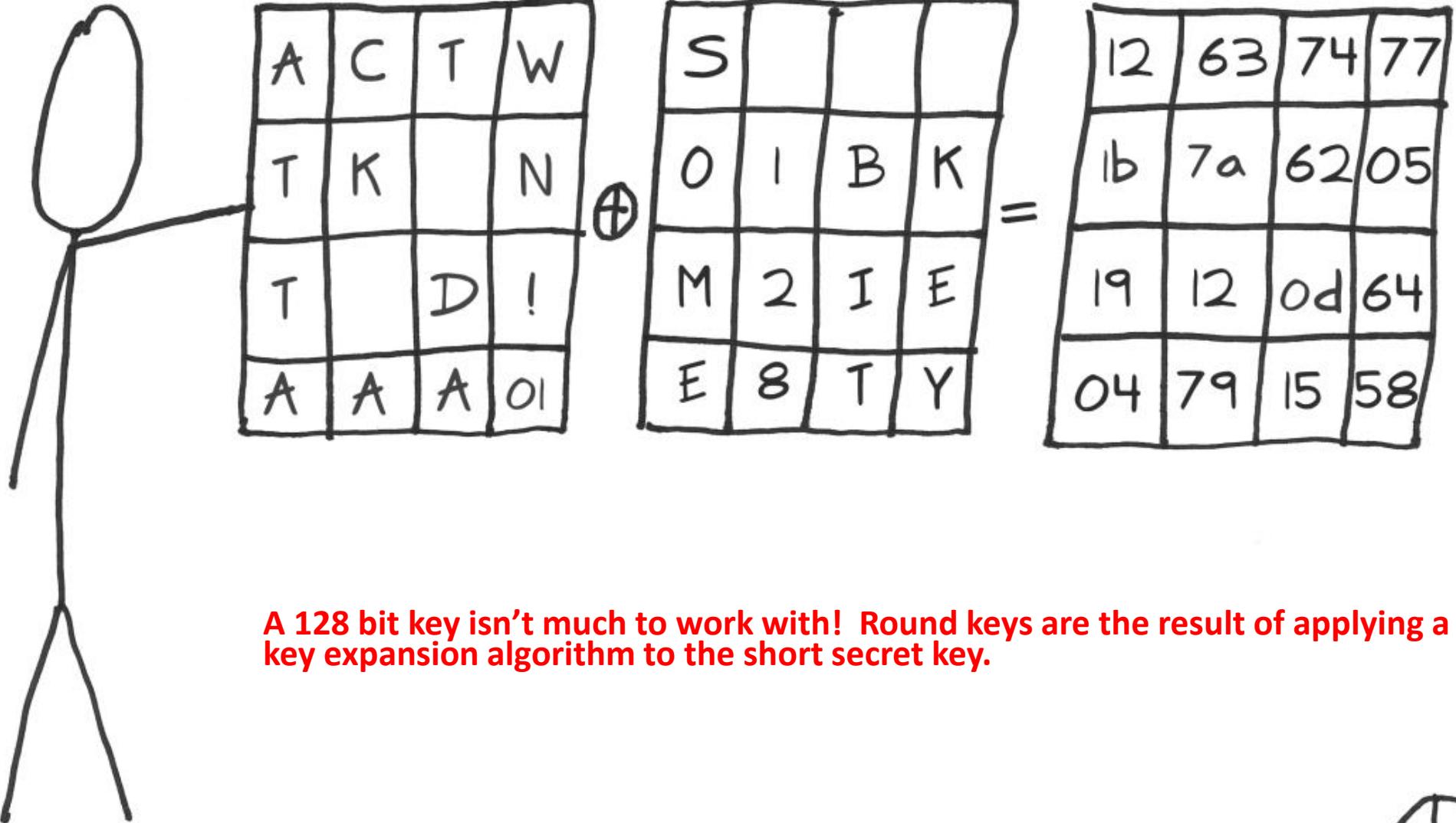
I take your data and load it  
into this 4x4 square.\*

16 bytes x 8 bits = 128 bit blocks



\* 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.

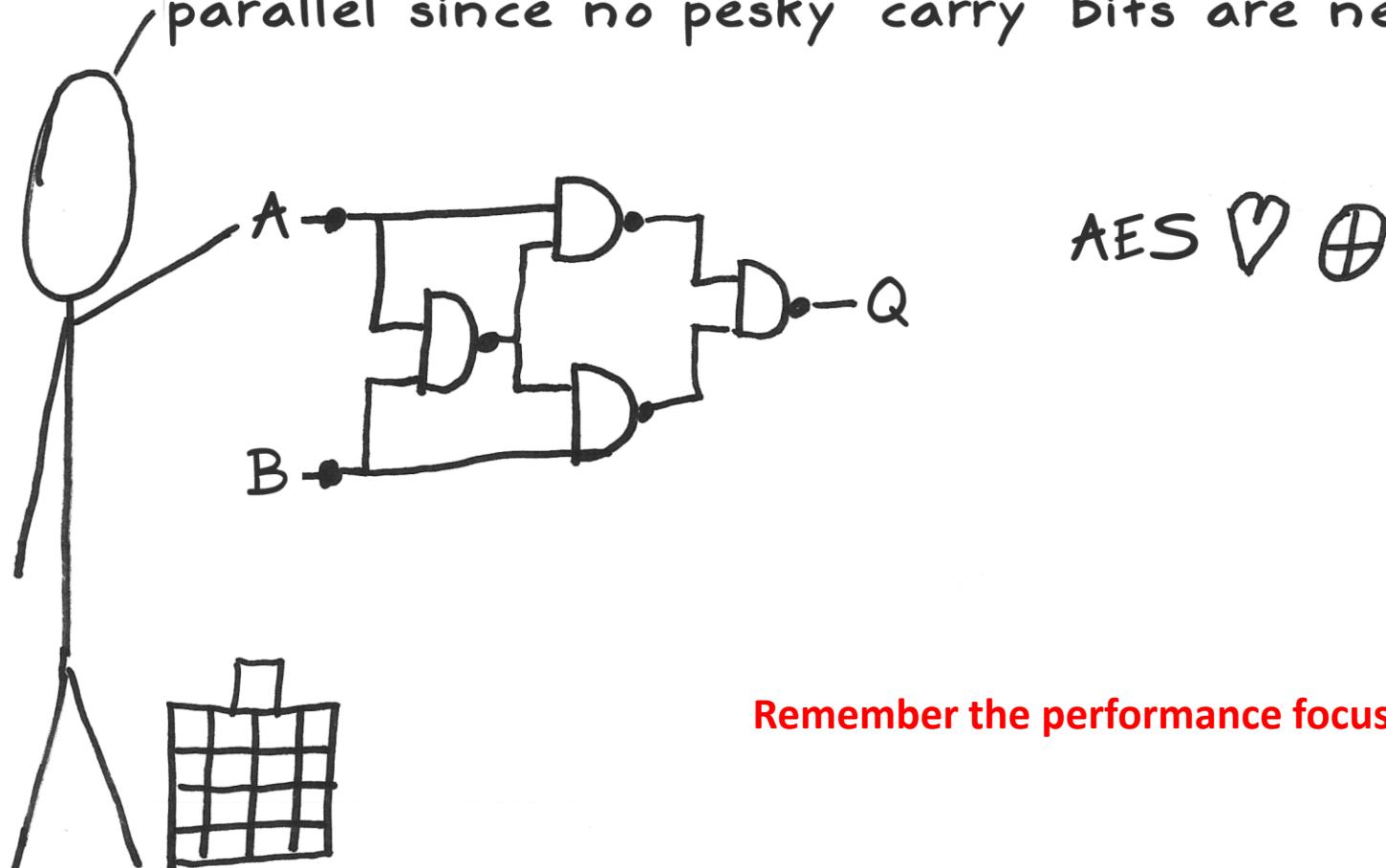


A 128 bit key isn't much to work with! Round keys are the result of applying a key expansion algorithm to the short secret key.



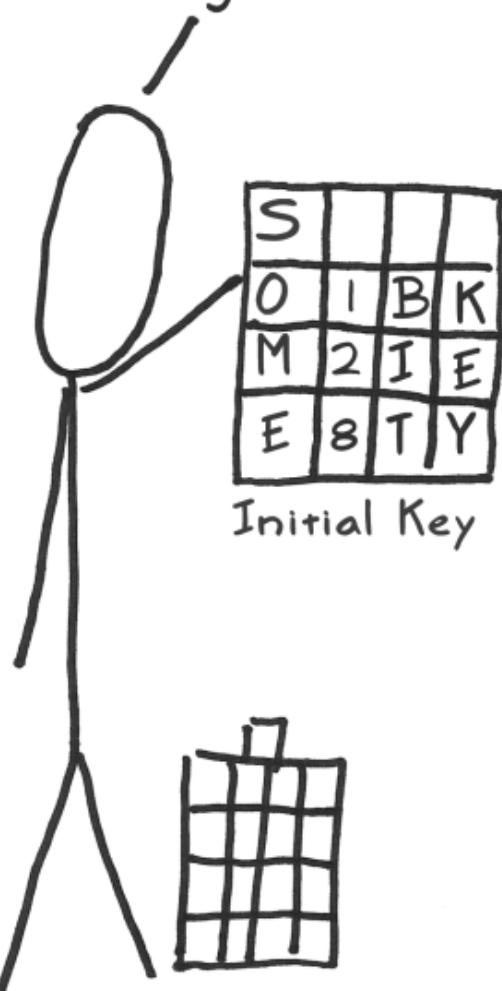
## A Tribute to XOR

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

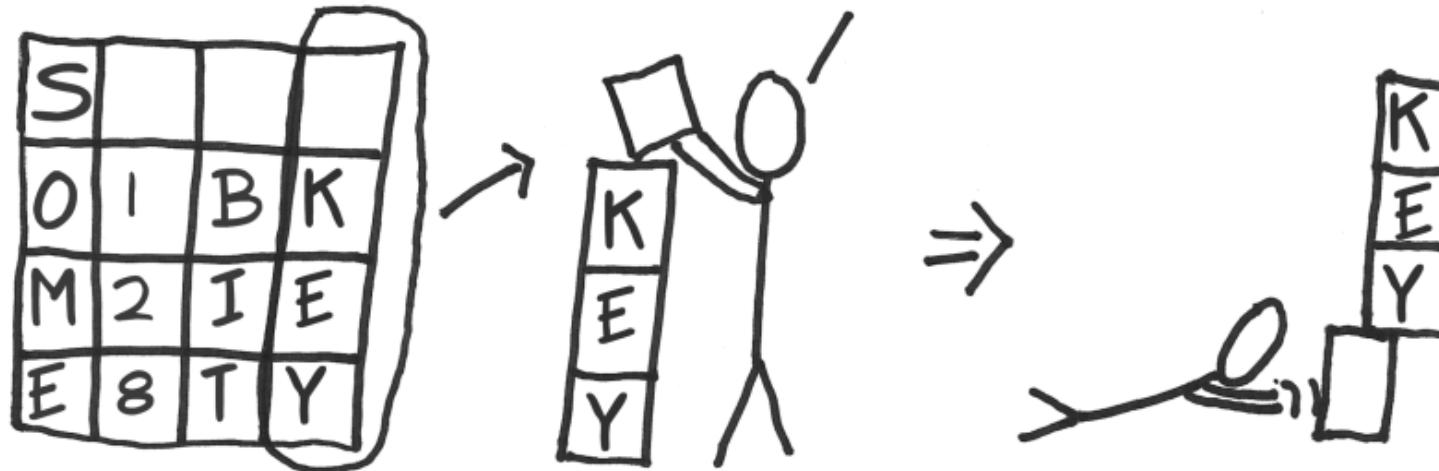
3e	98	38	ec
a2	7a	dc	19
22	5f	25	fc
a7	4a	4f	49

#10

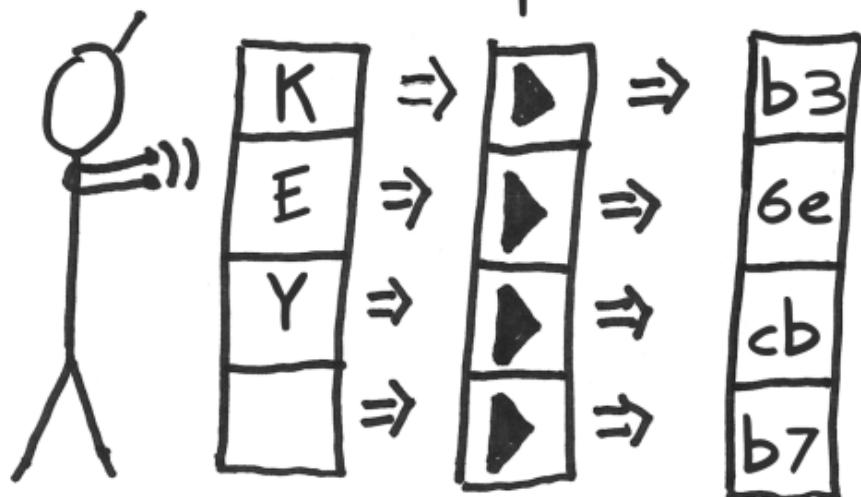
\* By far, most complaints against AES's design focus on this simplicity.

## Key Expansion: Part 2a

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

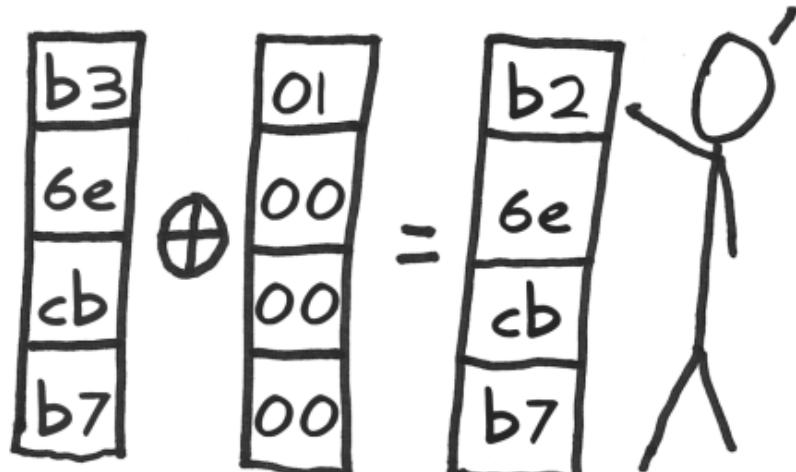


- ② Next, I run each byte through a substitution box that will map it to something else:

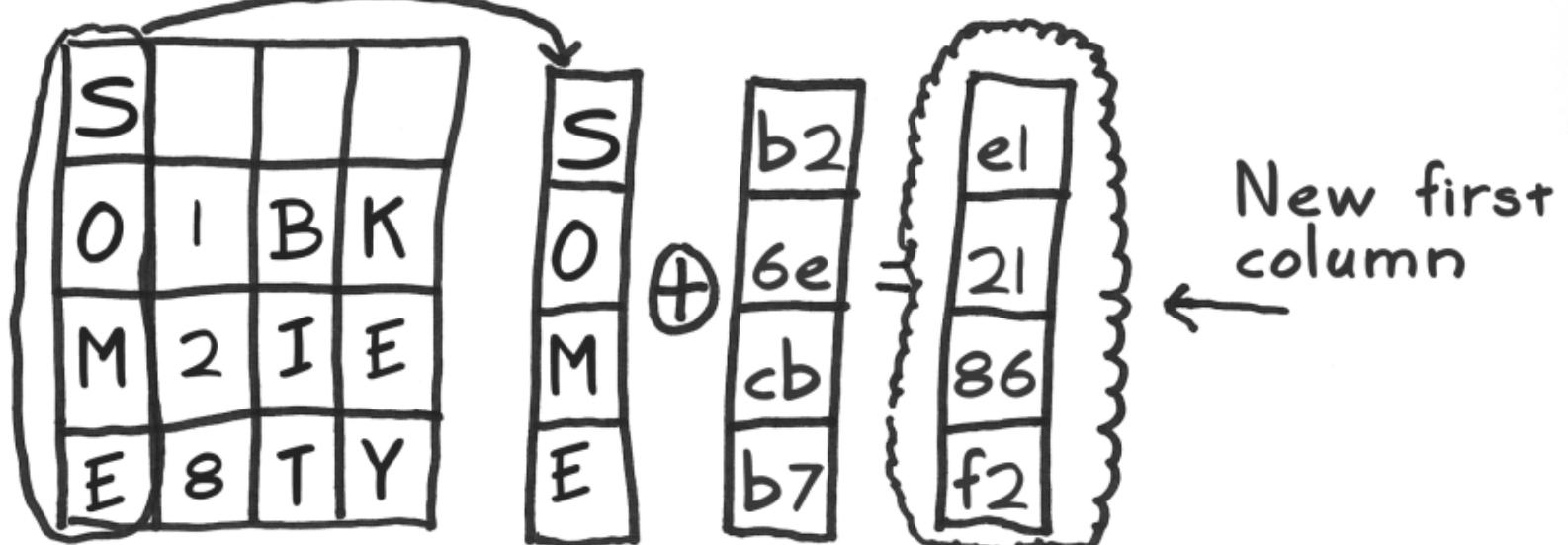


## Key Expansion: Part 2b

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

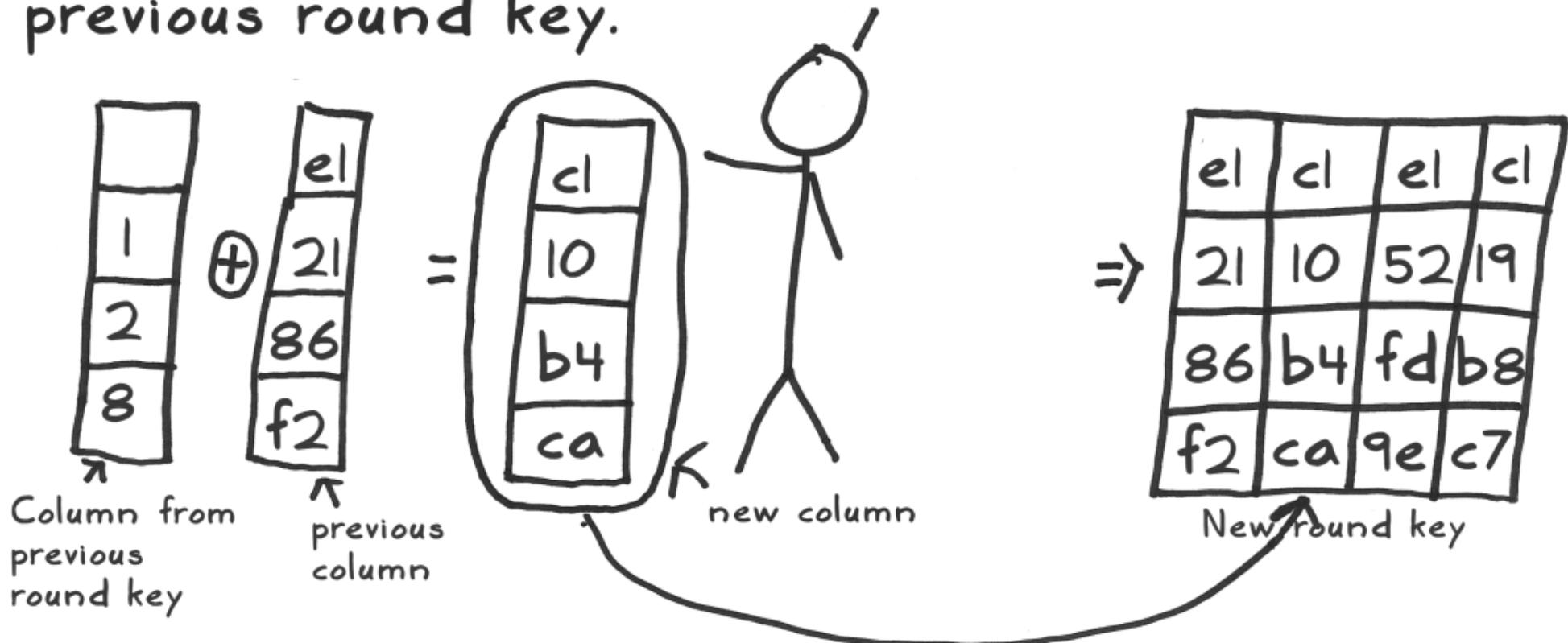


- ④ Finally, I xor it with the first column of the previous round key:



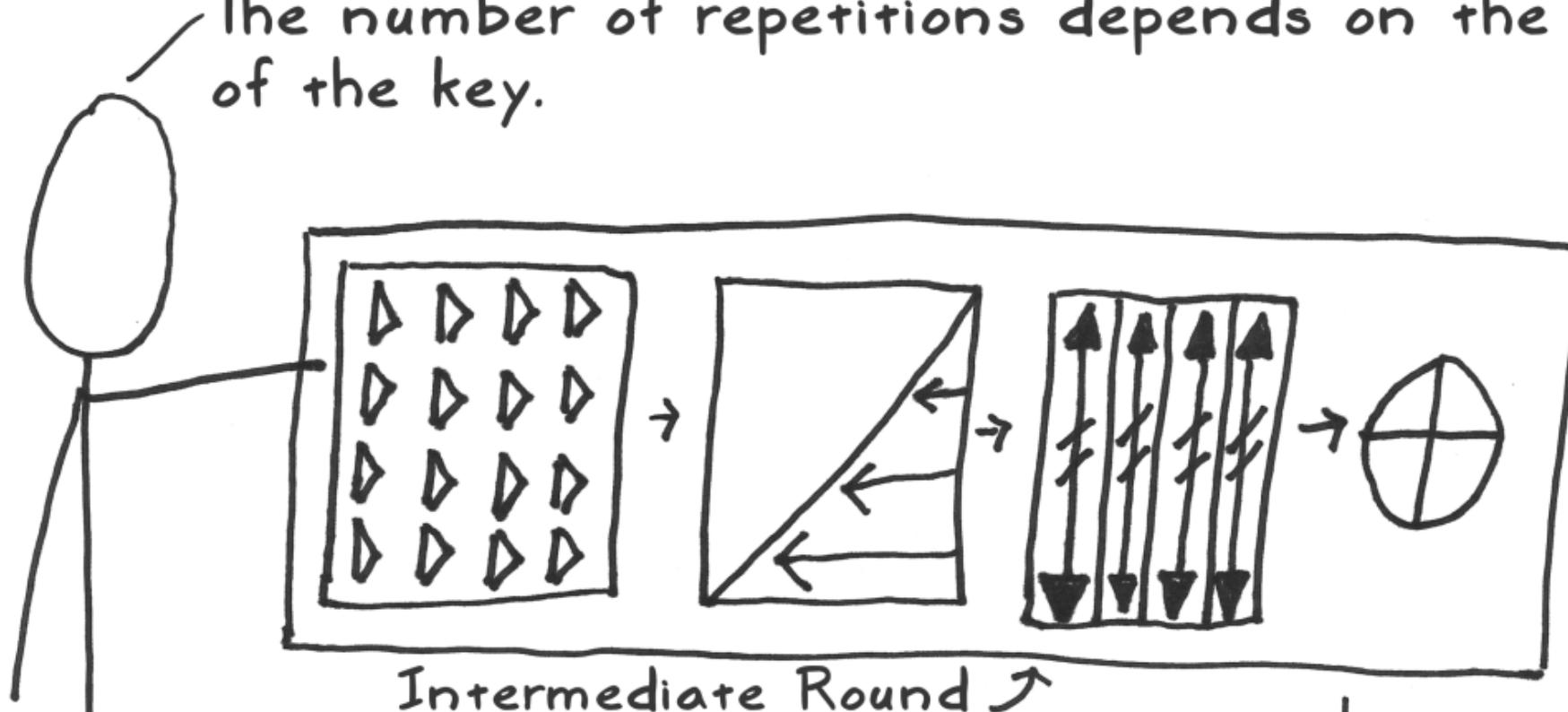
## Key Expansion: Part 3

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

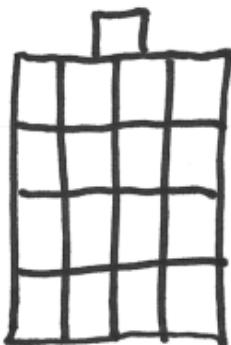


\*Note that 256 bit keys are slightly more complicated.

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.

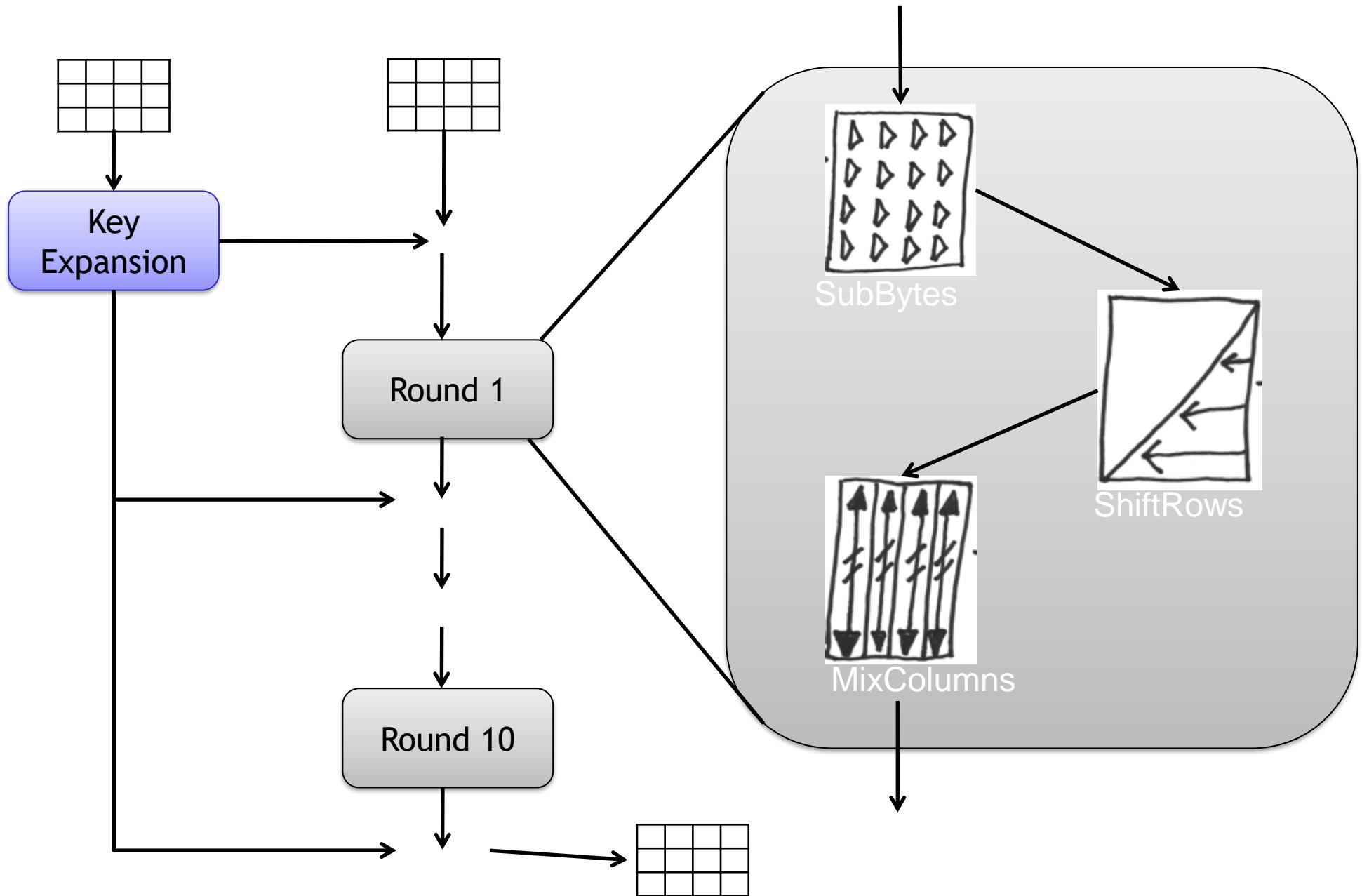


Intermediate Round ↗



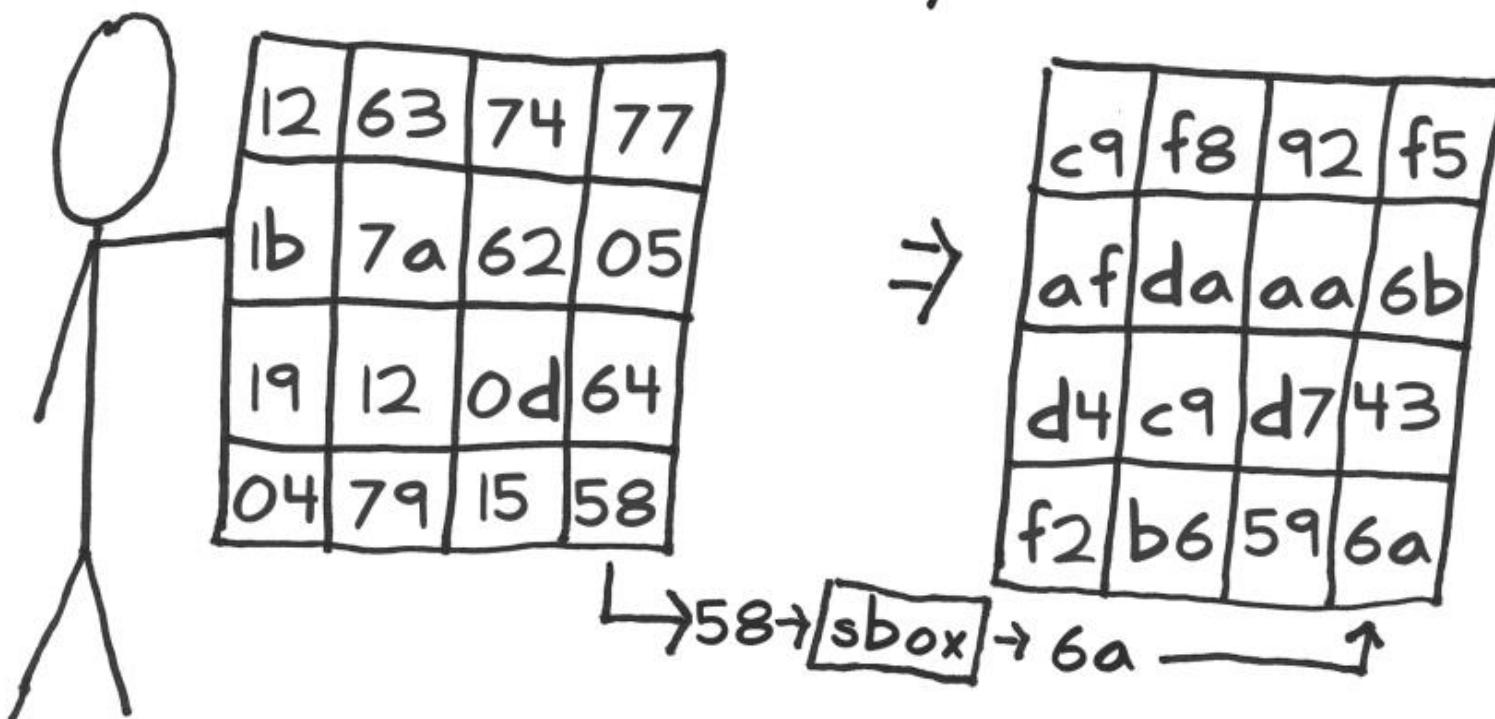
Round Repetitions	Key Size
9	128
11	192
13	256

# AES Round Diagram

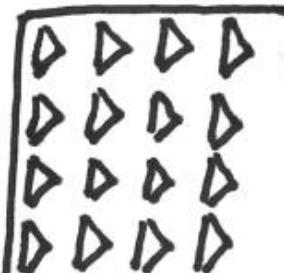


## Applying Confusion: Substitute Bytes

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:



Y Denotes  
"confusion"



# The S-Box is just a big lookup table!

		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

Figure 7. S-box: substitution values for the byte  $xy$  (in hexadecimal format).

# Applying Diffusion, Part I: Shift Rows

Next I shift the rows to the left

Hiiiii yaah!

	c9	fb	92	f5
	af	da	aa	6b
d4	c9	d7	43	
f2	b6	59	6a	



...and then wrap them around the other side

c9	fb	92	f5
da	aa	6b	af
d7	43	d4	c9
6a	f2	b6	59



Note: Every column will now contain entries that were previously in each other columns!

Denotes "permutation"



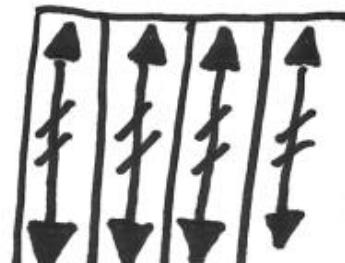
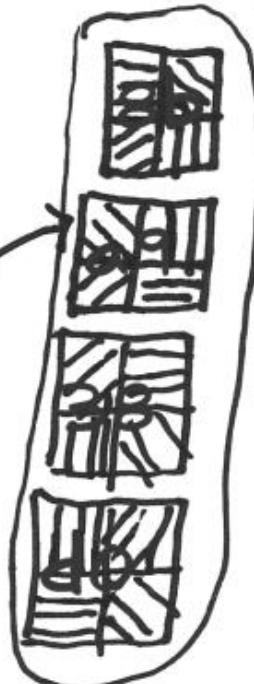
# Applying Diffusion, Part 2: Mix Columns

c9	fb	92	f5
20	aa	6b	af
d7	43	44	fa
6a	f2	b6	59

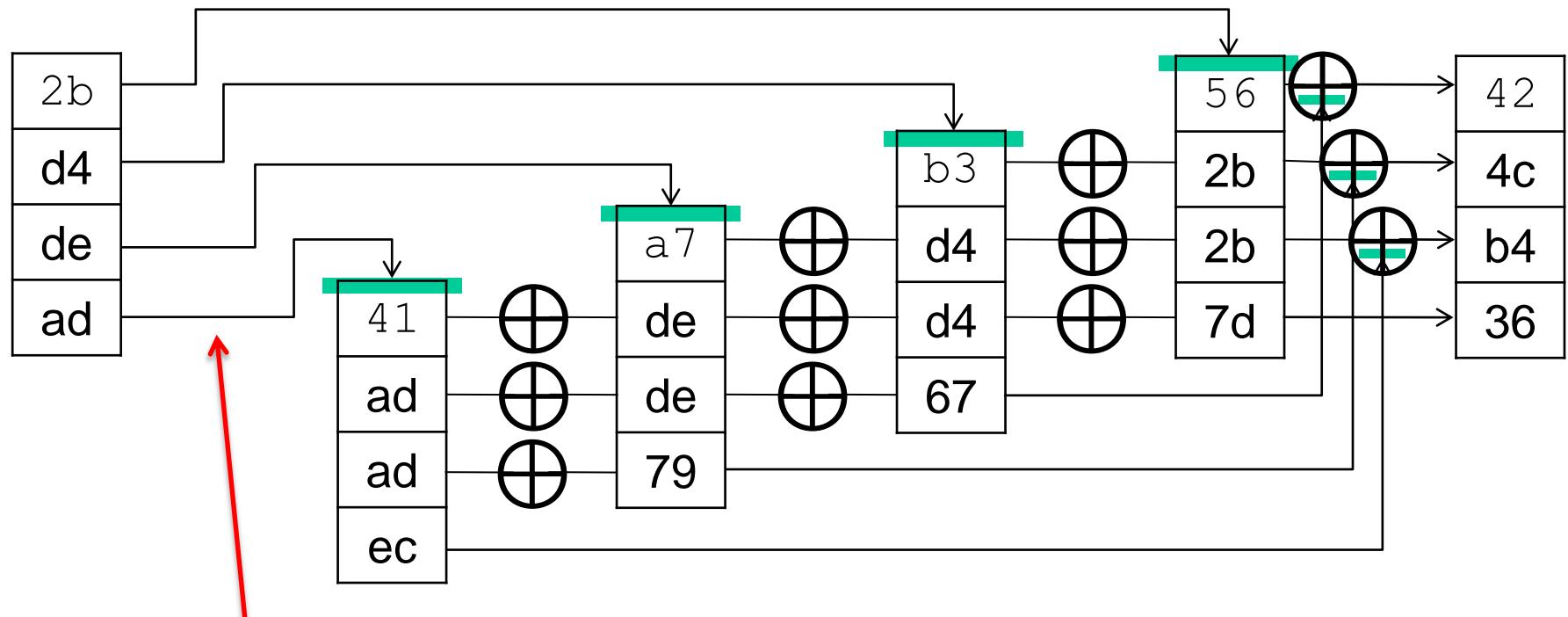
I take each column and mix up the bits in it.

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

Data begins to spill across rows



# MixColumns is essentially table lookups and XORs



Each line involves a table lookup

# 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

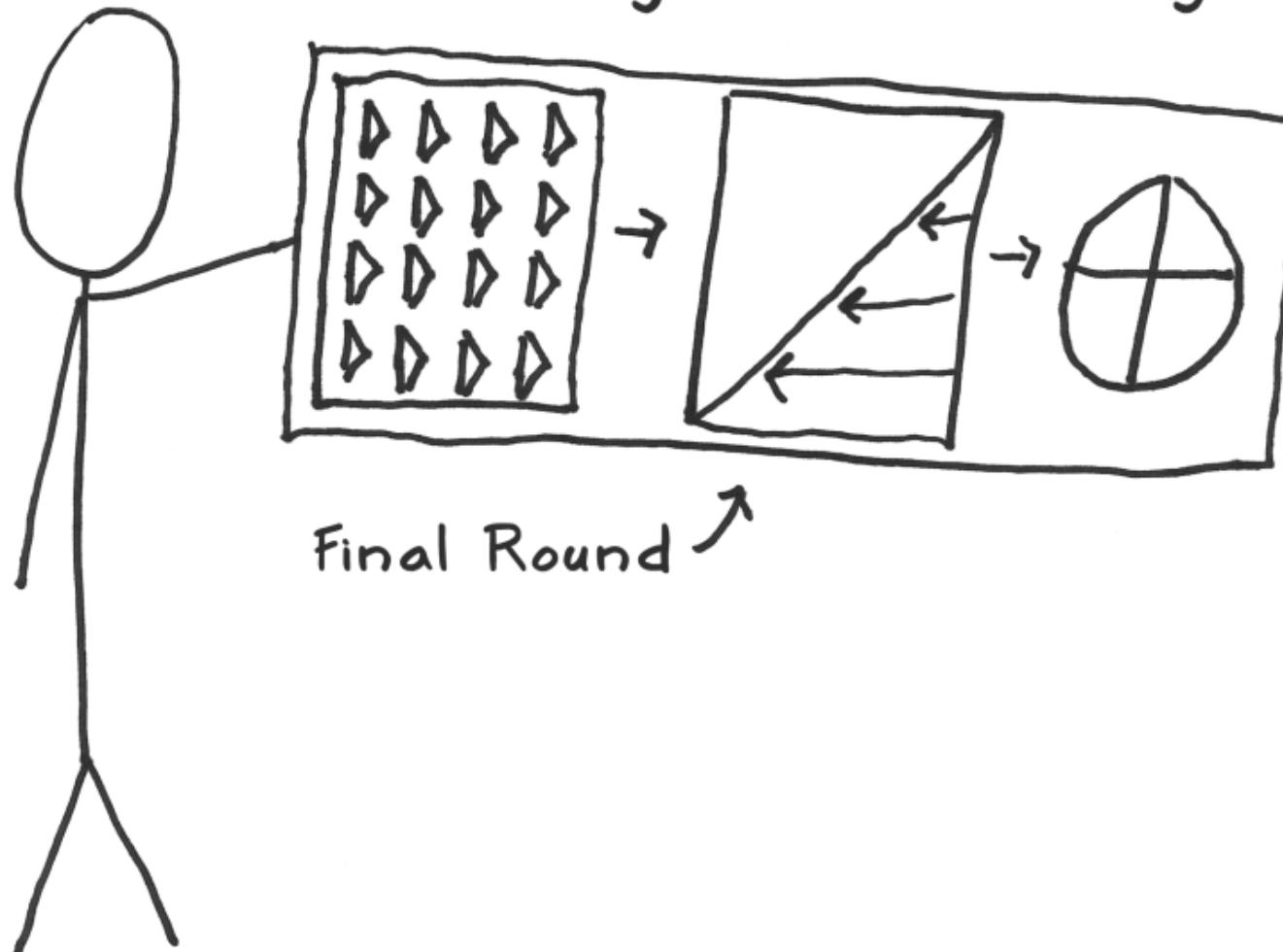


a0	78	01	4a
4f	93	c7	b0
9e	6e	76	80
6b	ca	fb	17

$$d0 \oplus c7 = 17$$

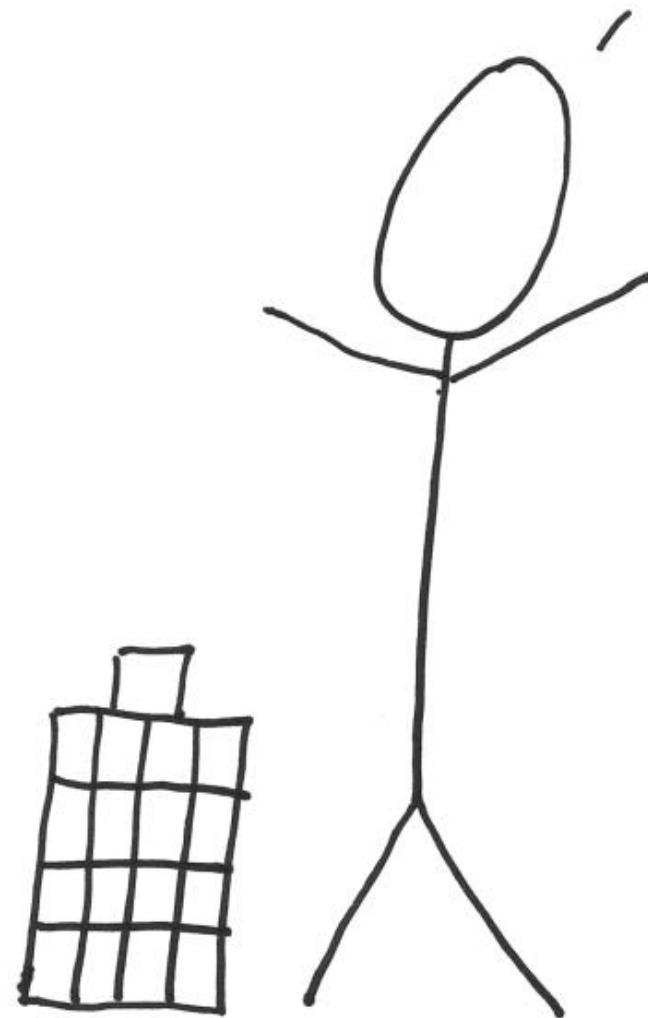


In the final round, I skip the "Mix Columns" step since it wouldn't increase security\* and would just slow things down:

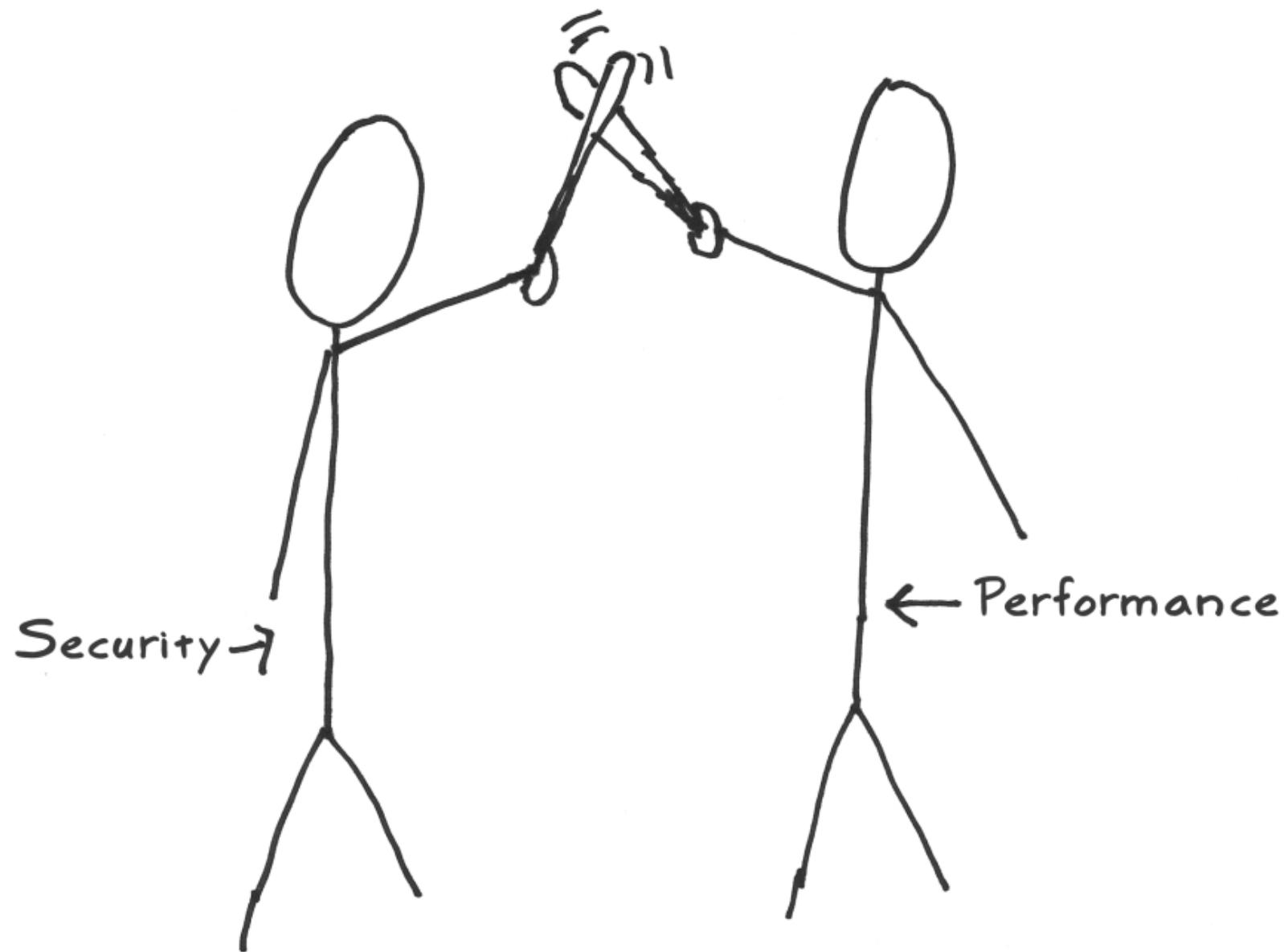


\*The diffusion it would provide wouldn't go to the next round.

...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!

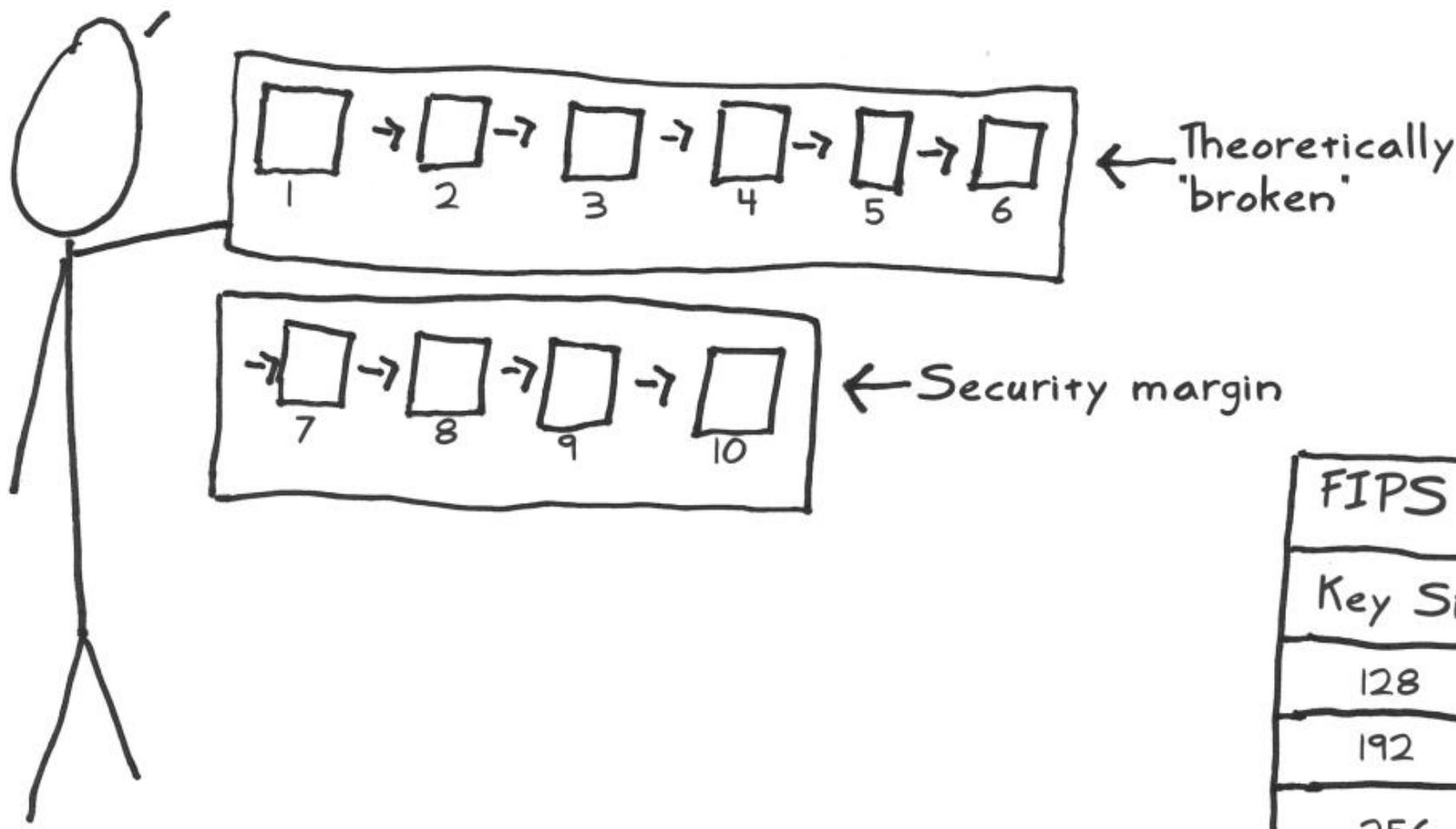


Determining the number of rounds always involves several tradeoffs.



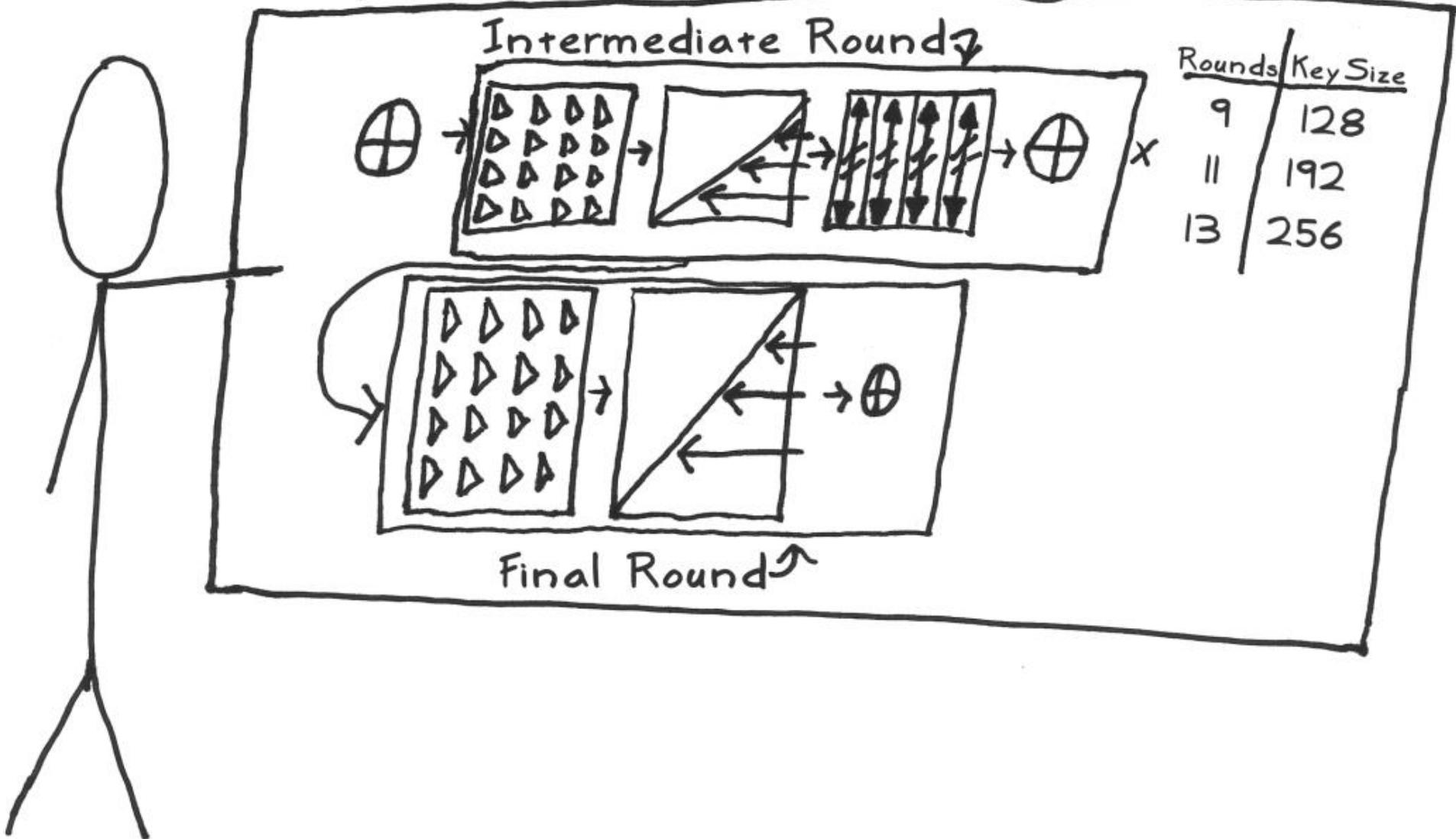
"Security always comes at a cost to performance" - Vincent Rijmen

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.'

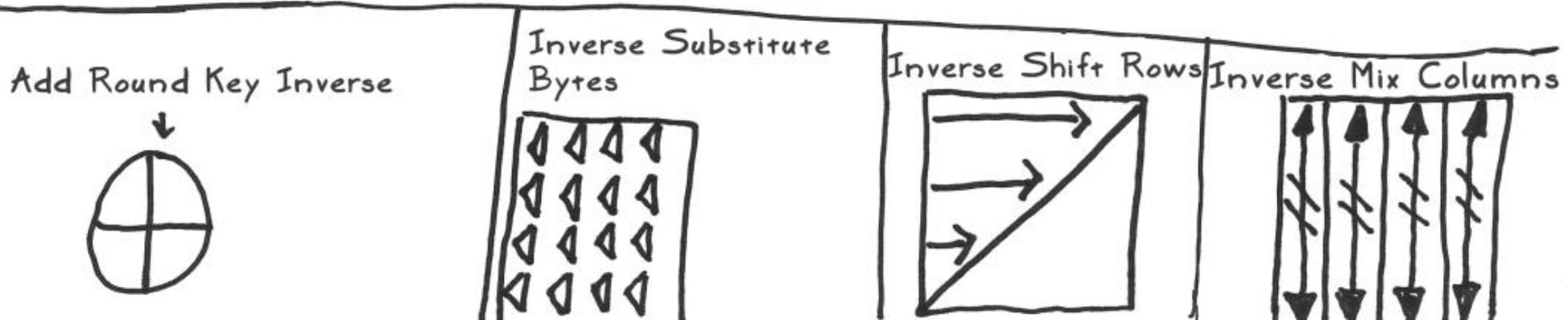
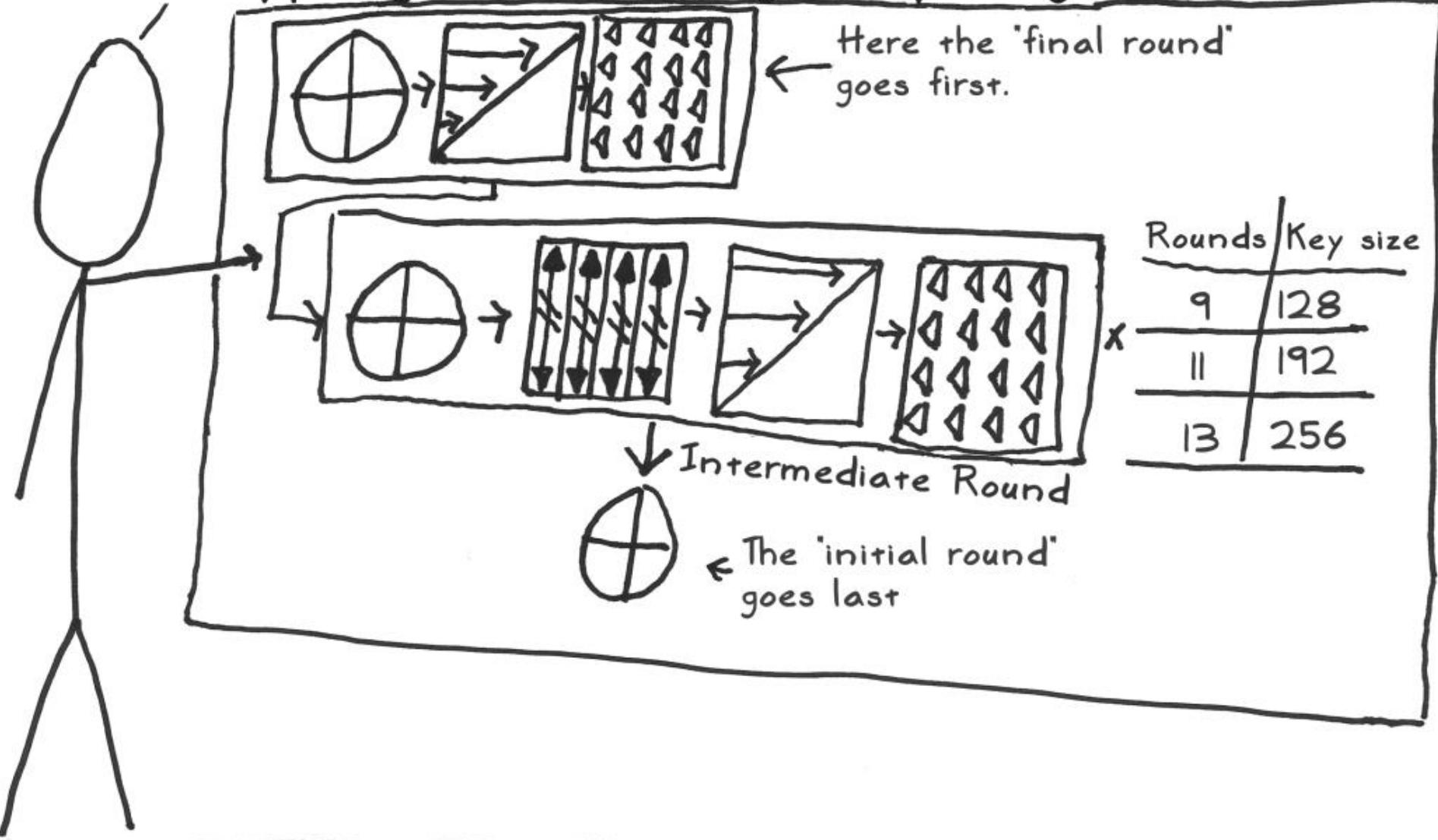


FIPS 197 Spec	
Key Size	Rounds
128	10
192	12
256	14

So in pictures, we have this:



# Decrypting means doing everything in reverse



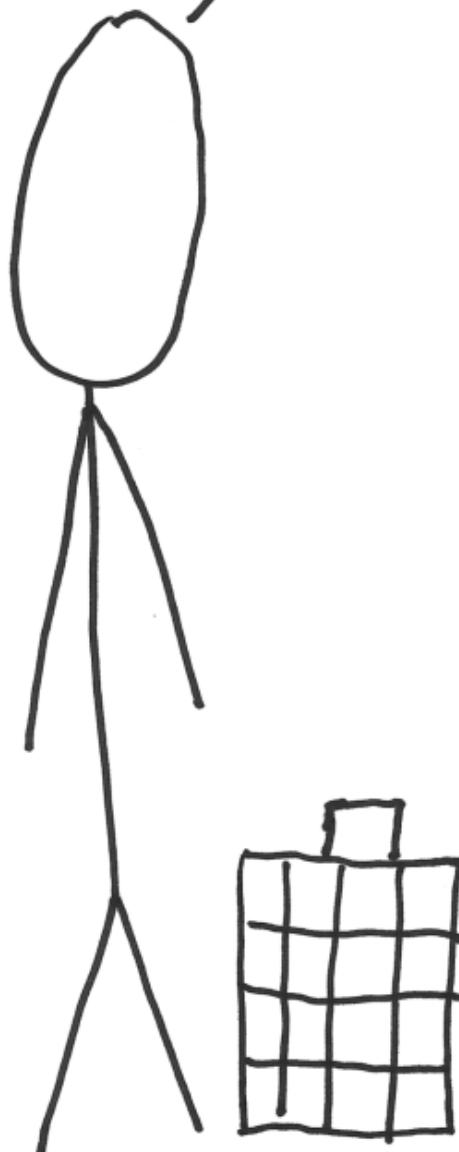
Make sense? Did that  
answer your question?



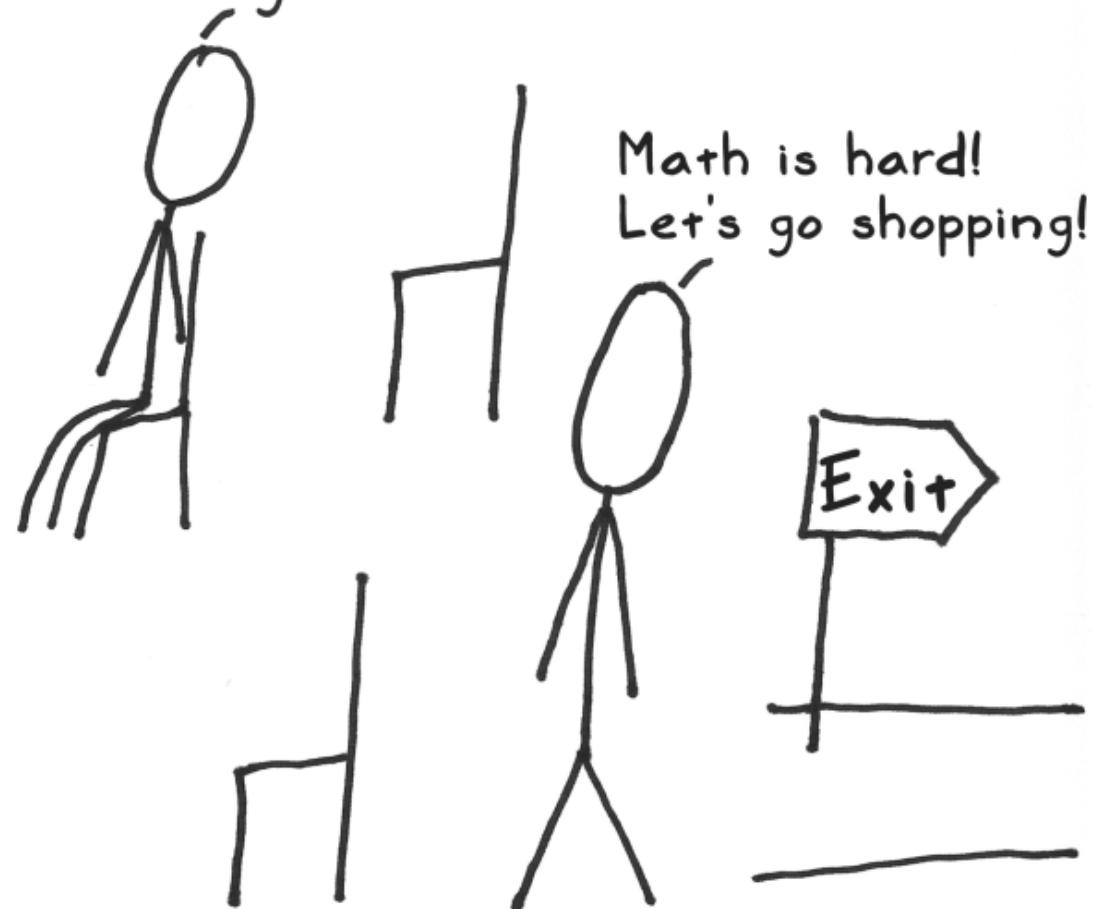
Almost...except you just  
waved your hands and  
used weird analogies.  
What really happens?



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...



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

square

the unknown

multiplication

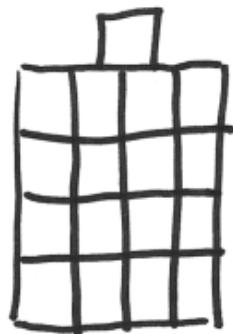
addition

polynomial

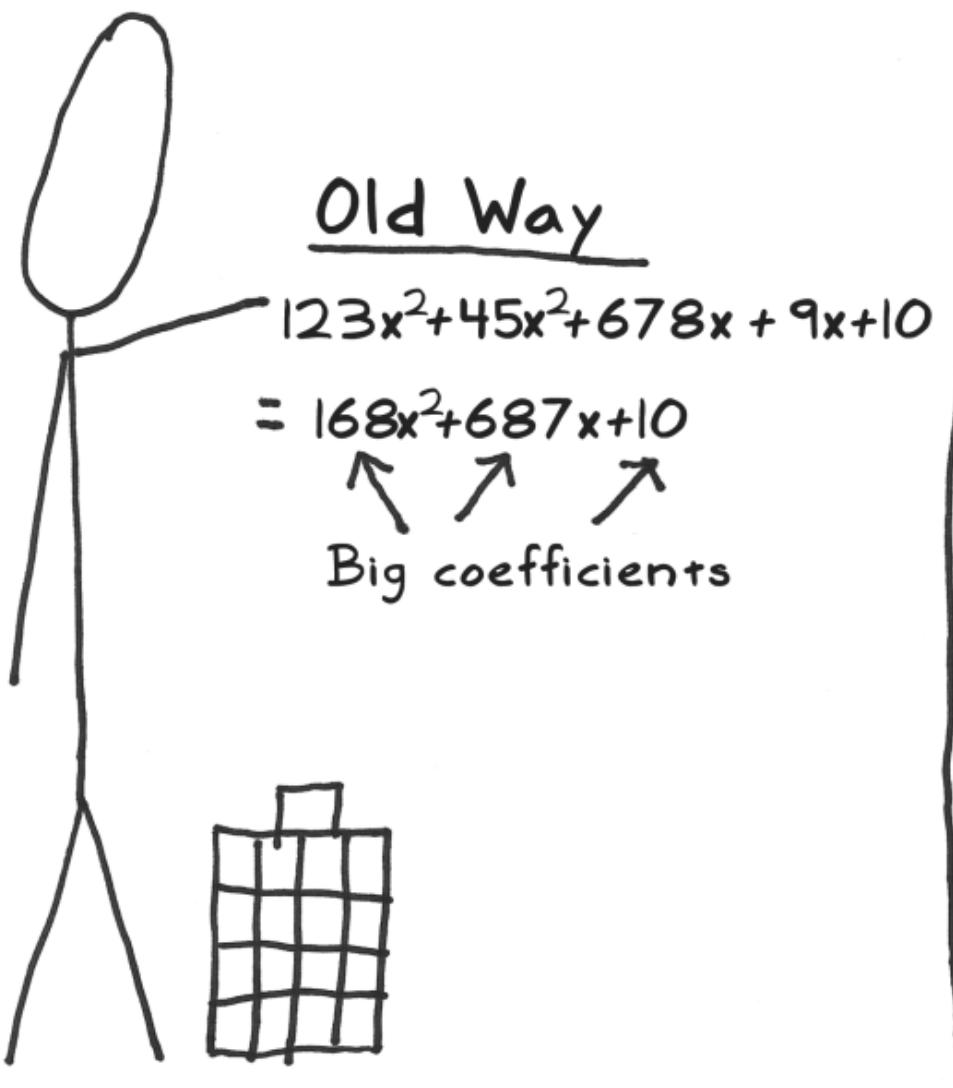
degree

coefficient

The diagram illustrates the expansion of a binomial square. It shows  $(x + 1)^2$  being expanded into  $(x+1) \cdot (x+1)$ , which then simplifies to  $x^2 + x + x + 1$ . This result is then simplified to  $x^2 + 2x + 1$ . Various parts of the equation are labeled: 'square' points to the squared term  $(x+1)^2$ ; 'the unknown' points to the variable  $x$ ; 'multiplication' points to the product  $(x+1) \cdot (x+1)$ ; 'addition' points to the sum  $x + x$ ; 'polynomial' points to the final result  $x^2 + 2x + 1$ ; 'degree' points to the highest power of  $x$  (the degree); and 'coefficient' points to the numerical factor in front of the  $x$  term.



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:



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

$$x^2 \oplus x^2 \oplus x^2 = (x^2 \oplus x^2) \oplus x^2$$
$$= 0 \oplus x^2$$
$$= x^2$$

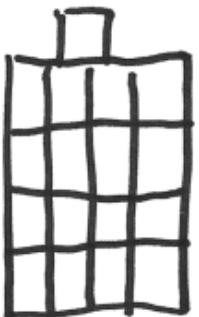
i.e., Coefficients drawn from  $\mathbb{Z}_2$ . You need material from CS0441 again!

\*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?

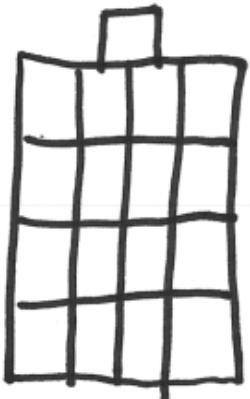
$$\begin{aligned} & (x^7 + x^5 + x^3 + x) \cdot (x^6 + x^4 + x^2 + 1) \\ &= x^{7+6} + x^{7+4} + x^{7+2} + x^{7+0} + x^{5+6} + x^{5+4} + x^{5+2} + x^{5+0} \\ &\quad + x^{3+6} + x^{3+4} + x^{3+2} + x^{3+0} + x^{1+6} + x^{1+4} + x^{1+2} + x^{1+0} \\ &= x^{13} + x^{11} + x^9 + x^7 + x^9 + x^7 + x^5 + x^9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

↗  
Big and yucky!



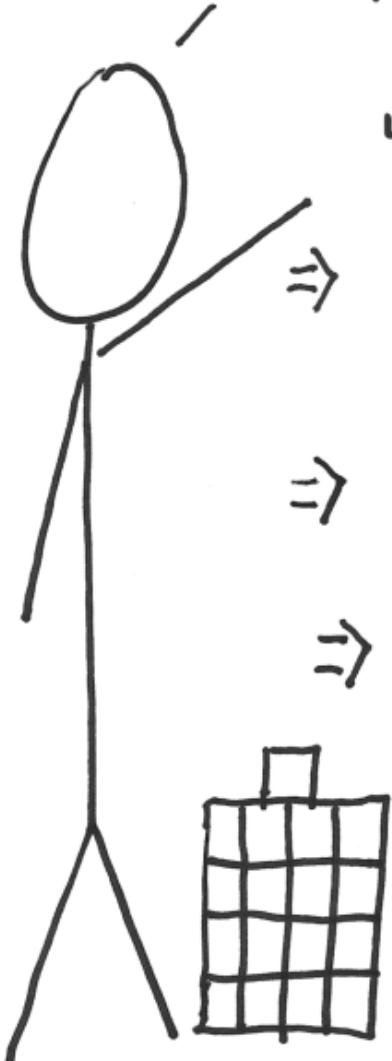
With the "new" addition, things are simpler, but the  $x^{13}$  is still too big. Let's make it so we can't go bigger than  $x^7$ . How can we do that?

$$\begin{aligned} & x^{13} \oplus 2x^{11} \oplus 3x^9 \oplus 4x^7 \oplus 3x^5 \oplus 2x^3 \oplus x \\ \Rightarrow & x^{13} \oplus 0x^{11} \oplus x^9 \oplus 0x^7 \oplus x^5 \oplus 0x^3 \oplus x \\ - & x^{13} \oplus x^9 \oplus x^5 \oplus x \end{aligned}$$



**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:



$$4 \text{ o'clock} + 10 \text{ hours} = 2 \text{ o'clock}$$

$$\Rightarrow \begin{array}{c} \text{Clock at } 4 \\ + 10 \text{ hours} \end{array} = \begin{array}{c} \text{Clock at } 2 \end{array}$$

$$\Rightarrow 4$$

$$\Rightarrow$$

$$\begin{array}{r} 14 \\ - 12 \\ \hline 2 \end{array}$$

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  $m(x) = x^8 + x^4 + x^3 + x + 1$ . Let's say we wanted to multiply  $x \cdot b(x)$  where  $b(x)$  has coefficients  $b_7 \dots b_0$ :

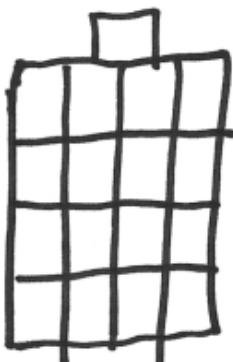
$$x \cdot b(x)$$

$$= x \cdot (b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0)$$

$$= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$



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

$$x^8 + x^4 + x^3 + x + 1 \\ m(x)$$

$$\begin{array}{r} b_7 \\ \hline b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ \oplus \quad b_7x^8 \\ \hline \end{array}$$

$$\begin{array}{l} b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + b_7)x^4 + (b_2 + b_7)x^3 \\ \qquad \qquad \qquad \oplus b_1x^2 + (b_0 + b_7)x + b_7 \\ \hline \end{array}$$

Remainder

$$\begin{array}{l} \rightarrow b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ \qquad \qquad \qquad \qquad \qquad \qquad \qquad \oplus b_7 \cdot (x^4 + x^3 + x + 1) \\ \qquad \qquad \qquad \nearrow \qquad \qquad \qquad \nearrow \\ \end{array}$$

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

This is just  $b_7$  multiplied by a small polynomial.

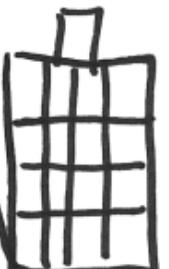
Now we're ready for the hardest blast from the past: logarithms. After logarithms, everything else is cake! Logarithms let us turn multiplication into addition:


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

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

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  $x \oplus 1$  and watch the magic unravel.\*



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

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

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

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 generator 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<sup>th</sup> degree polynomial can be represented in exactly 1 byte since the new way uses only 0 or 1 for coefficients:


$$\begin{aligned} & x^4 \oplus x^3 \oplus x \oplus 1 \\ = & 0x^7 \oplus 0x^6 \oplus 0x^5 \oplus 1x^4 \oplus 1x^3 \oplus 0x^2 \oplus 1x \oplus 1 \\ = & \underbrace{0 \quad 0 \quad 0}_{\downarrow} \quad \underbrace{1}_{\downarrow} \quad \underbrace{1 \quad 0 \quad 1 \quad 1}_{\downarrow \downarrow \downarrow \downarrow} \end{aligned}$$

$1011_2 = 11_{10} = b_{16} \leftarrow \text{hexadecimal}$

$= \boxed{b} \nwarrow \text{A single byte!!}$

\*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.\*

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

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

\*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.



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

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

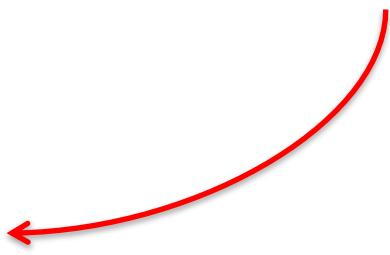
$$\text{sbox}[a] = f(g(a))$$

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

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

$$58 \cdot 18 = 01$$

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



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	

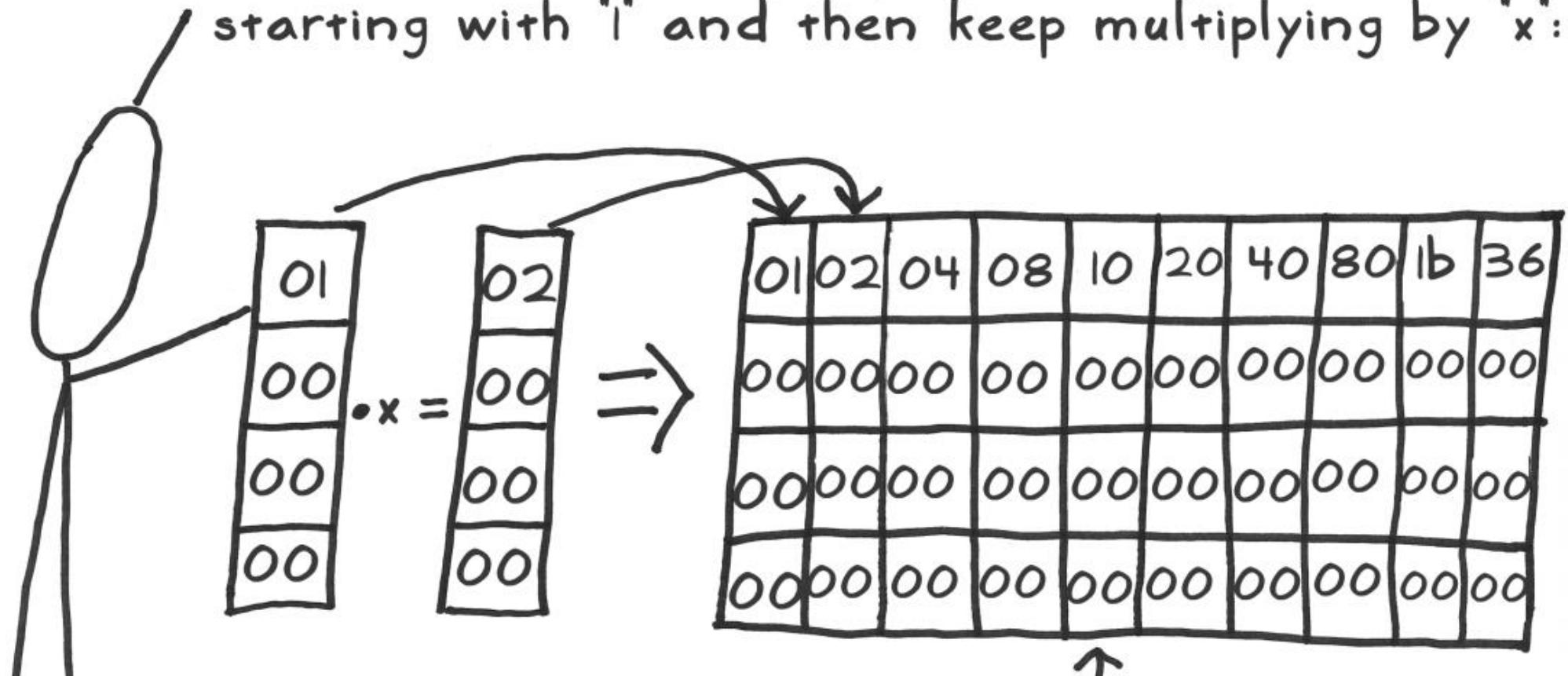
Recall:

$$\begin{aligned}sbox[58] &= f(g(58)) \\ sbox[58] &= f(18) = 6a\end{aligned}$$

$$58 \cdot 18 = 01$$

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?

First 10 round constants

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

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

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

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

...

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

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

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

↑ special polynomial    ↑ the column

$$= x^4 + 1 \frac{03a_3 \cdot x^2 + (3a_2 + a_3)x + (3a_1 + a_2 + a_3)}{03a_3x^6 + 03a_2x^5 + 03a_1x^4 + 03a_0x^3 + 01a_3x^5 + 01a_2x^4 + 01a_1x^3 + 01a_0x^2 + 01a_3x^4 + 01a_2x^3 + 01a_1x^2 + 01a_0x + 02a_3x^3 + 02a_2x^2 + 02a_1x + 02a_0}$$

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

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

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

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



General Math

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

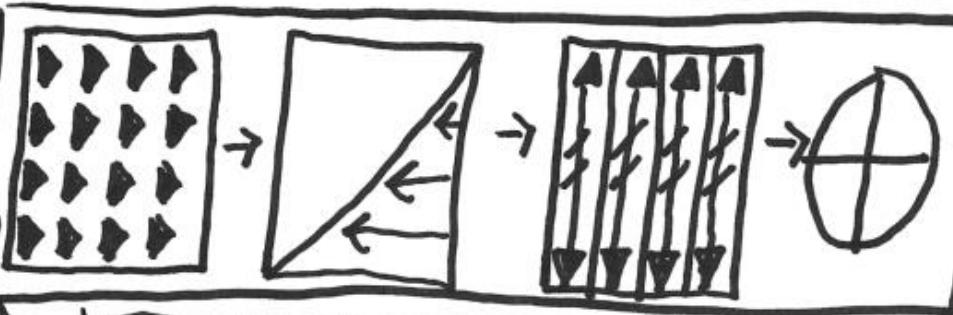
Fast Multiply

$$X^8 + X^4 + X^3 + X + 1$$

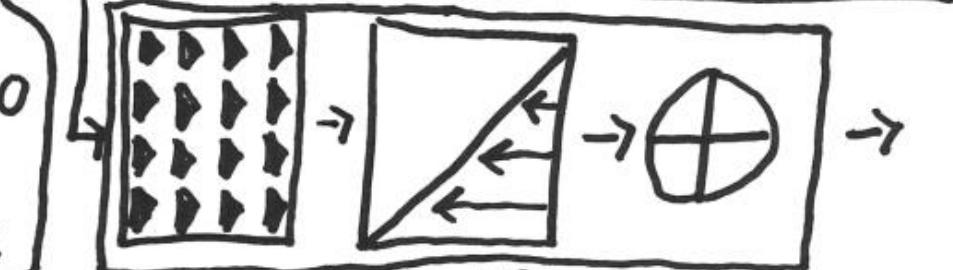
$$X \cdot a(x) = (a \ll 1) \oplus (a_7 = 1) ? 1B : 00$$

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

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



Intermediate Rounds	
#	Key
X	128
11	192
13	256



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

Ciphertext

S-Box (SRD)

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

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

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

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

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

Final Round

Key Expansion: Round Constants

S	K	B3	D1	B2
0	E	6E	00	CE
1	T	CB	00	CB
2	Y	B7	00	B7
3	P			

Round Key 0

Other Columns:

T	E1	C1
Z	Z1	10
8	86	B4

Prev Col  $\oplus$  Col from Previous round key

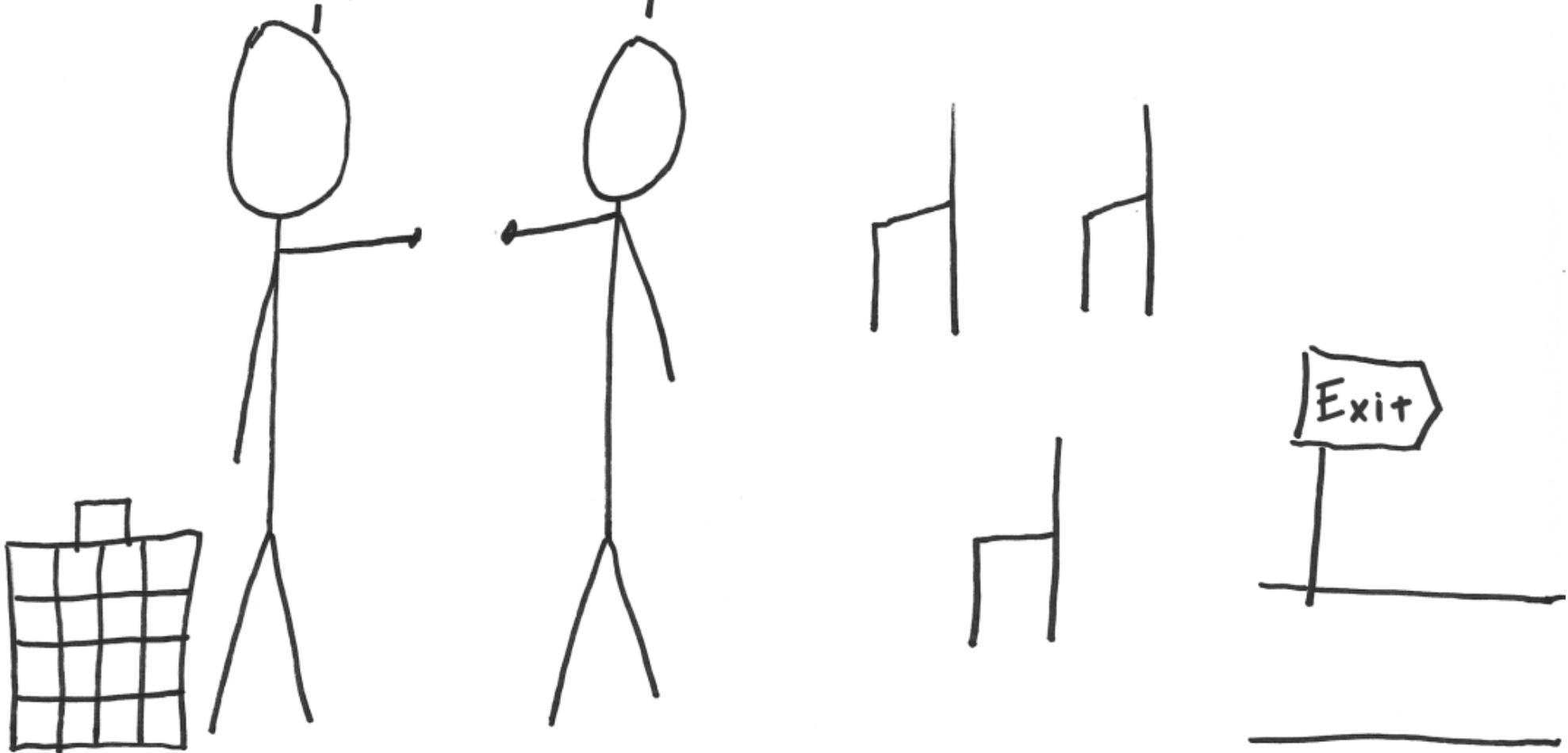
Mix Columns:

2	1	1	3	2
2	1	3	1	1
3	2	1	1	1
1	3	2	1	1
1	1	1	3	2

Inverse Mix

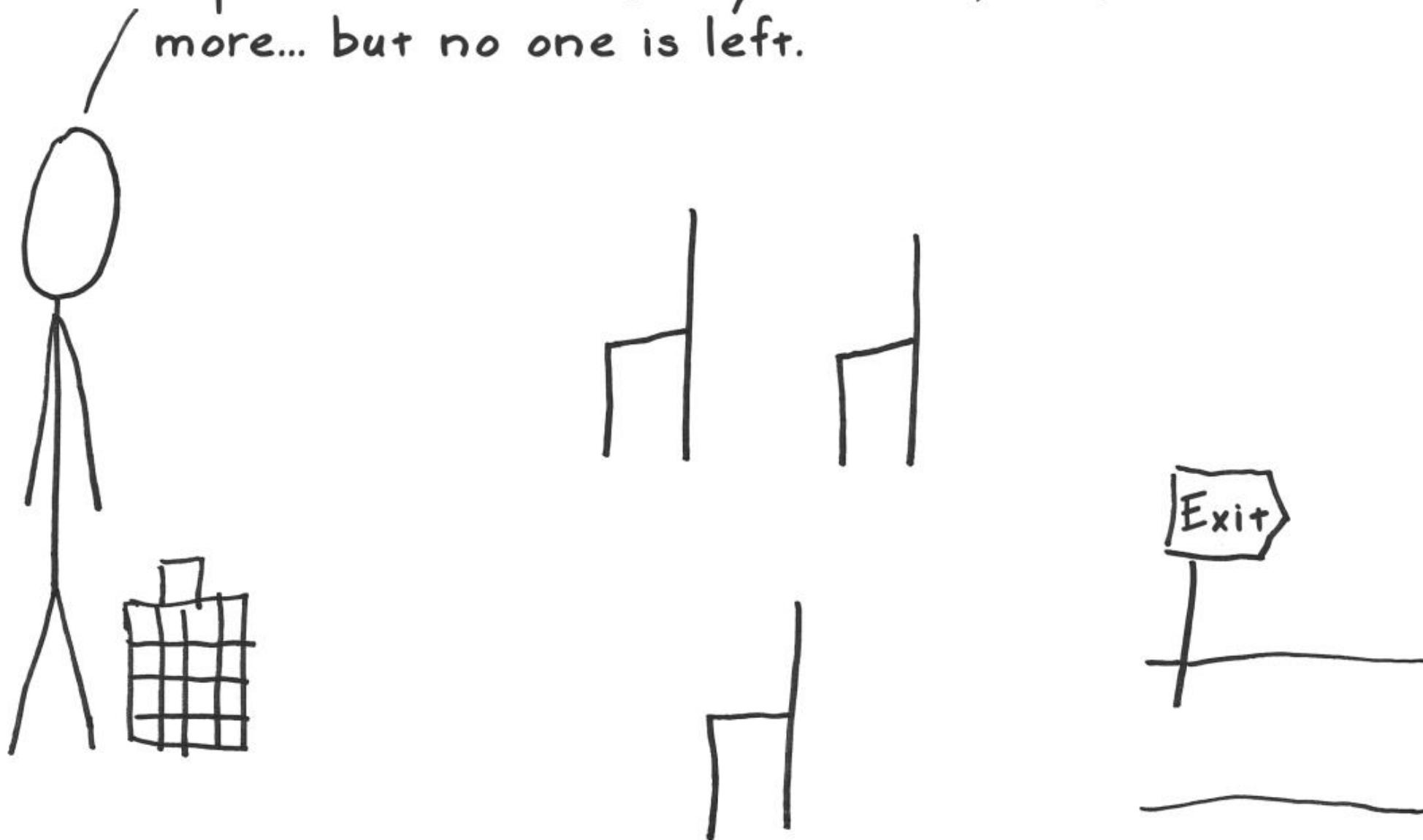
E	B	D	9	a3
9	E	B	D	a2
D	9	E	D	a1
B	D	9	B	a0

My pleasure.  
Come back anytime!

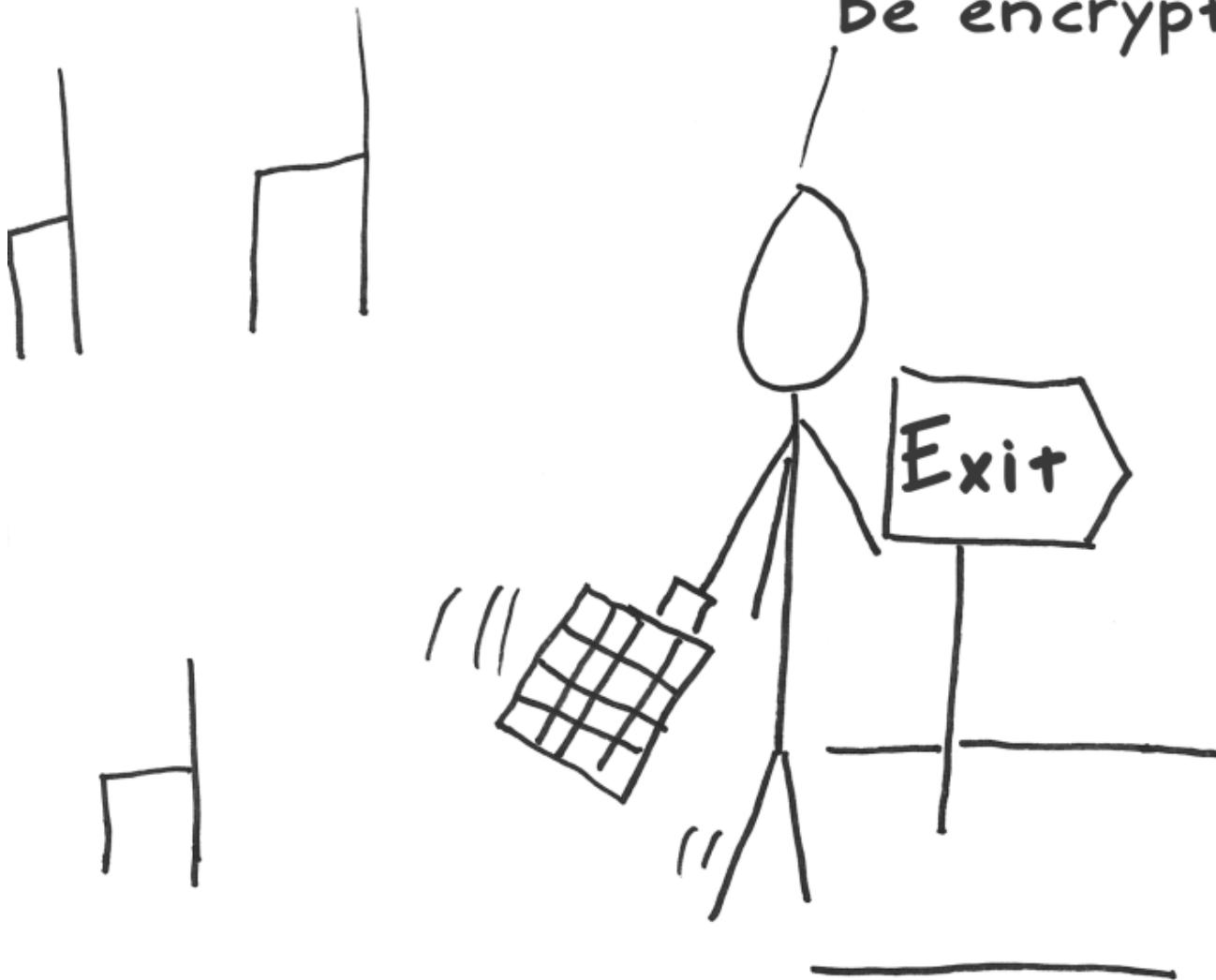


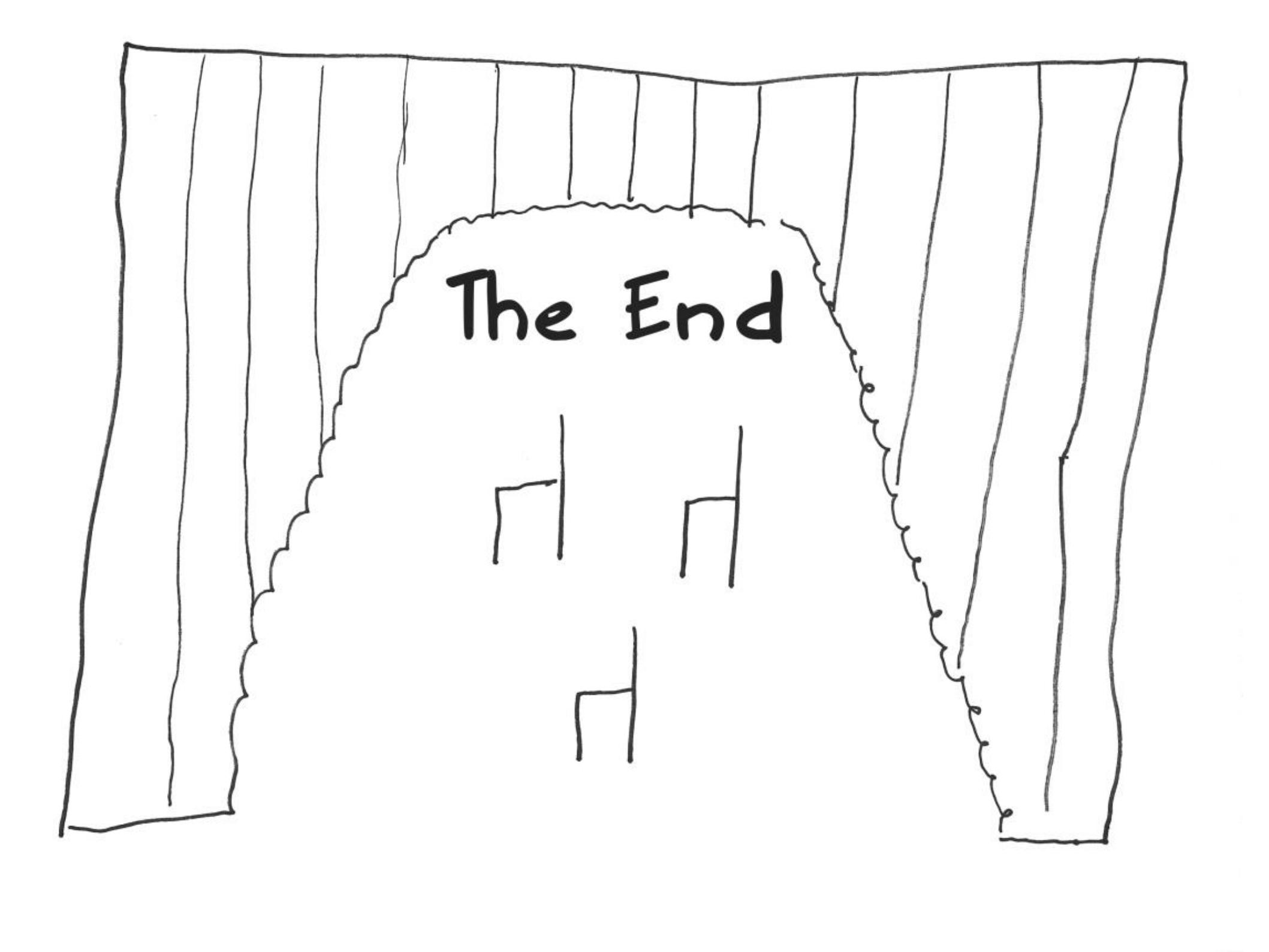
Whoa... I think I get it now. It's relatively simple once you grok the pieces. Thanks for explaining it. I gotta go now.

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.

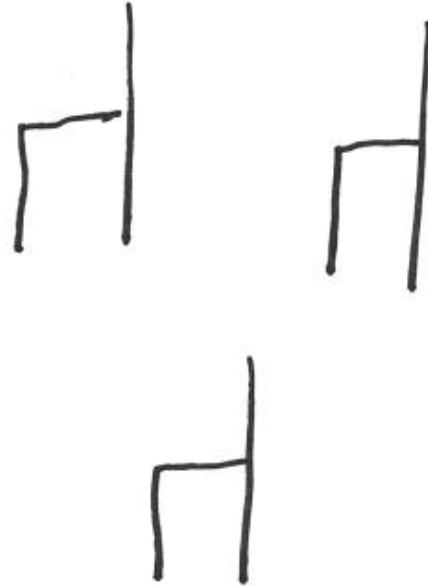


Oh well... there's some boring  
router traffic that needs to  
be encrypted. Gotta go!





The End



# Final Thoughts...

In theory, you now understand how 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

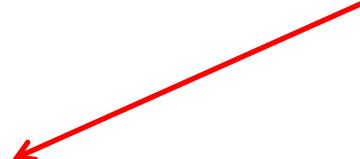
**Good non-linearity properties**

**Low level:**

- 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 😊

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

Data computed from the key, used in encryption

### P-array

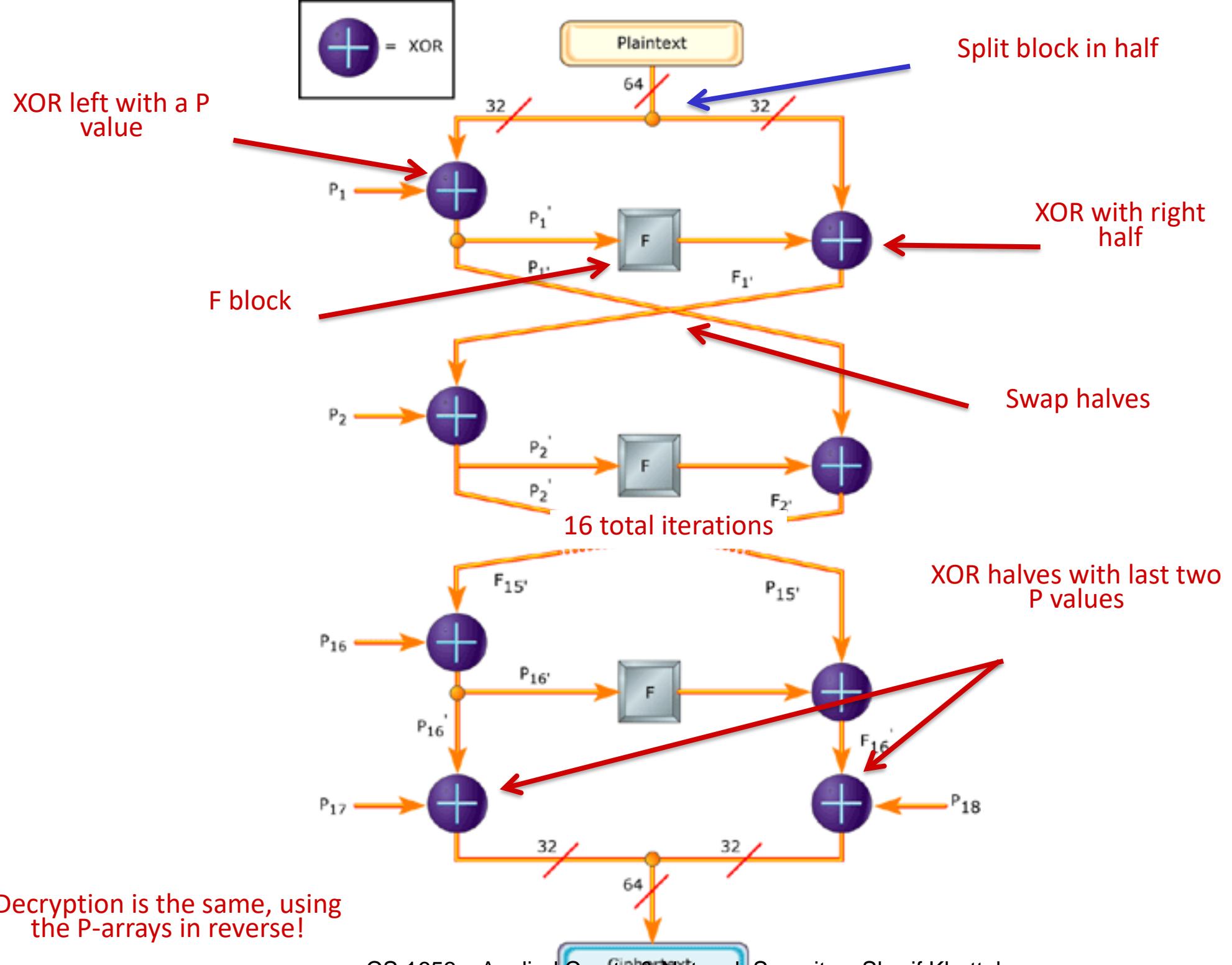
- 18 subkeys
- 32 bits each
- $P_1, P_2, \dots, P_{18}$

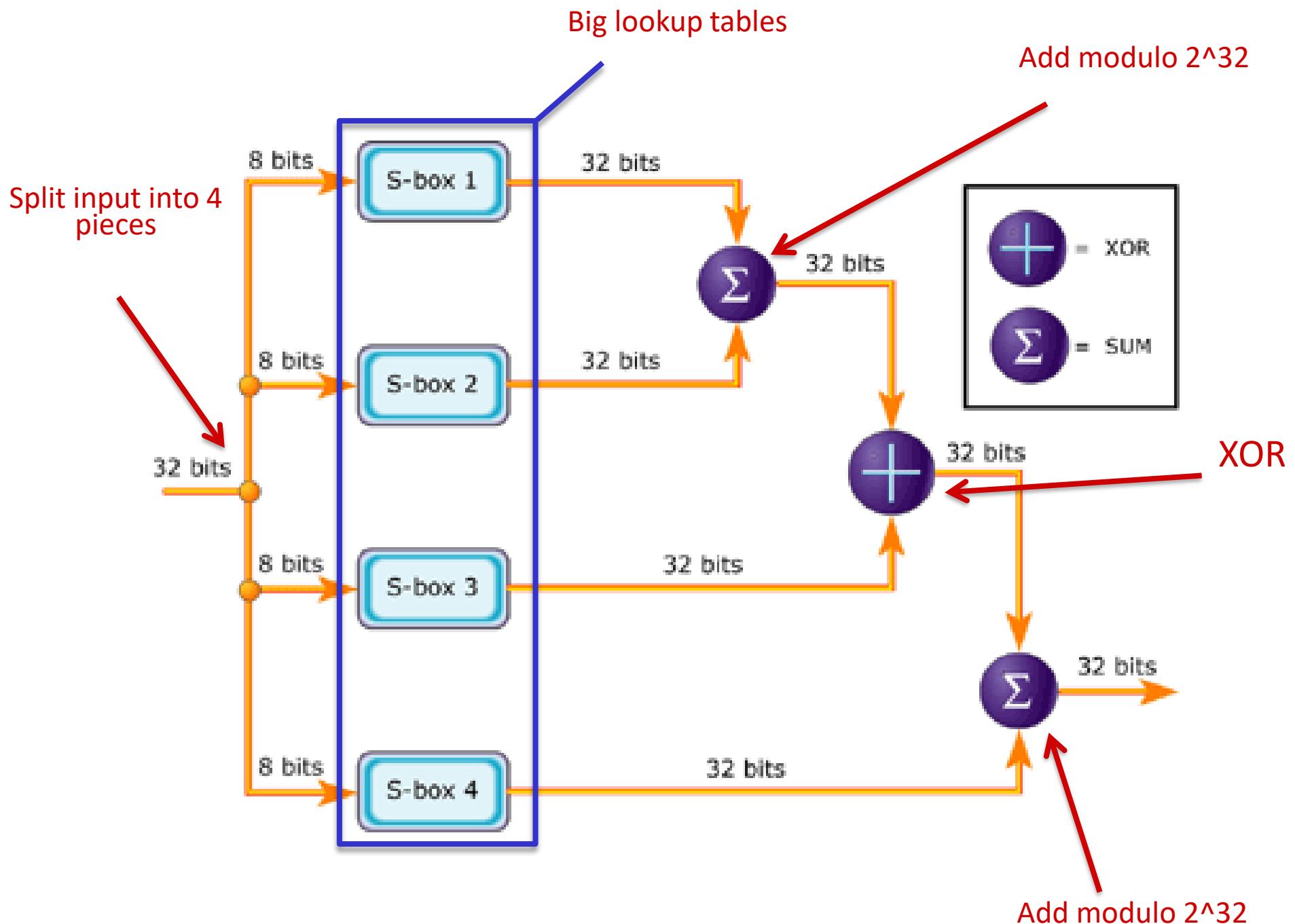
72 bytes

### S-boxes

- 4 lookup tables
- 8 bits  $\rightarrow$  32 bits
- $S_{1,0}, S_{1,1}, \dots, S_{1,255};$   
 $S_{2,0}, S_{2,1}, \dots, S_{2,255};$   
 $S_{3,0}, S_{3,1}, \dots, S_{3,255};$   
 $S_{4,0}, S_{4,1}, \dots, S_{4,255};$

4096 bytes





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

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

1. Fill P-array and S-boxes with the (hex)digits of pi
2. XOR the key into the P-array

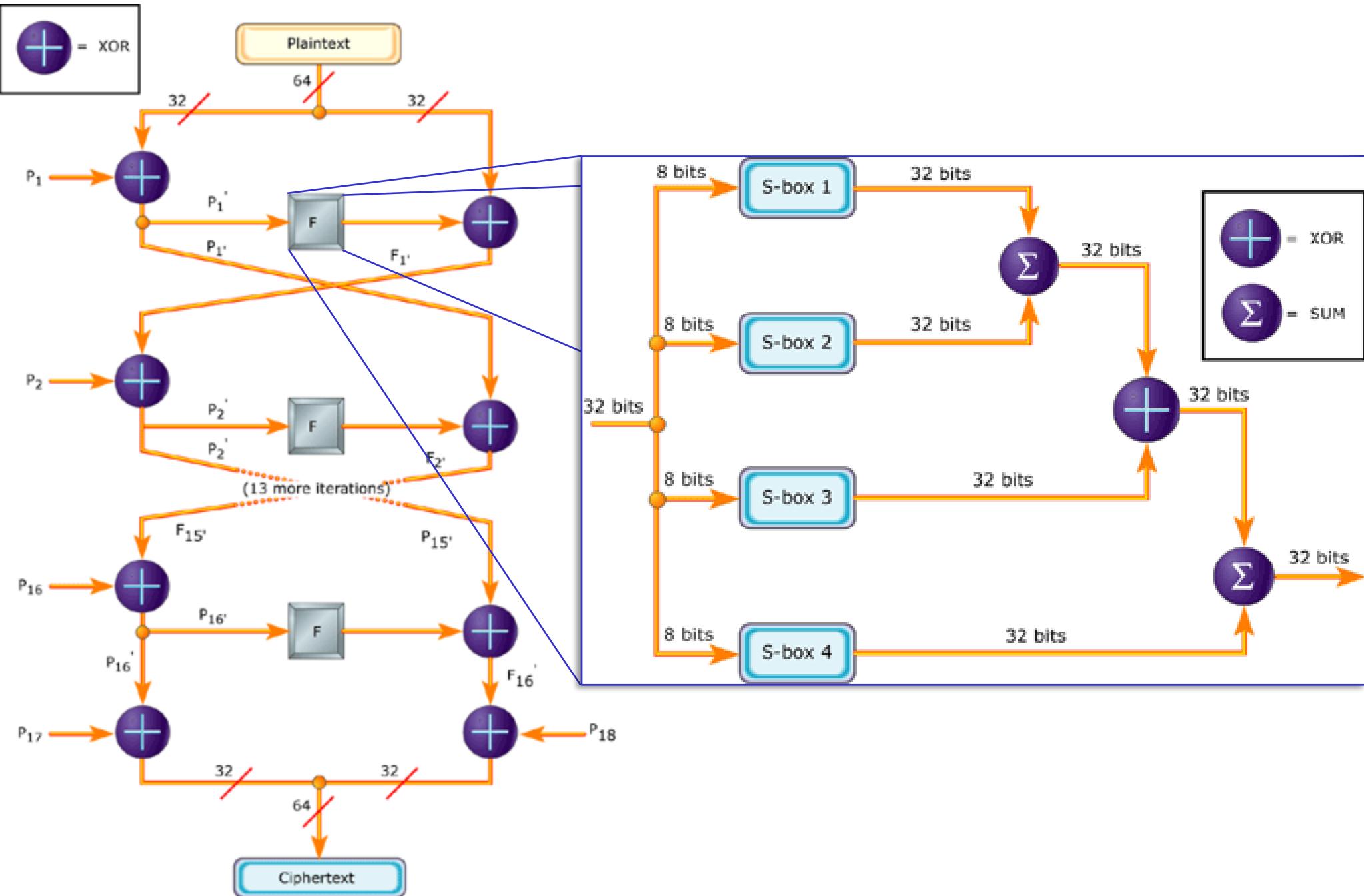
$$P_1 = P_1 \oplus \text{first 32 bits of key}$$

$$P_2 = P_2 \oplus \text{second 32 bits of key}$$

... repeat key as needed

3. Encrypt 0 string, replace  $P_1, P_2$  with output
4. Encrypt output, replace  $P_3, P_4$  with new output
5. Repeat until entire P-array and all S-boxes are replaced

521 encryptions!!



# Discussion question!

Why initialize the  
constants with the  
digits of pi?

Nothing up my sleeve!

# Discussion question!

Which step in Blowfish  
is very inefficient?

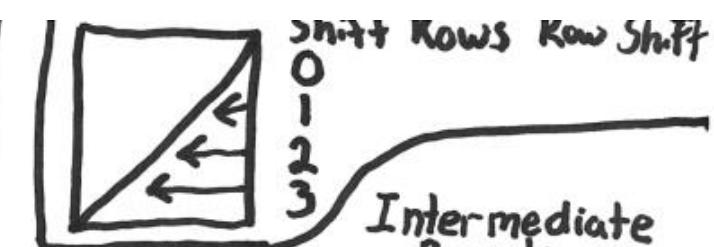
Deriving subkeys, changing keys

In what way is that a  
good thing?

Makes brute forcing hard!



Initial Round



General Math

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

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

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

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

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

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

$[11111000] [a_7 \ 0]$

$[01111100] [a_6 \ 1]$

$[00111110] [a_5 \ 0]$

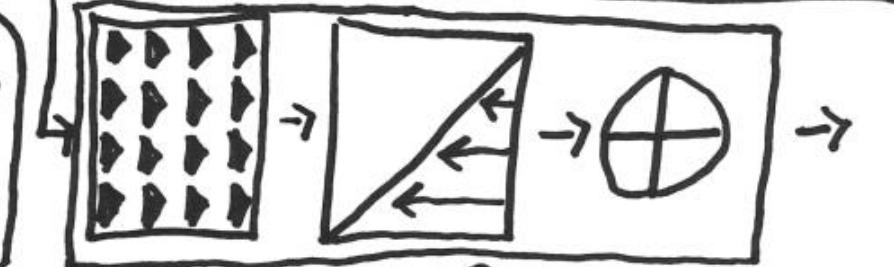
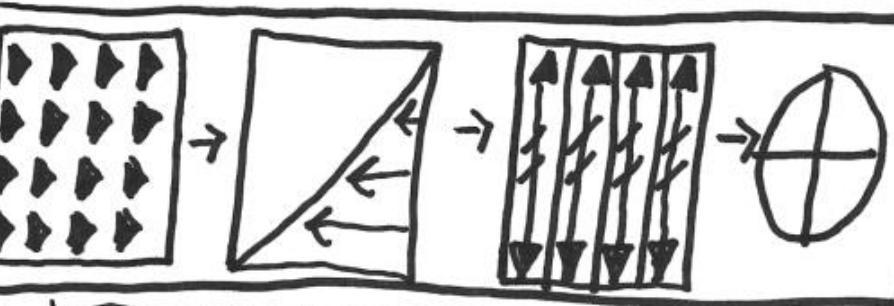
$[00011111] [a_4 \ 0]$

$[10001111] [a_3 \ 0]$

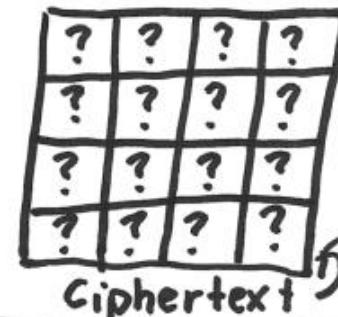
$[10001111] [a_2 \ 1]$

$[11000111] [a_1 \ 1]$

$[11100011] [a_0 \ 1]$



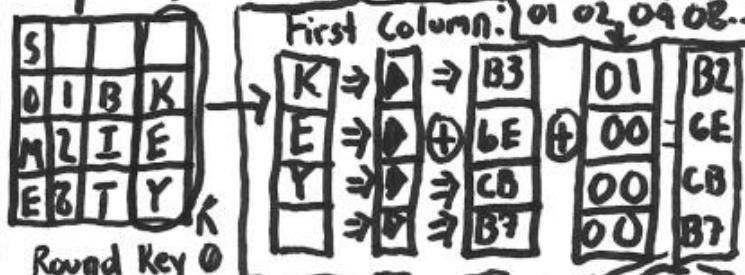
#	Key
9	128
11	192
13	256



Ciphertext

Final Round

Key Expansion: Round Constants



Mix Columns:

2113 2

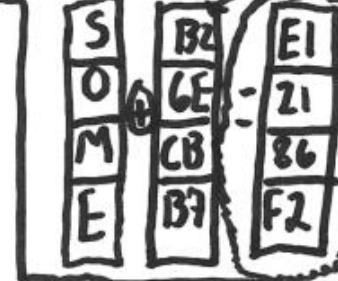
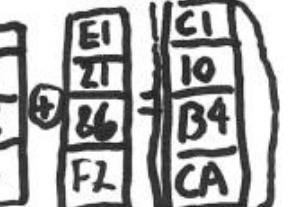
2113 [a<sub>3</sub>]

3211 [a<sub>2</sub>]

1321 [a<sub>1</sub>]

1132 [a<sub>0</sub>]

Other Columns:



Inverse Mix

EBD9

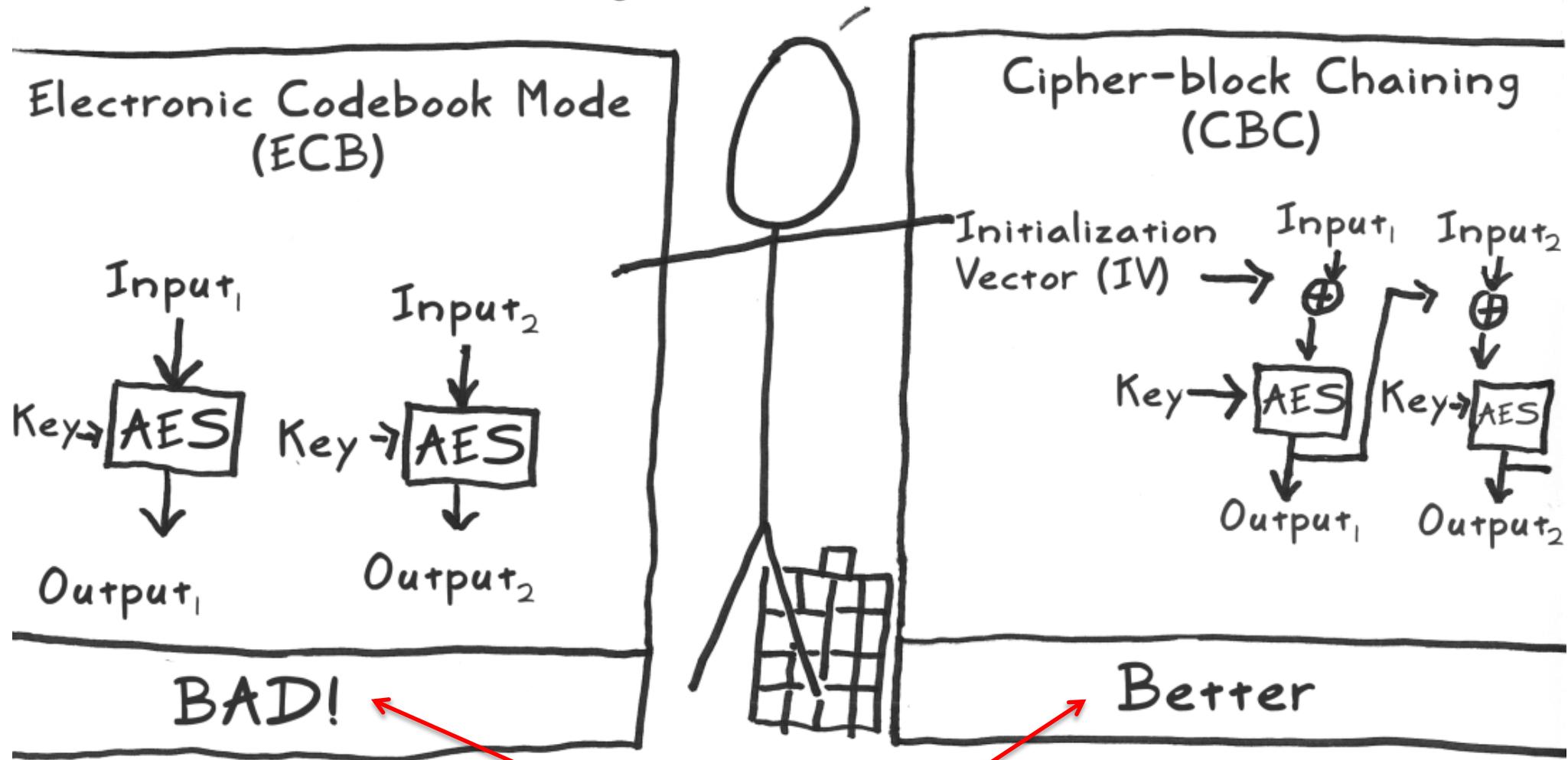
E B D9 [a<sub>3</sub>]

9 E B D [a<sub>2</sub>]

D 9 E D [a<sub>1</sub>]

B D 9 B [a<sub>0</sub>]

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?

# Final Thoughts

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

What about more complex ciphers like AES?

Additions, multiplications, and inversion of polynomials in  $\text{GF}(2^8)$   
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