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

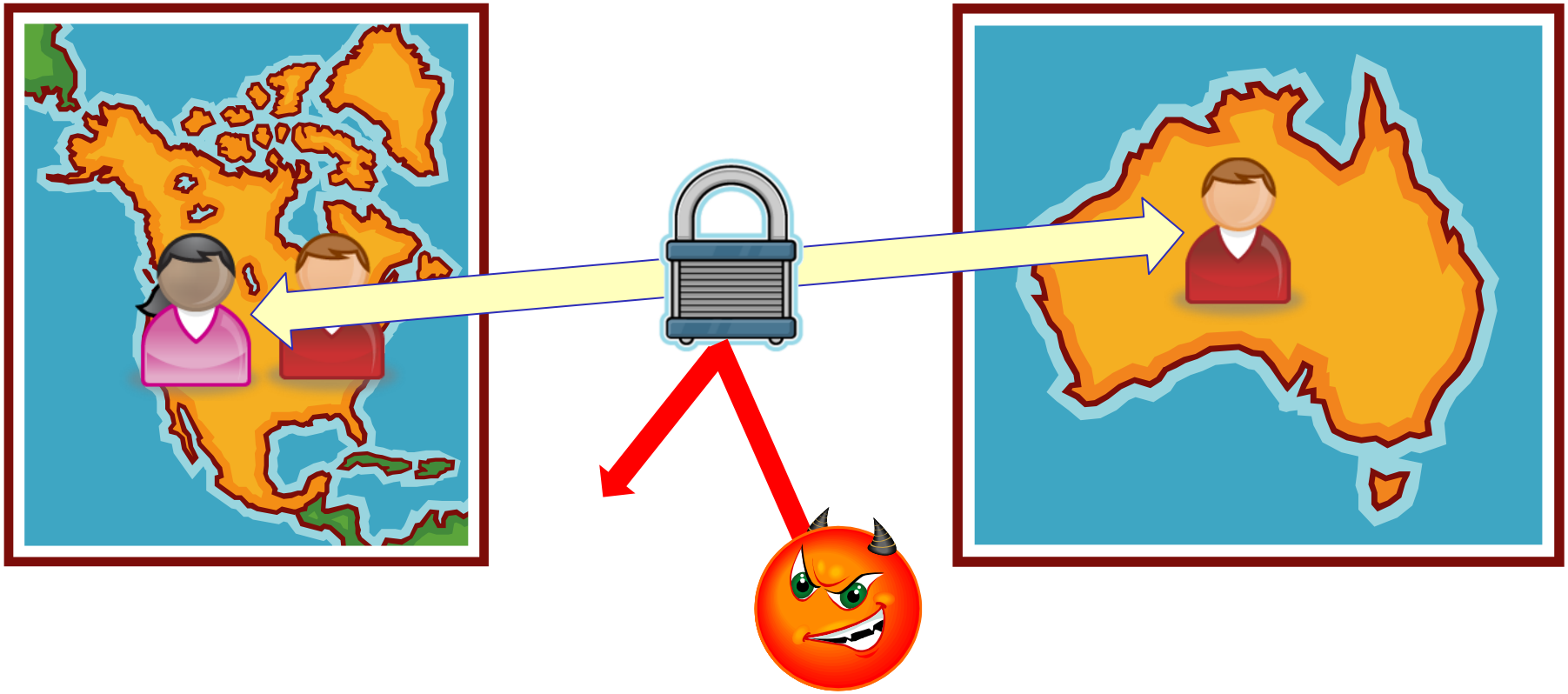
- Please skim over assigned reading
- Homework 1 due this Friday @ 11:59 pm
  - 3 attempts
- add 15 minutes to each lecture
  - updated lecture time: **1:30 pm – 3:30 pm**
  - **feel free to leave at 3:15 if you have to**
- Makeup lectures
  - Friday 6/16 11:00-12:45
  - Friday 6/23 15:00-16:15
- **lectures 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

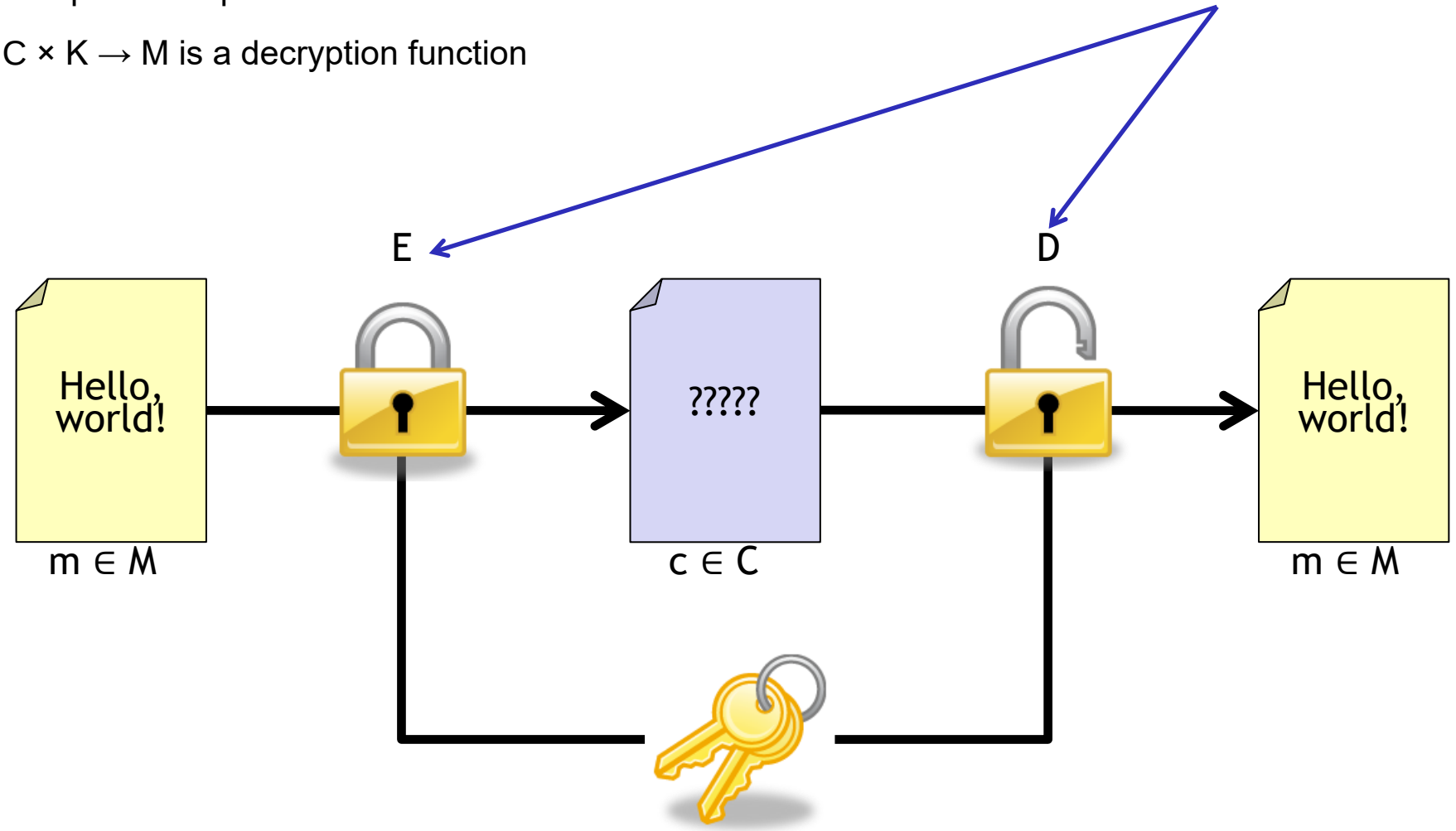
K is a key space

E : M × K → C is an encryption function

C is a ciphertext space

D : C × K → M is a decryption function

Our focus now is on symmetric key encryption



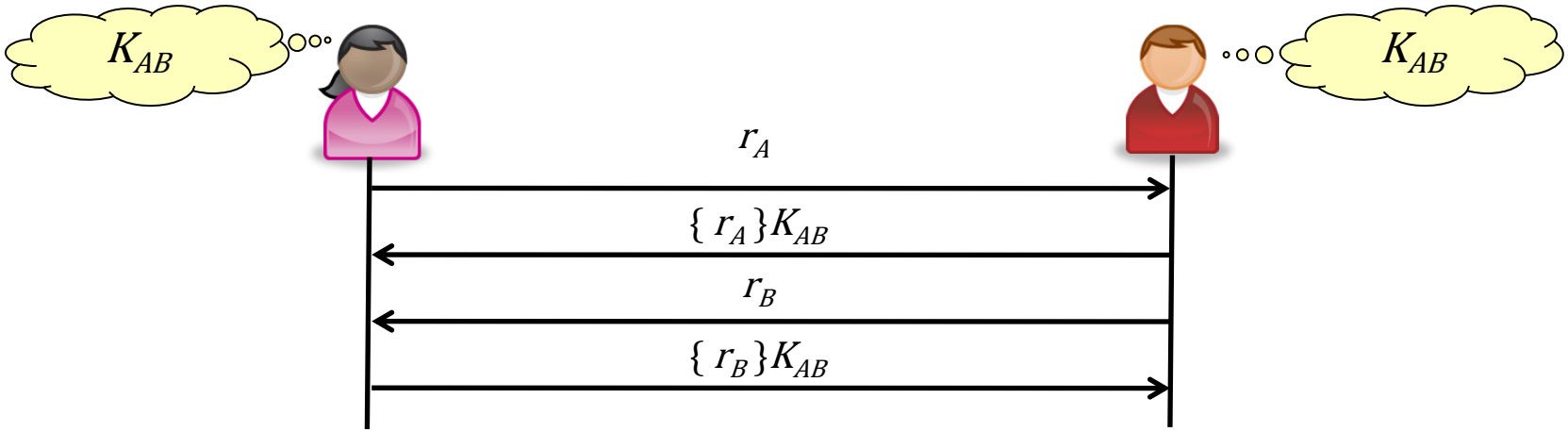
Why study symmetric key cryptography?

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, $\mathrm{H}(m) = \mathrm{H}(m \mid c)$

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.

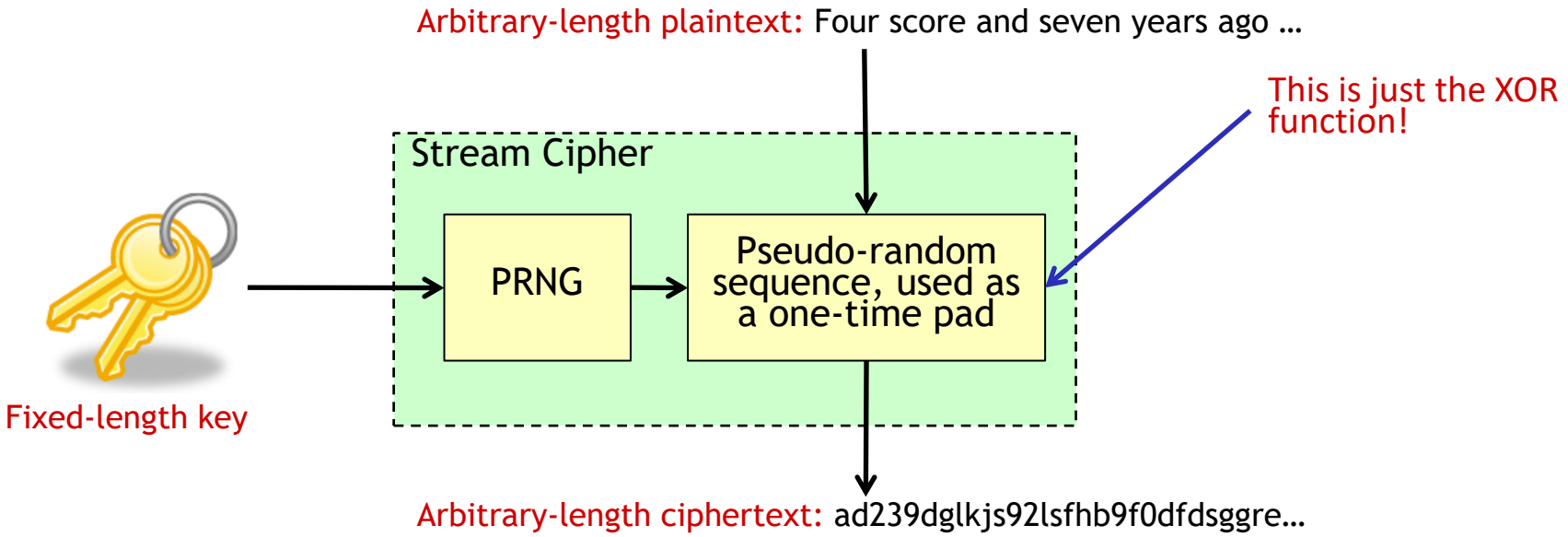
Stream ciphers work like a one-time pad

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**

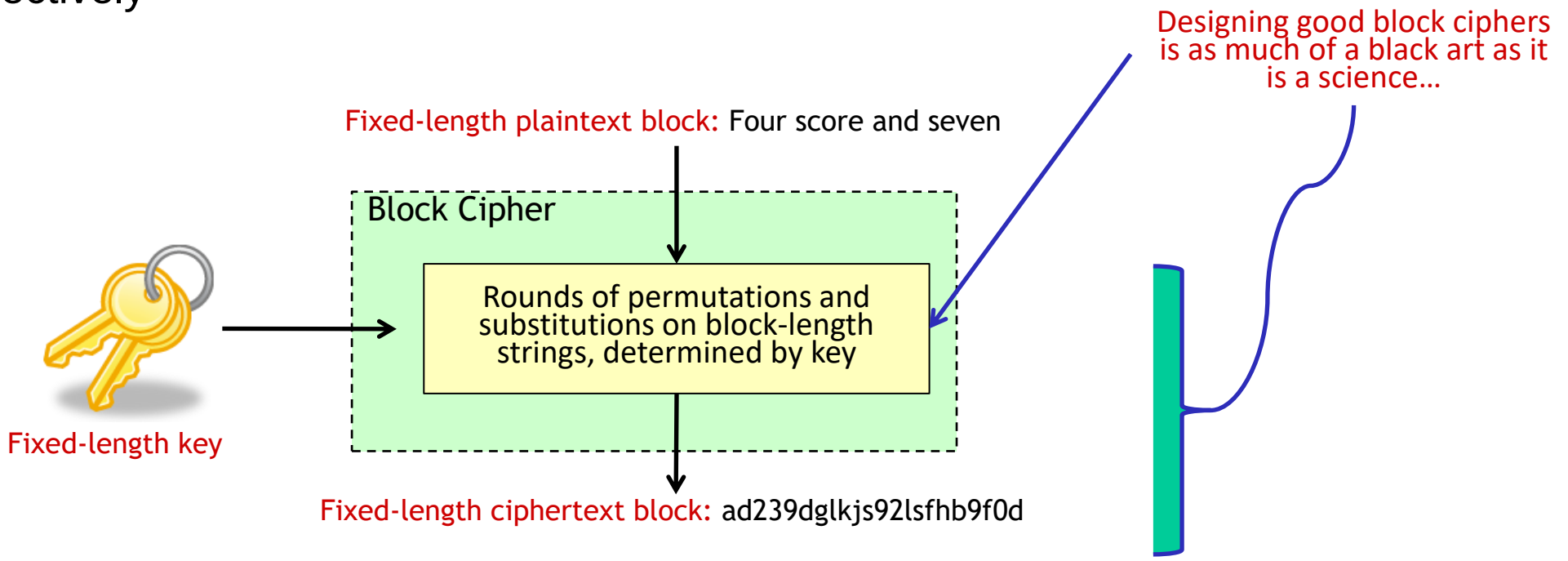
Goal: hide/obscure patterns in plaintext and key

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

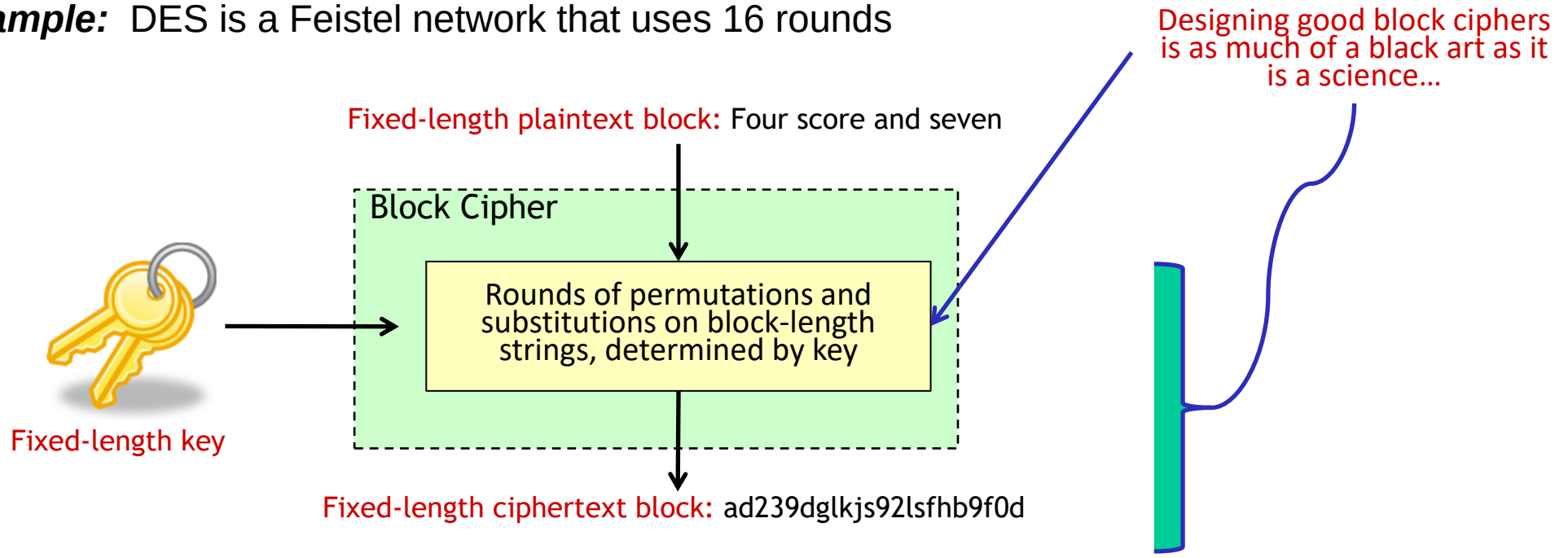
Often, block ciphers apply several rounds of a simpler function

Most block ciphers can be categorized as Feistel networks

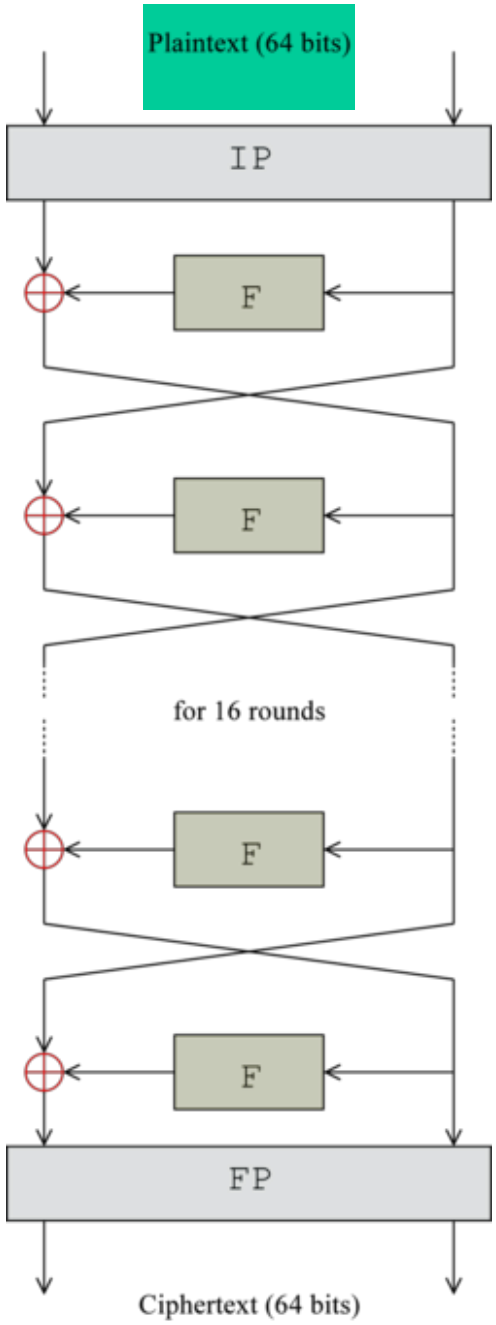
Bit shuffling, non-linear substitution, and linear mixing (XOR)

Confusion and diffusion *a la* Claude Shannon

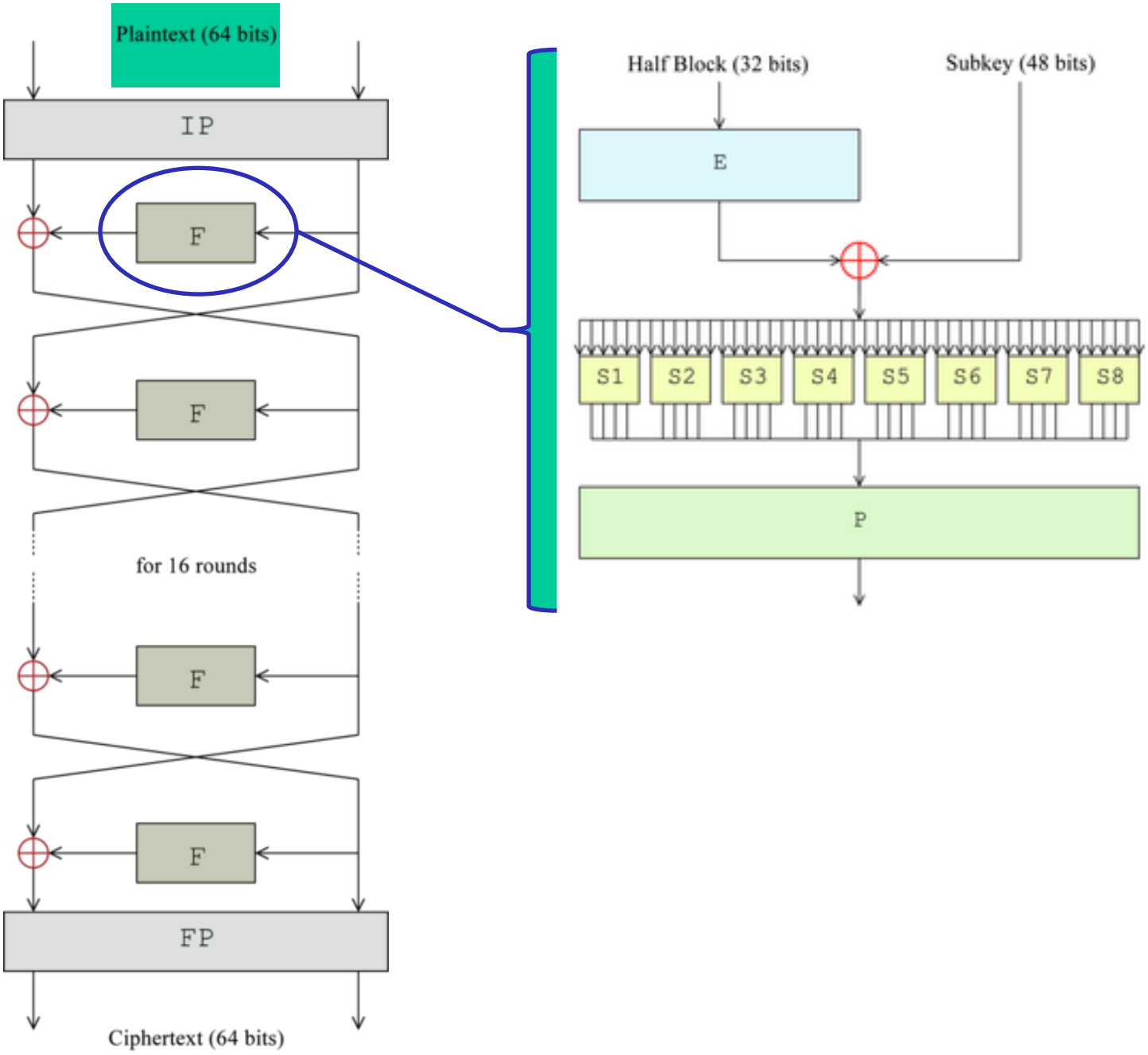
Example: DES is a Feistel network that uses 16 rounds



Example: Data Encryption Standard (DES)

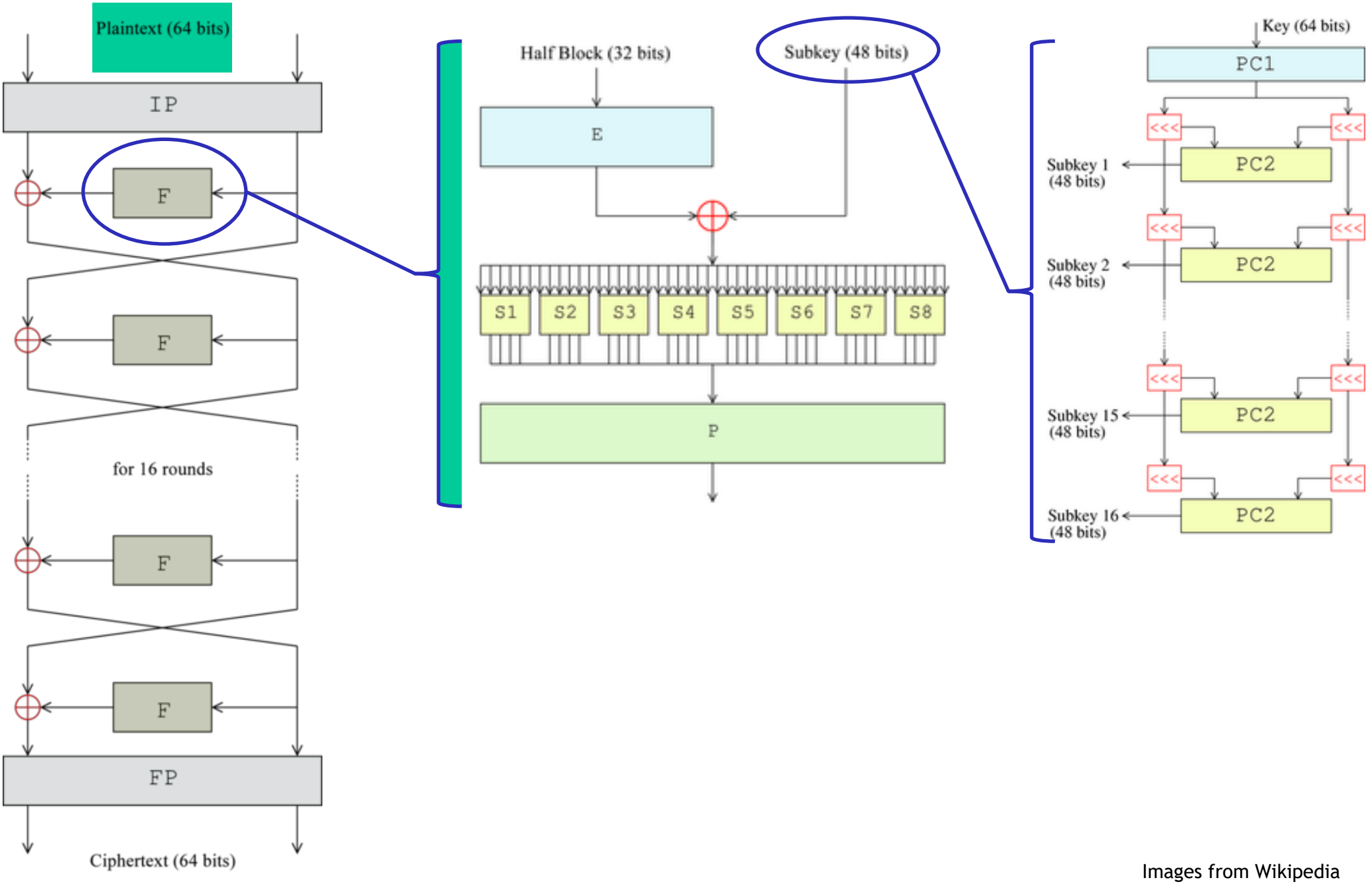


Example: DES



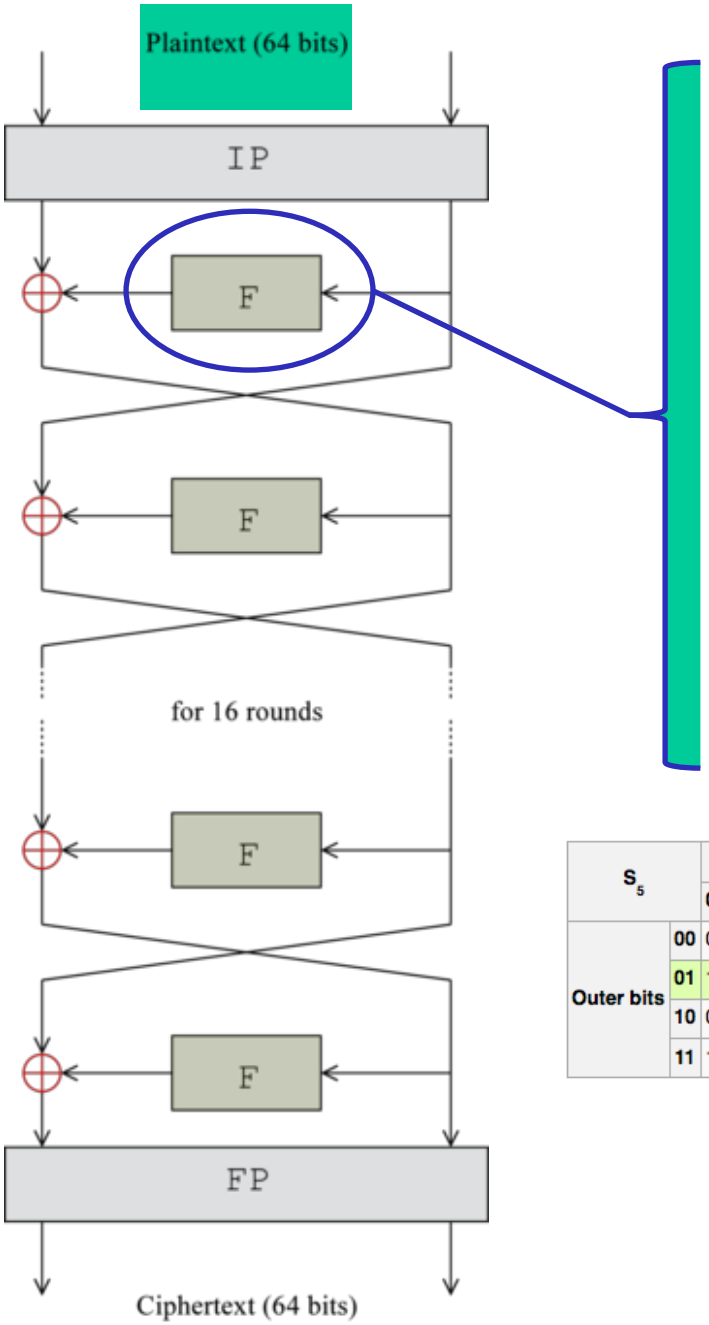
Images from Wikipedia

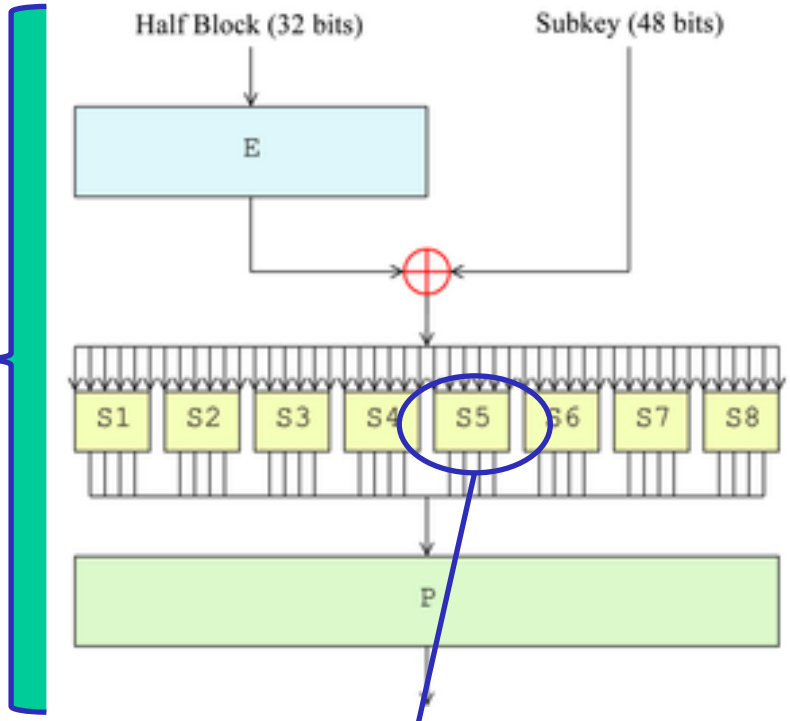
Example: DES



Images from Wikipedia

Example: DES

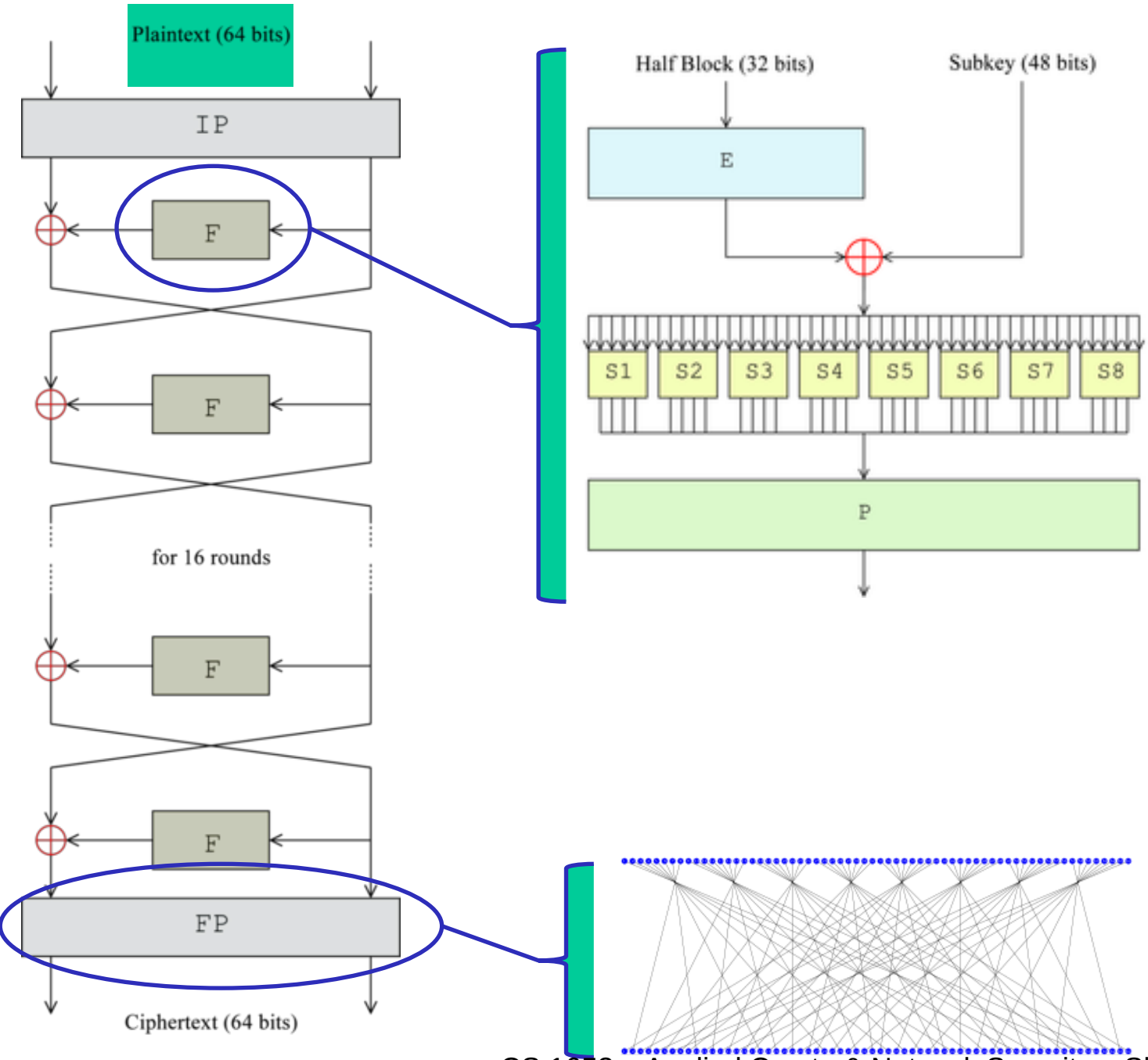




S_5		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Images from Wikipedia

Example: DES



Images from Wikipedia

Example: Advanced Encryption Standard (AES)

Develop a (brief) history of modern cryptography

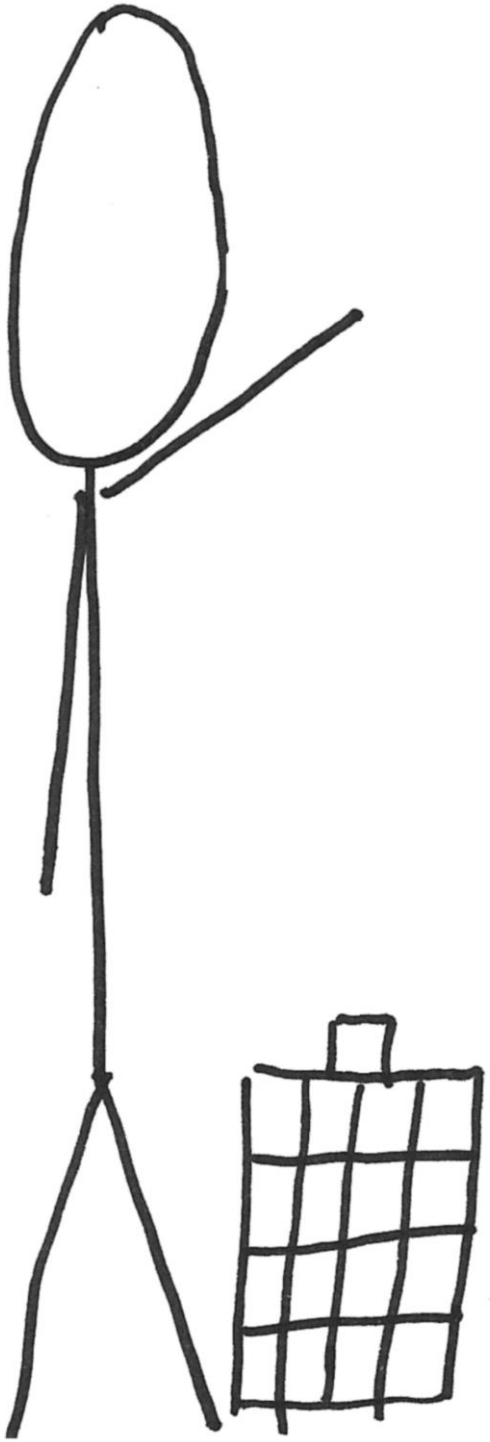
Learn about the AES standardization process

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

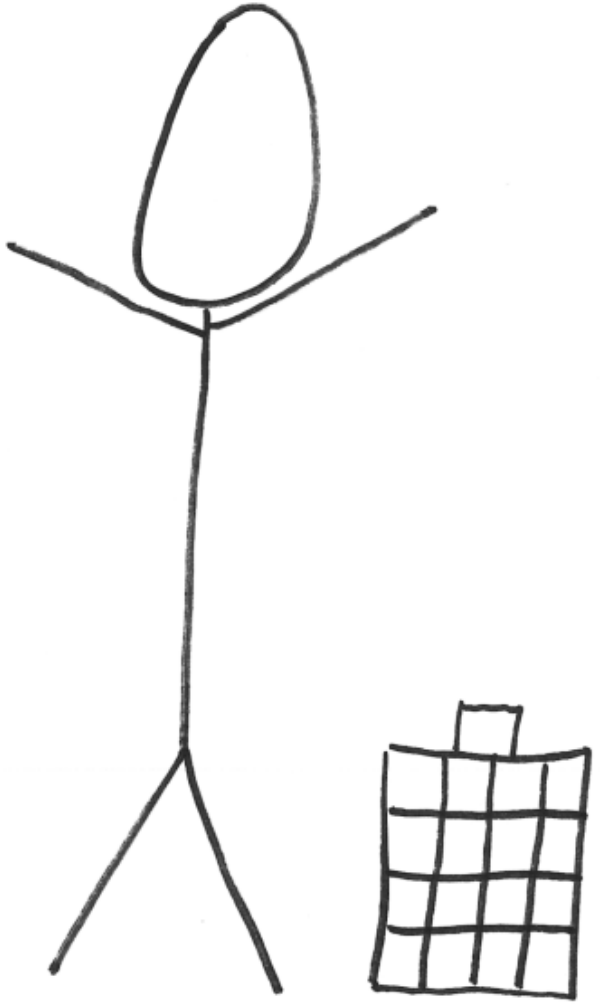
A Stick Figure Guide to the Advanced Encryption Standard (AES)



http://www.moserware.com/

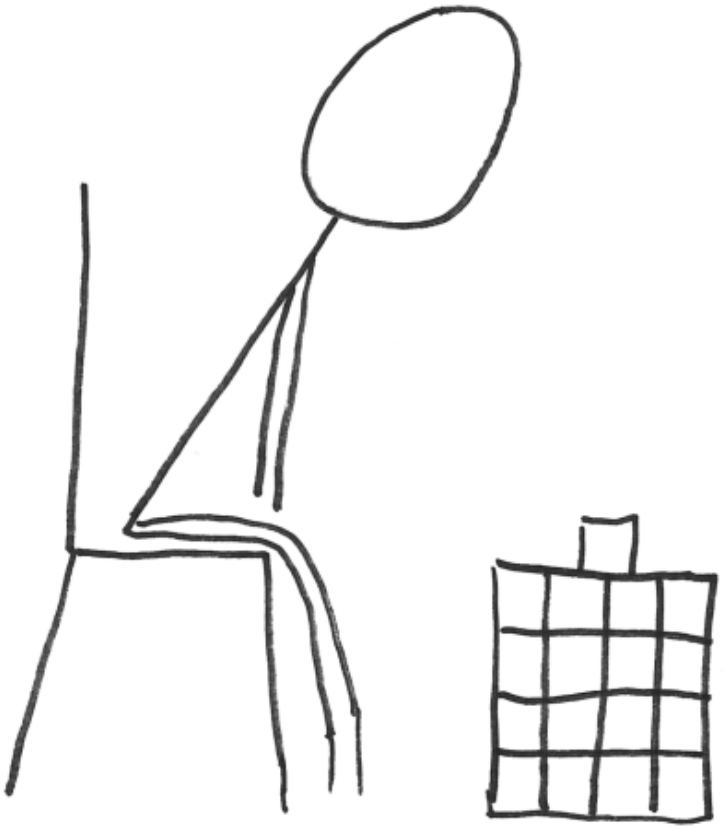
Act 1: Once Upon a Time…

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!

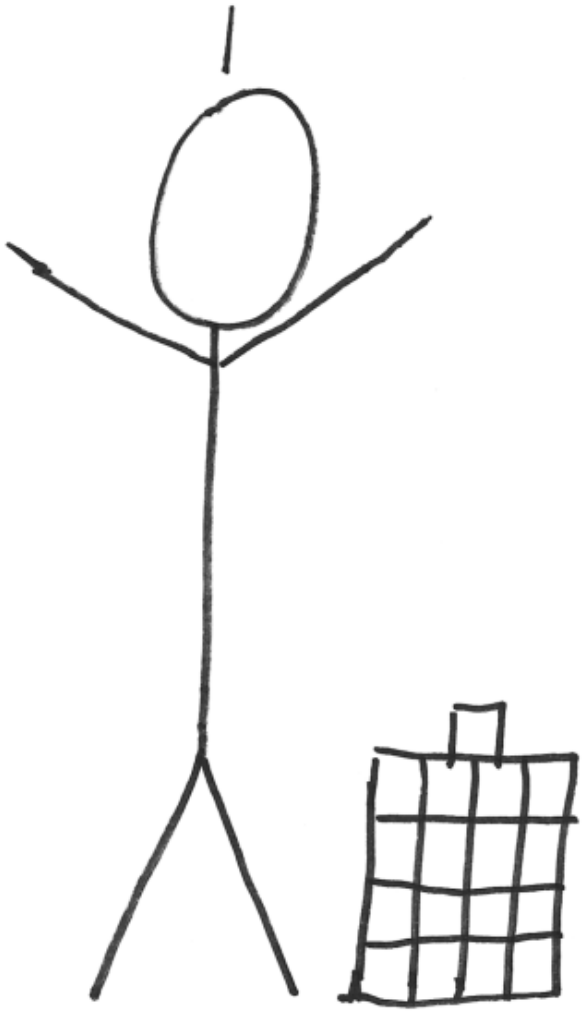


* 1 petabyte ≈ a lot

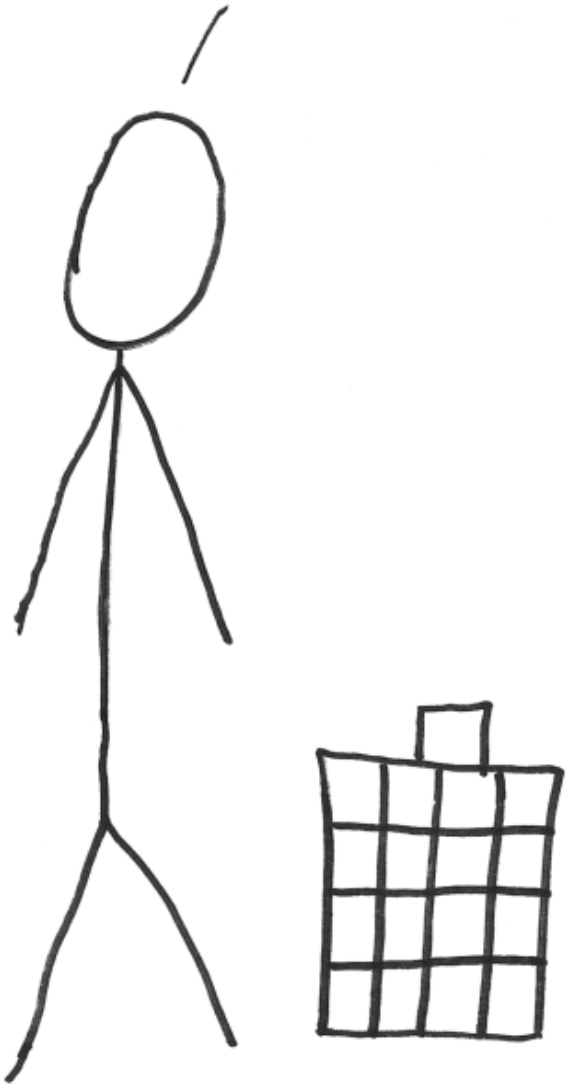
... and still no one seems to care about me or my story.

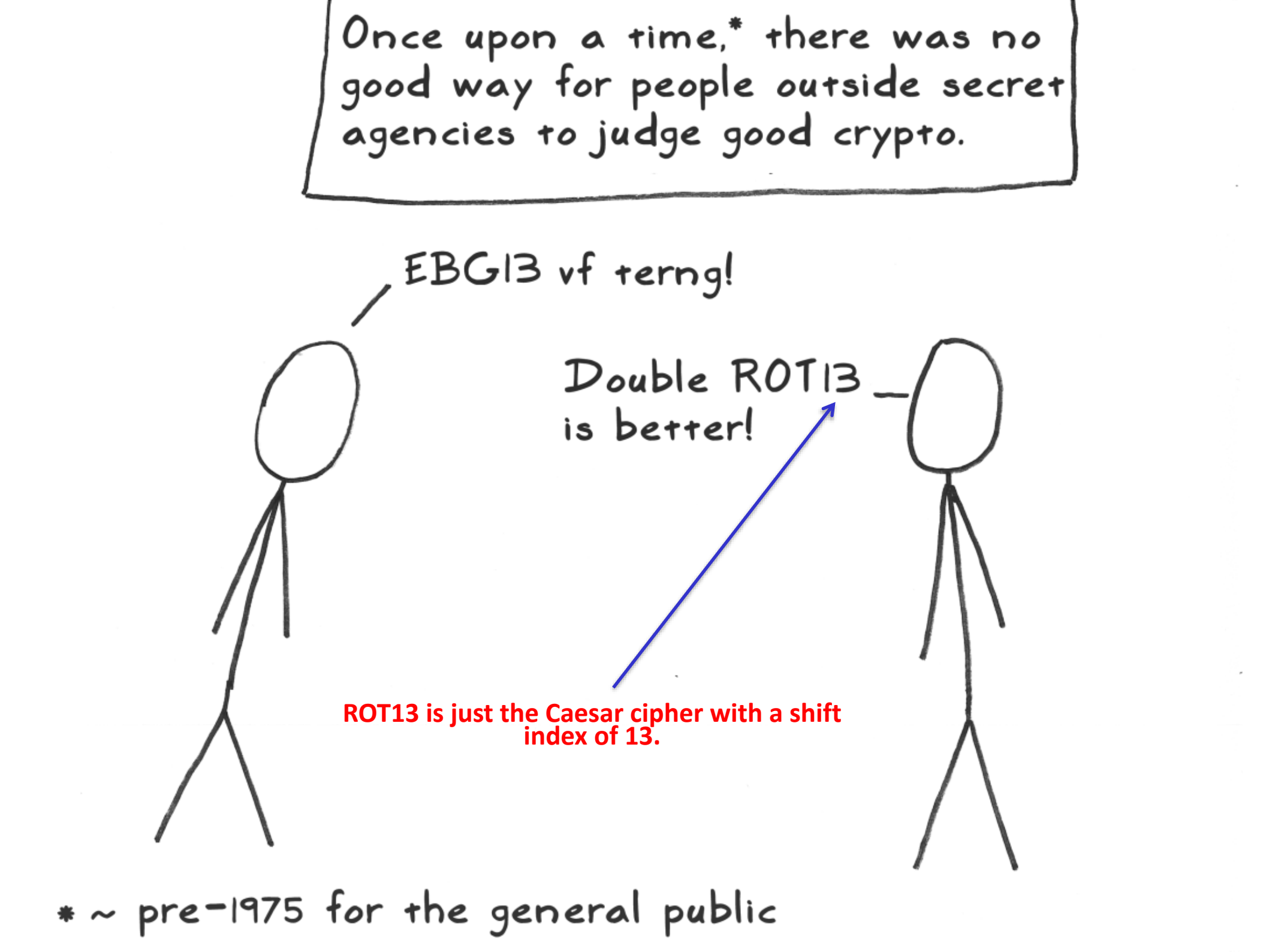


I've got a better-than-Cinderella story as I made my way to become king of the block cipher world.



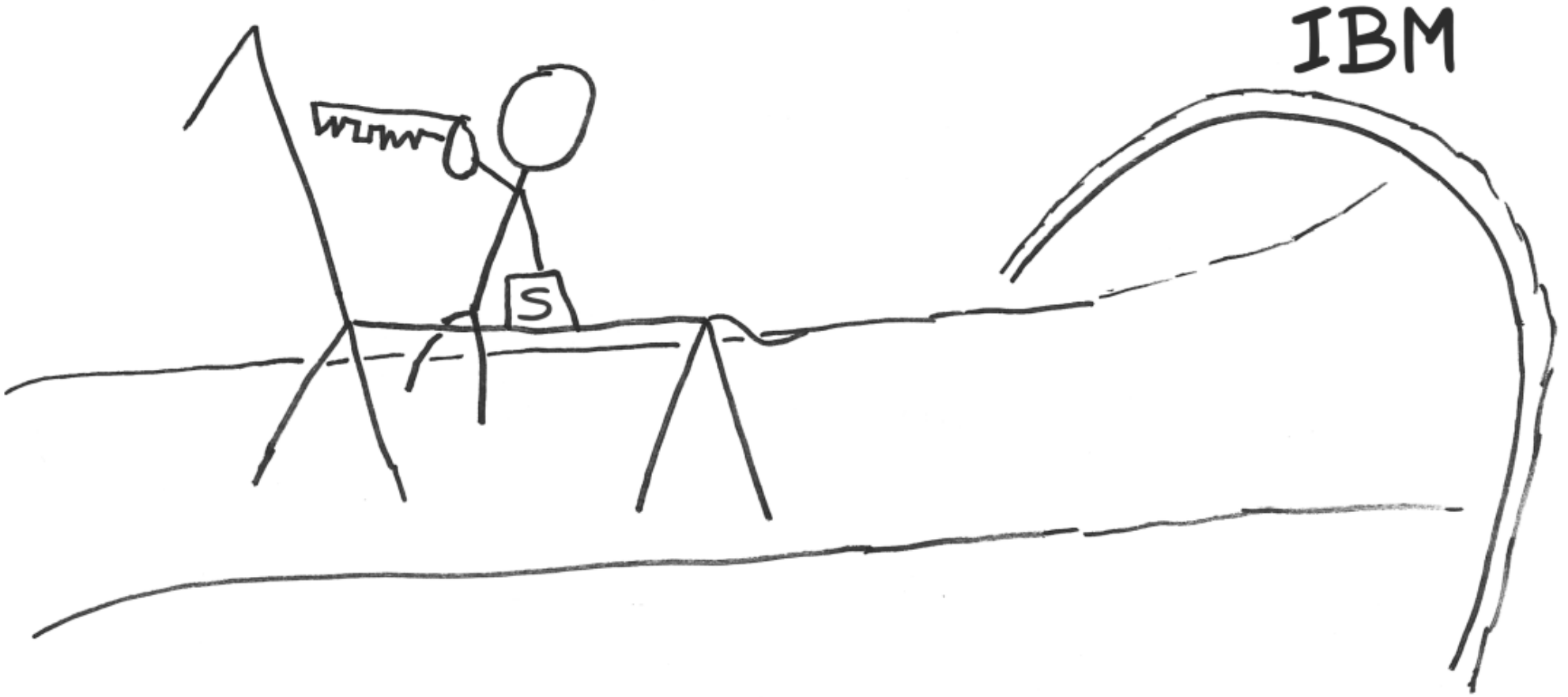
Whoa! You're still there. You want to hear it? Well let's get started...



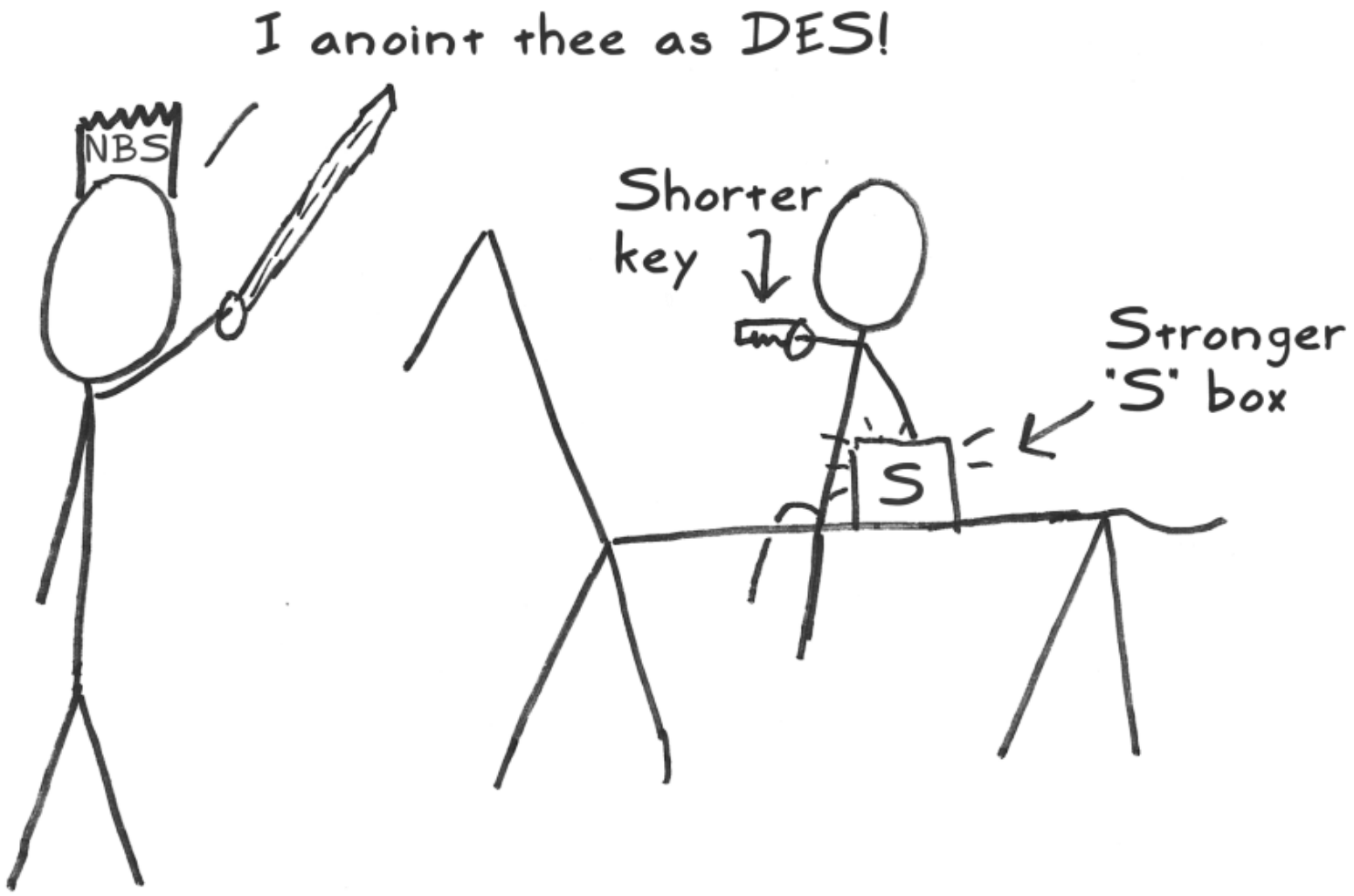

Once upon a time,* there was no good way for people outside secret agencies to judge good crypto.
EBG13 vf terng!
Double ROT13 is better!
ROT13 is just the Caesar cipher with a shift index of 13.
* ~ pre-1975 for the general public

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

One worthy competitor named Lucifer came forward.



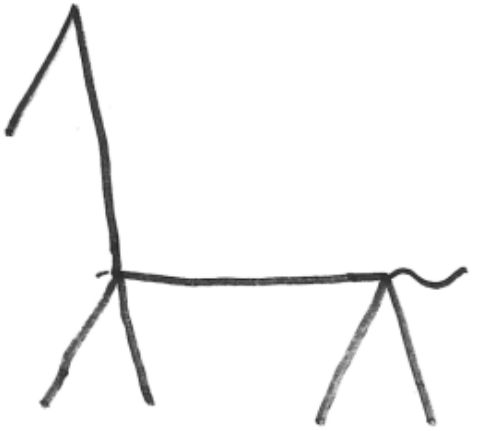
After being modified by the National Security Agency (NSA), he was anointed as the Data Encryption Standard (DES).



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."

NSA

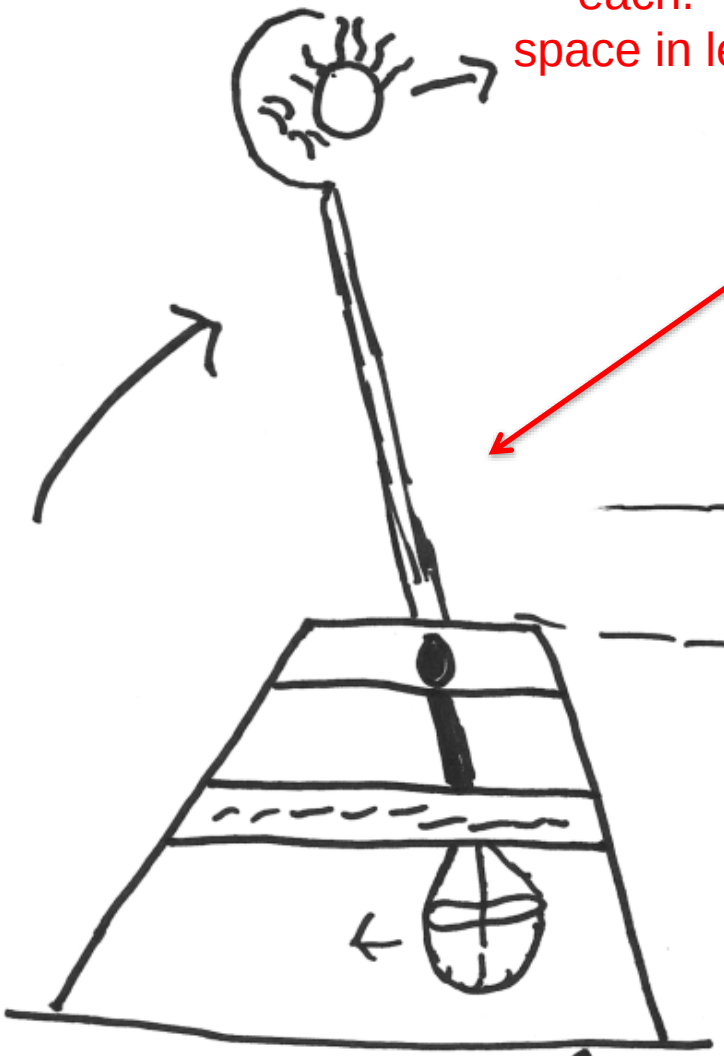


Check out that Feistel network!

Zzzz

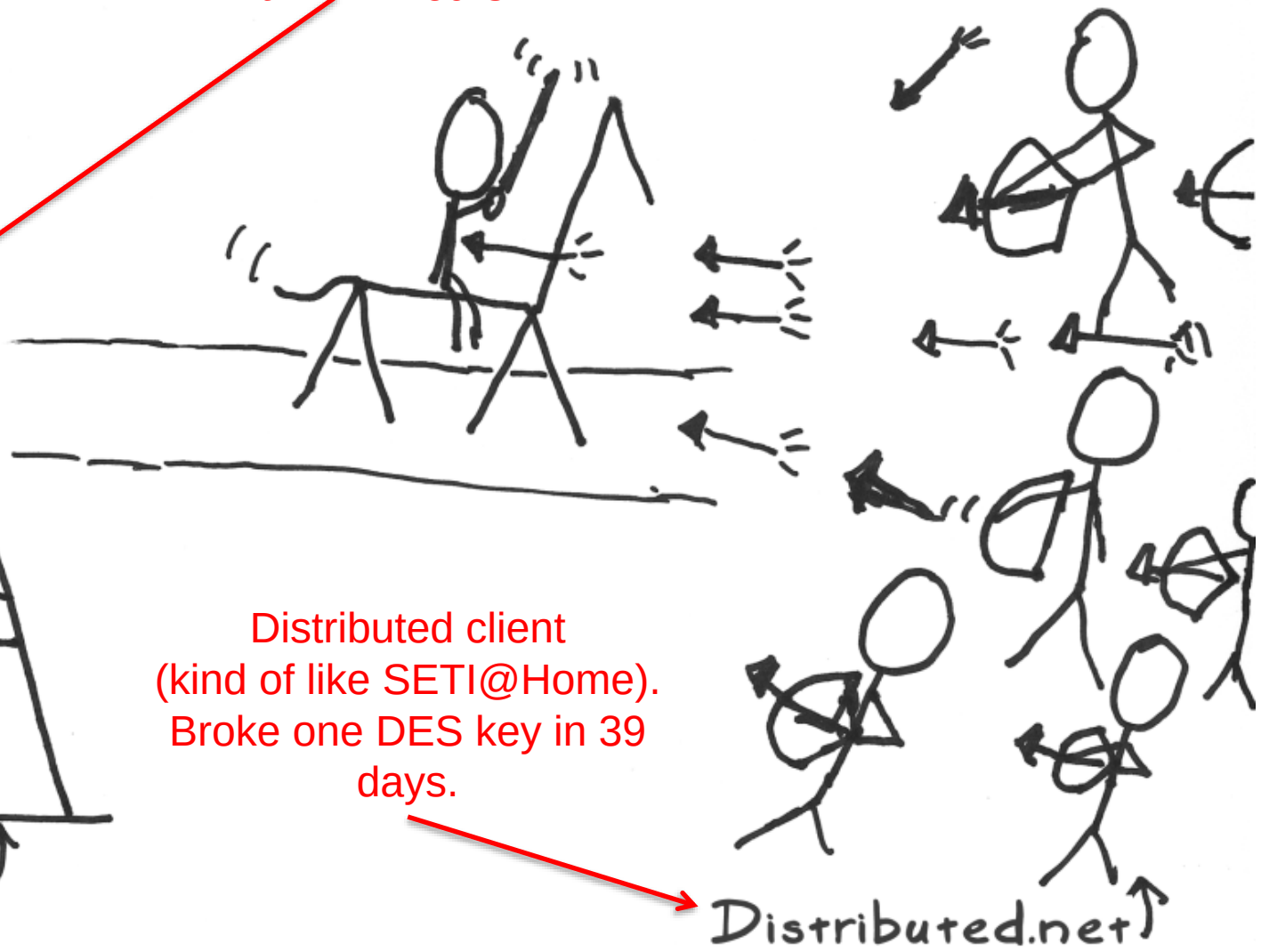
Over the years, many attackers challenged DES. He was defeated in several battles.

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.



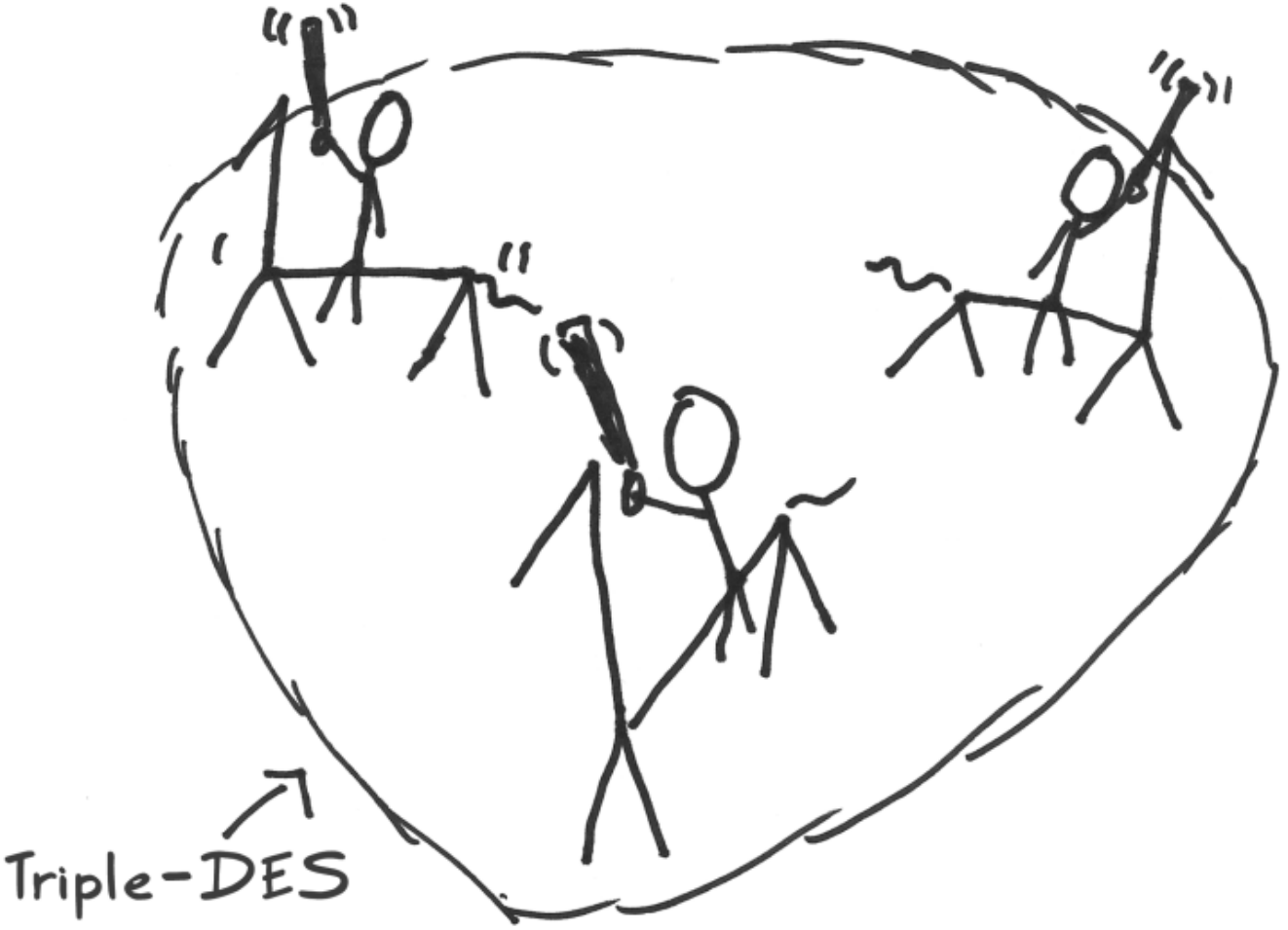
EFF's Deep Crack

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

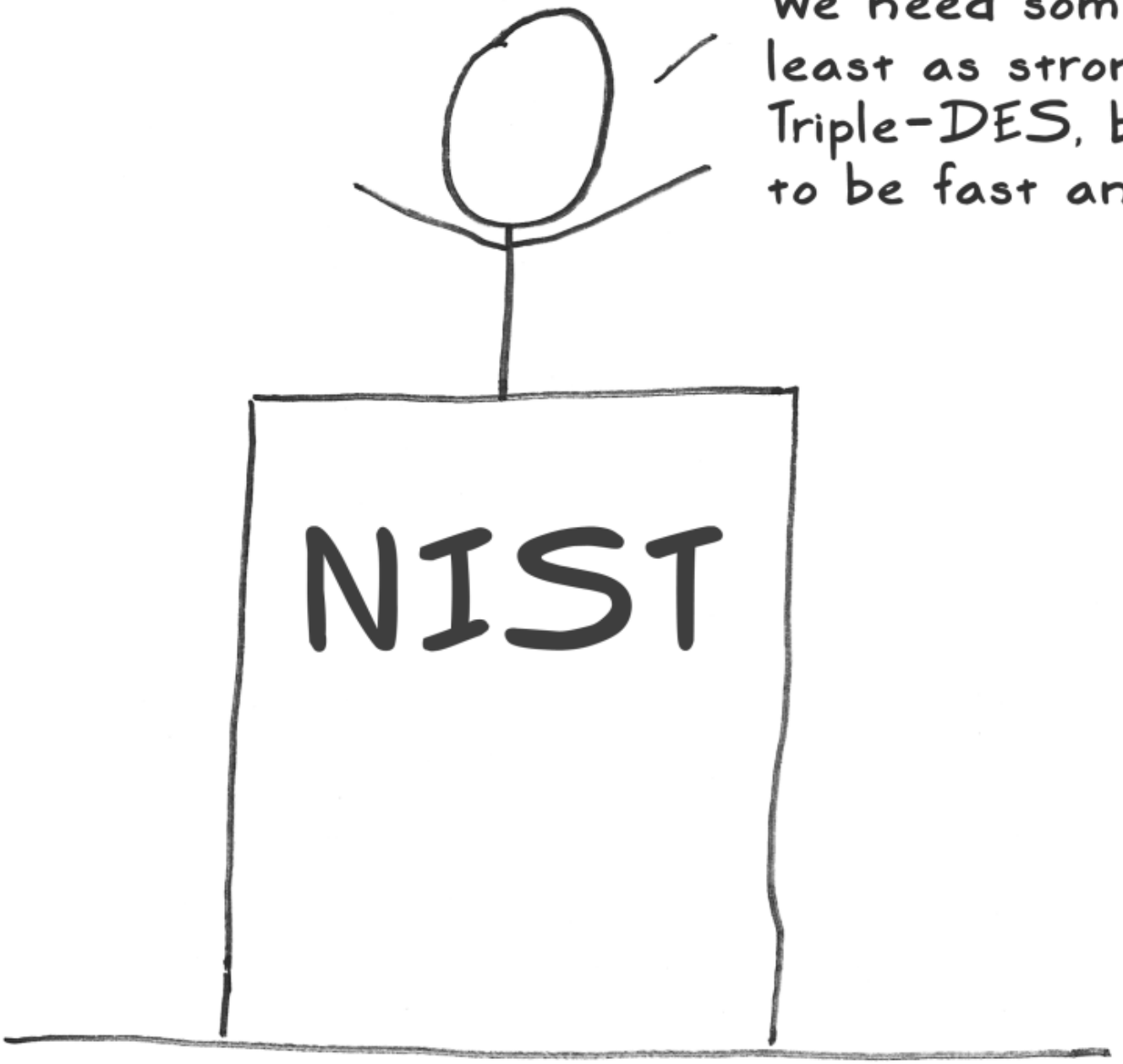


Distributed.net

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.

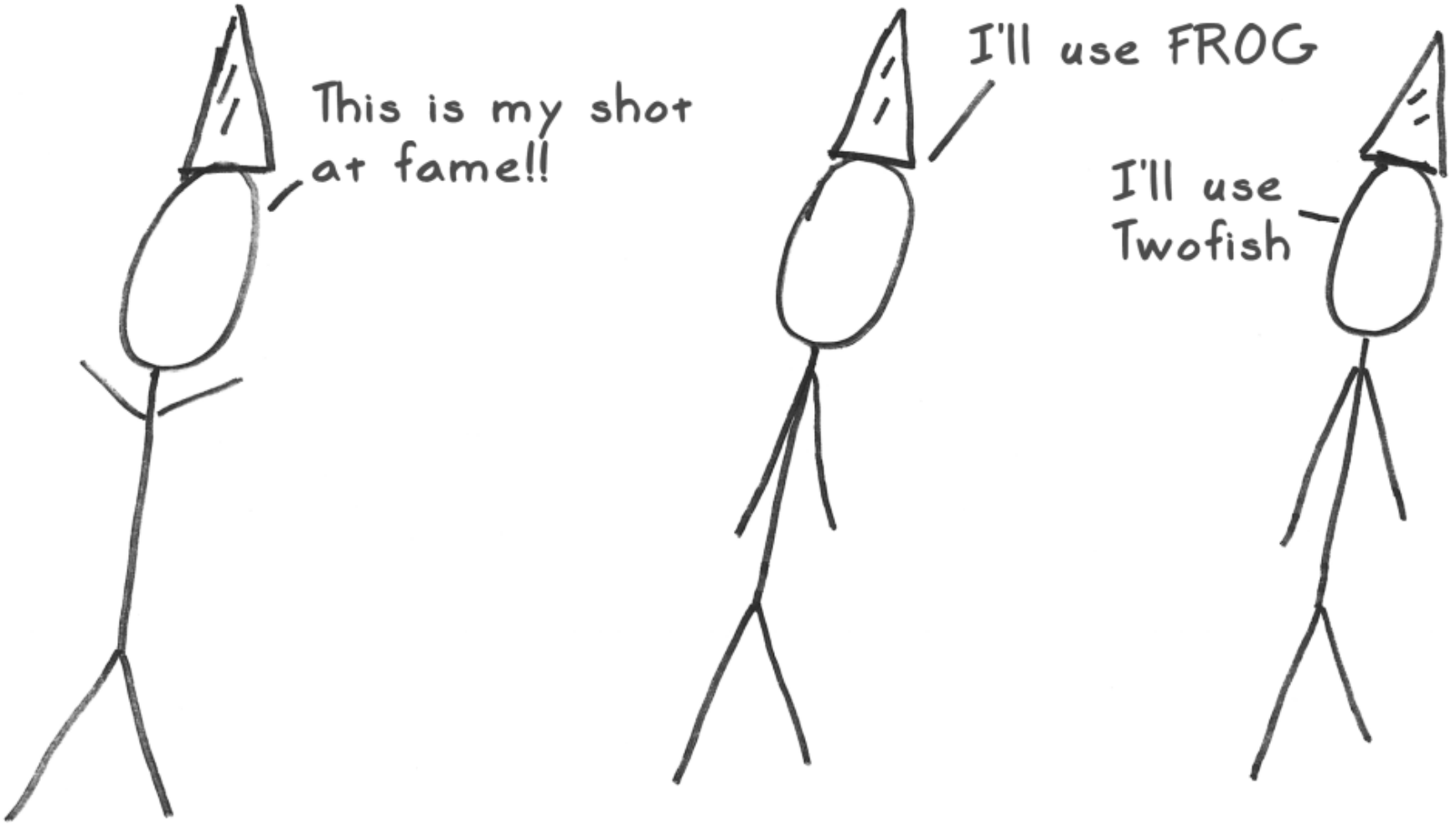


Another decree went out*...

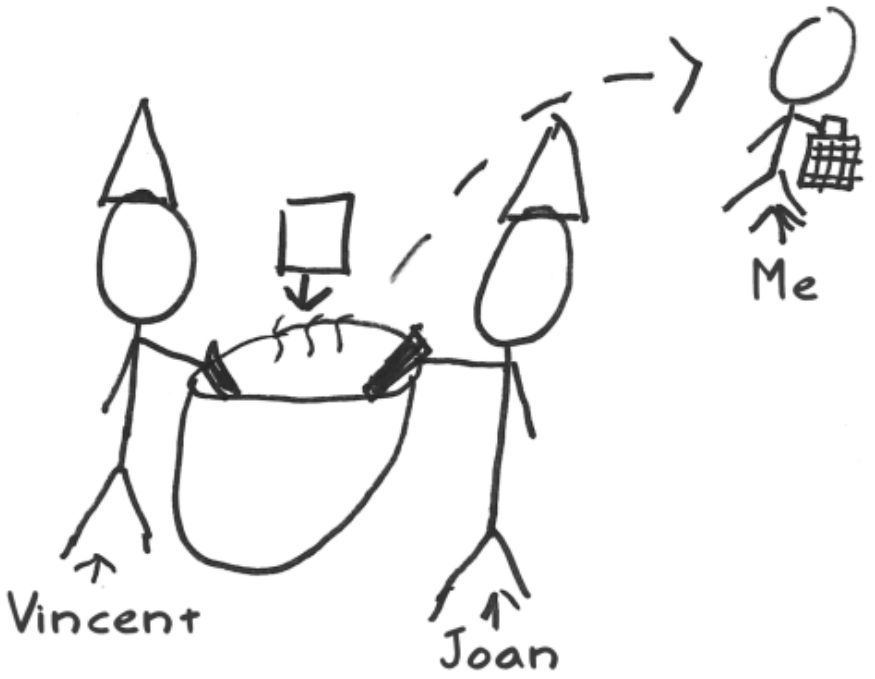


* ~ early 1997

This call rallied the crypto wizards to develop something better.

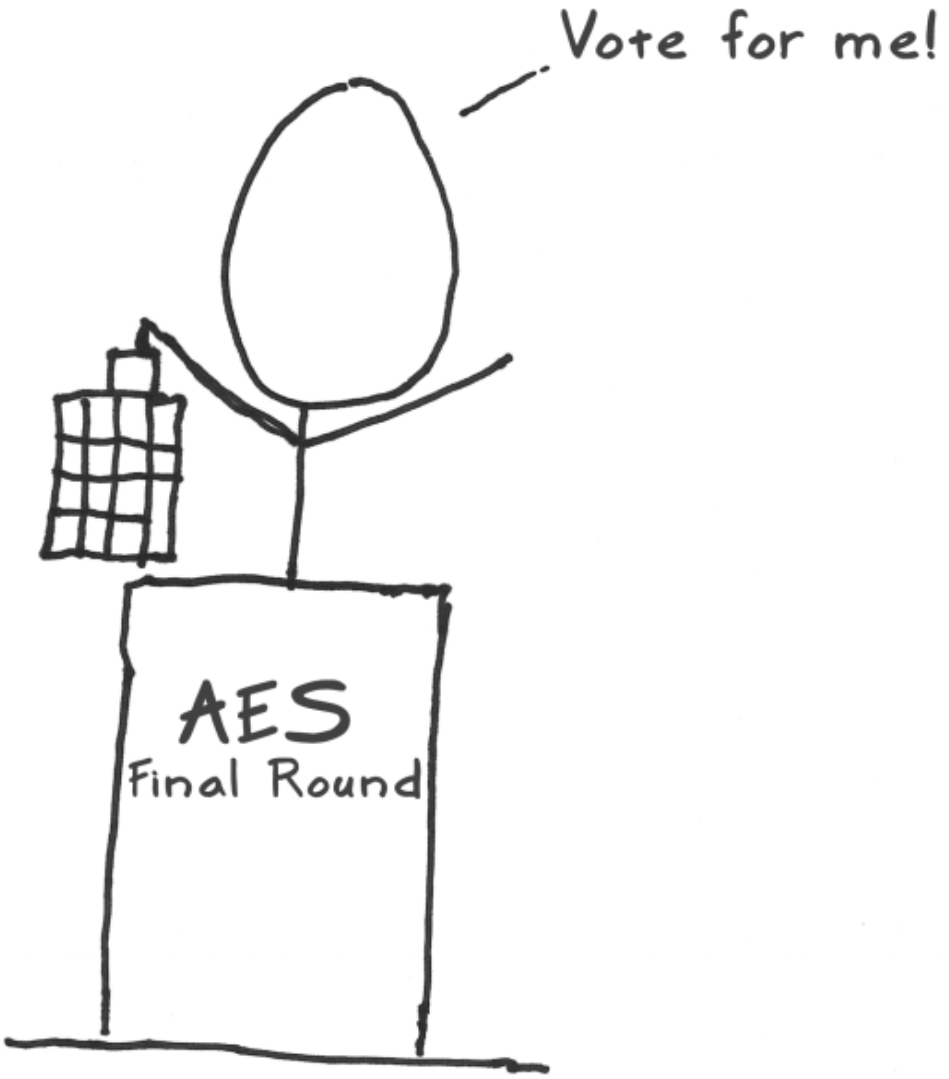


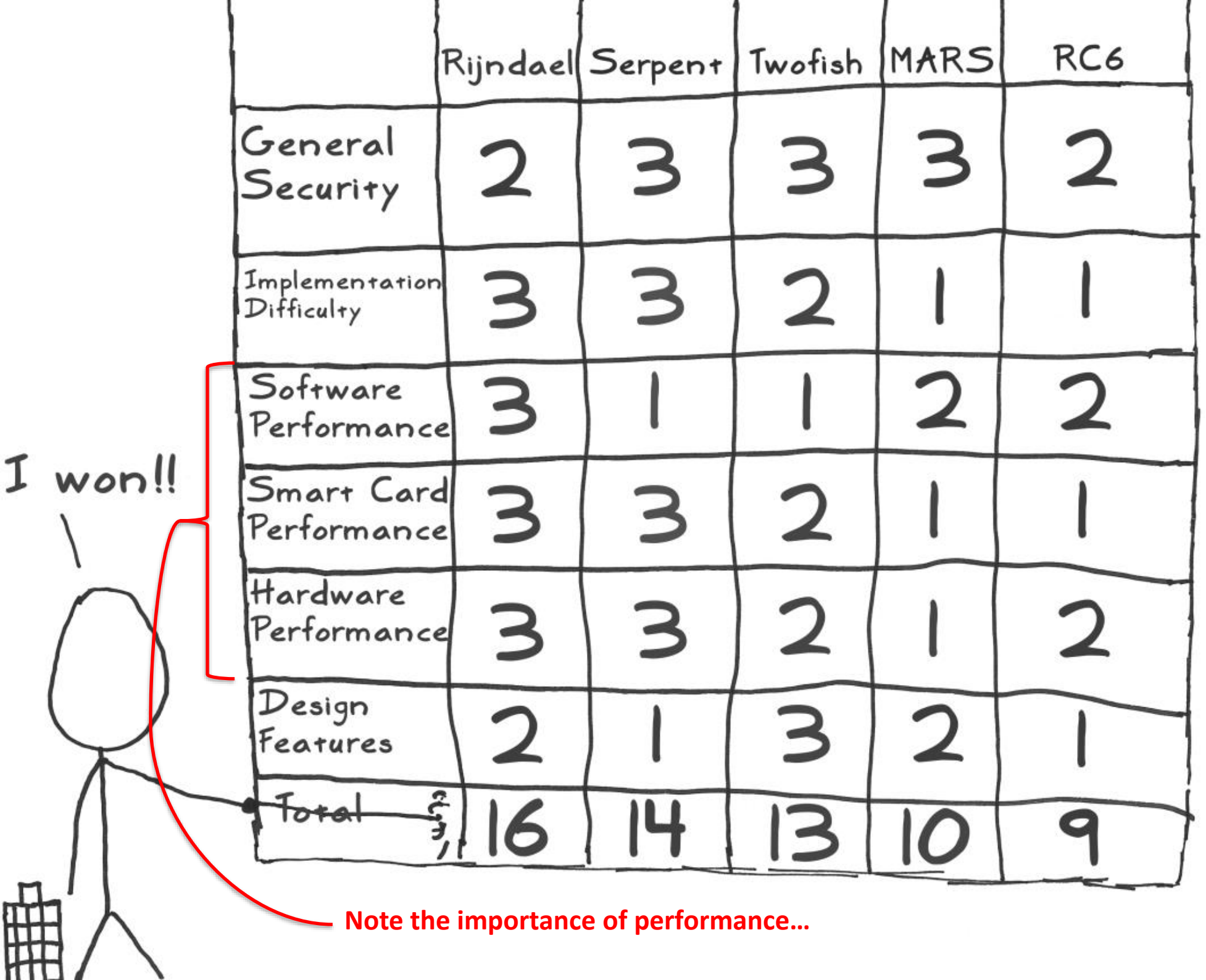
My creators, Vincent Rijmen and Joan Daemen, were among these crypto wizards. They combined their last names to give me my birth name: Rijndael.*



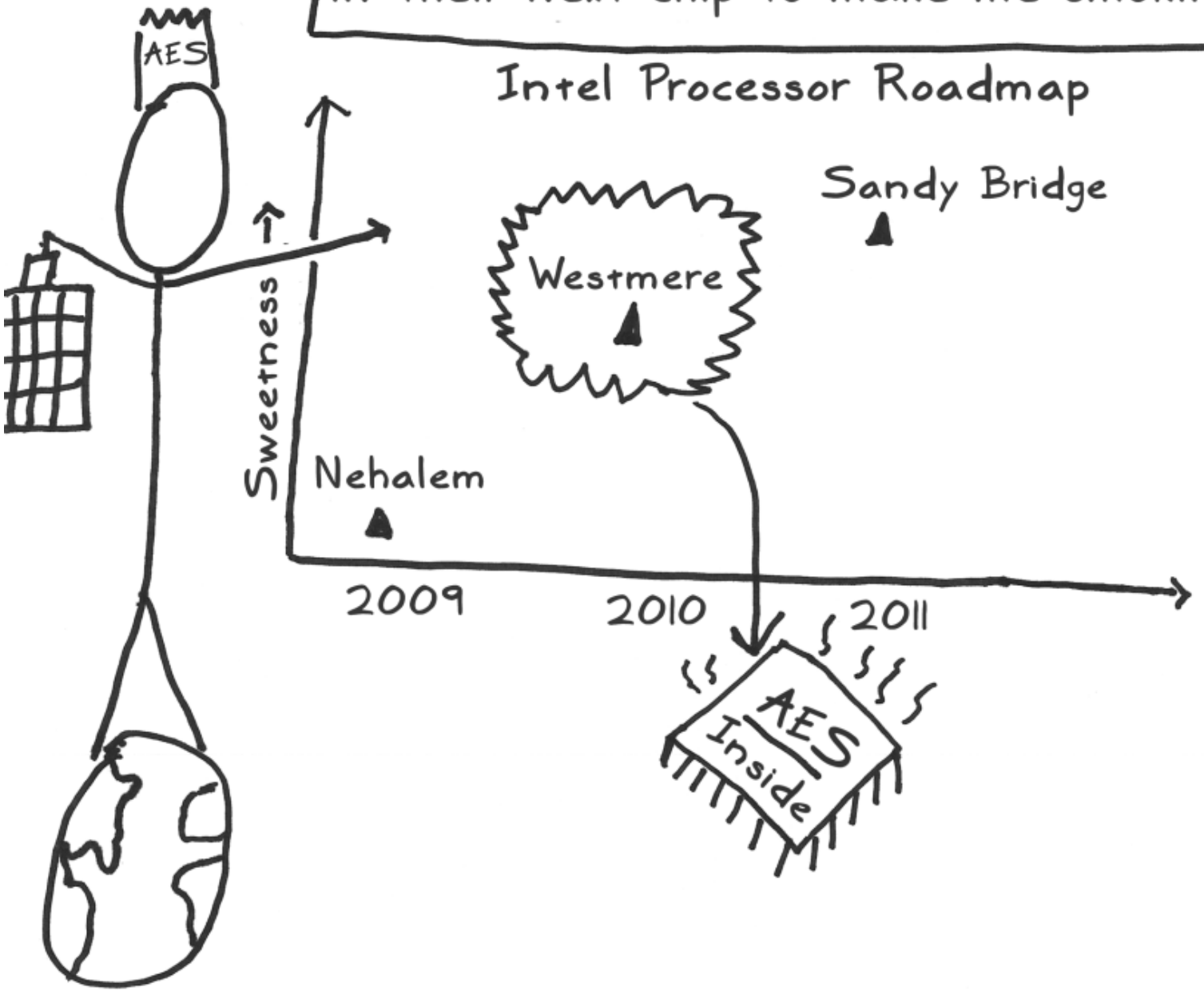
* That's pronounced "Rhine Dahl" for the non-Belgians out there.

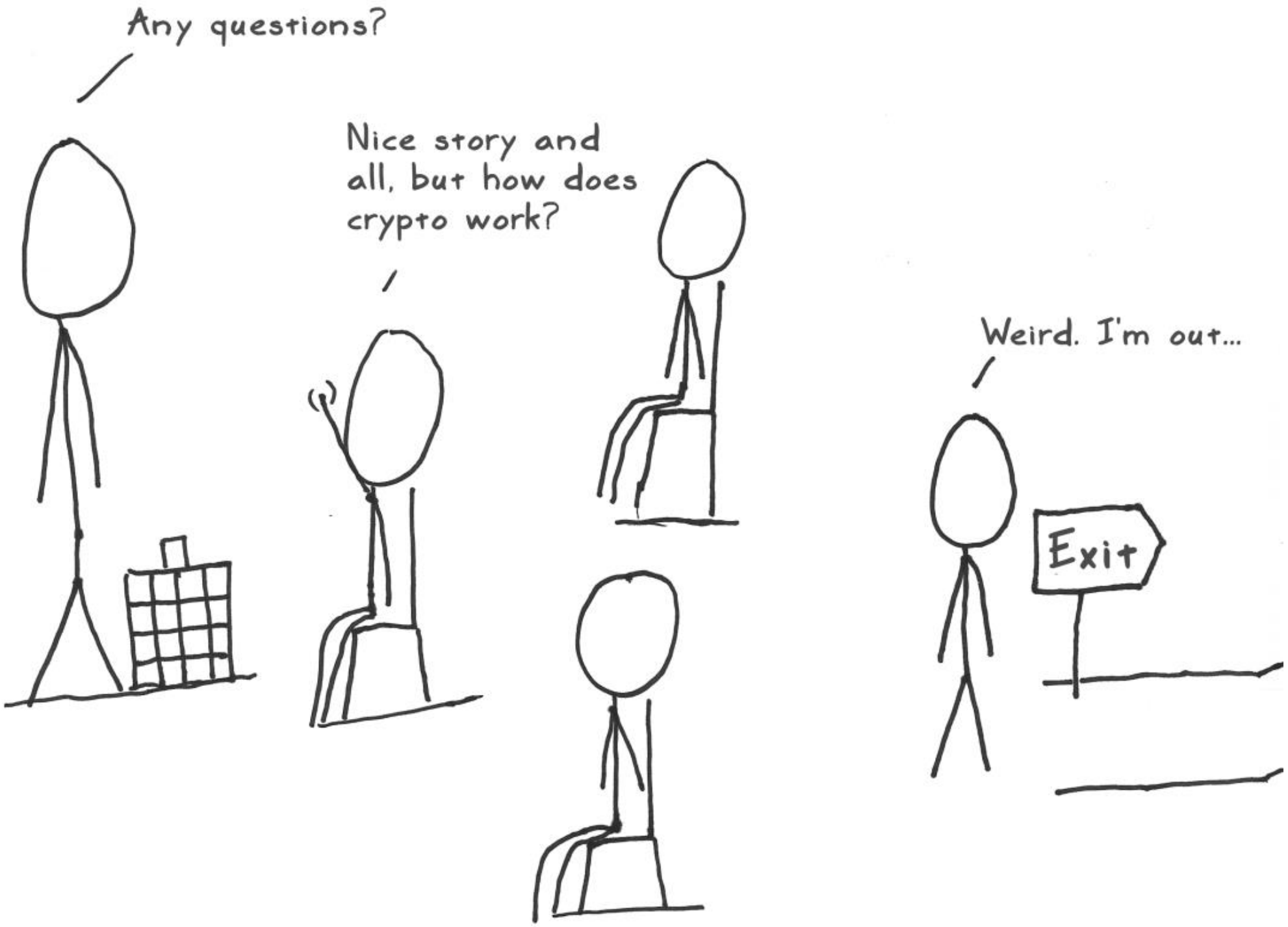
Everyone got together to vote and...




I won!!
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
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!



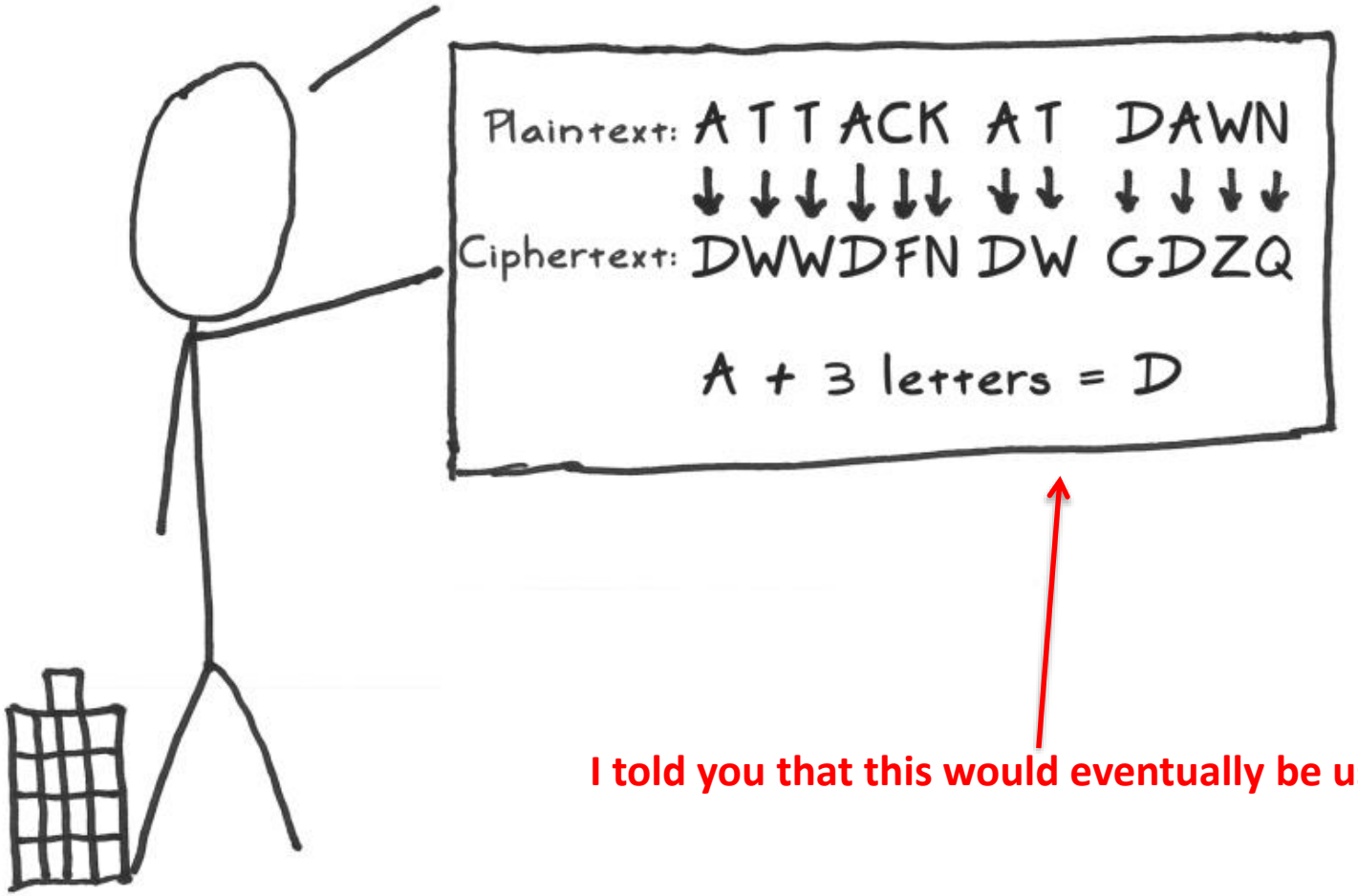

Any questions?
Nice story and all, but how does crypto work?
Weird. I'm out...
Exit

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:



I told you that this would eventually be useful…

Big Idea #2: Diffusion

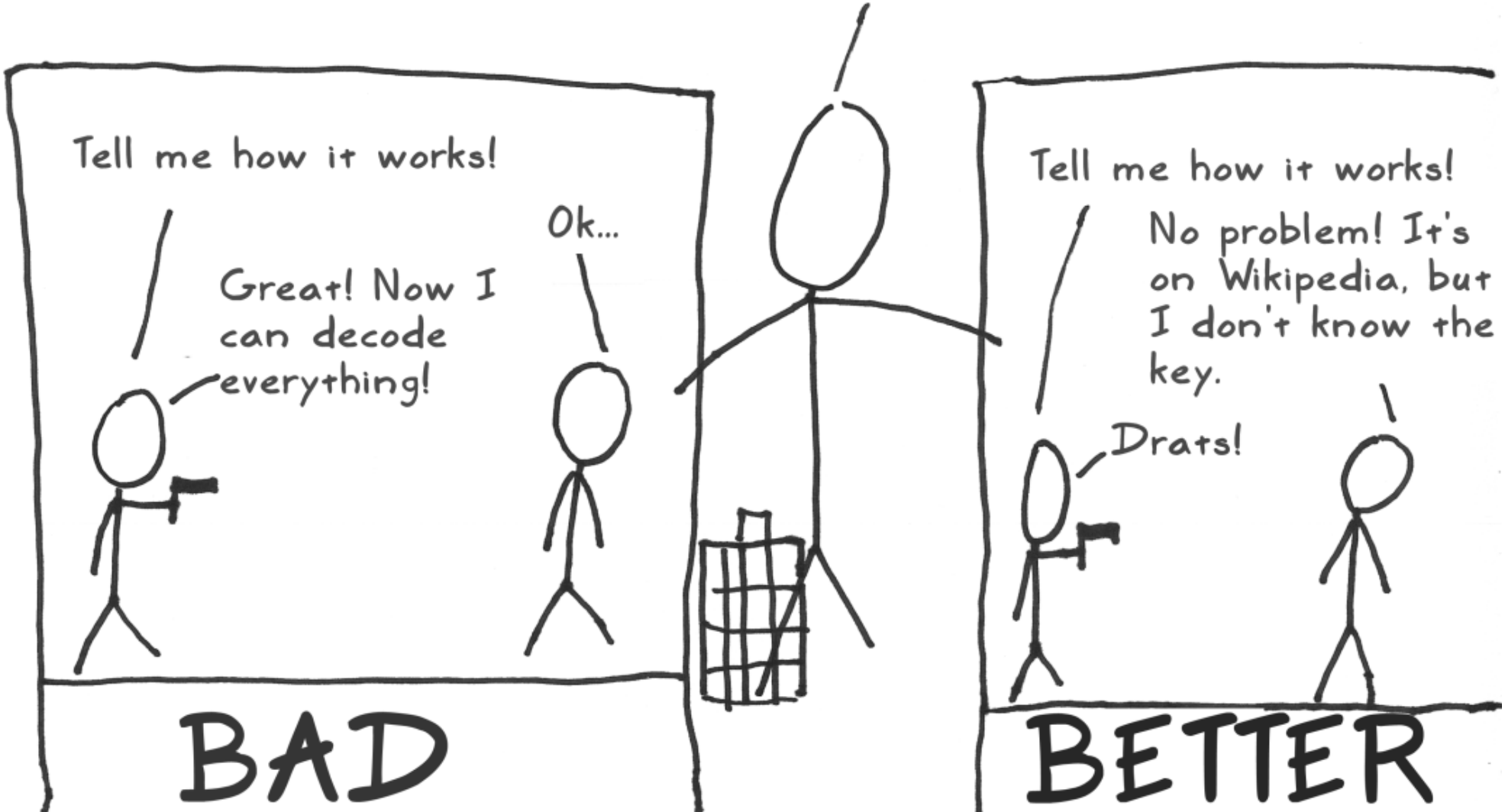
It's also a good idea to spread out the message. An example of this "diffusion" is a simple column transposition:

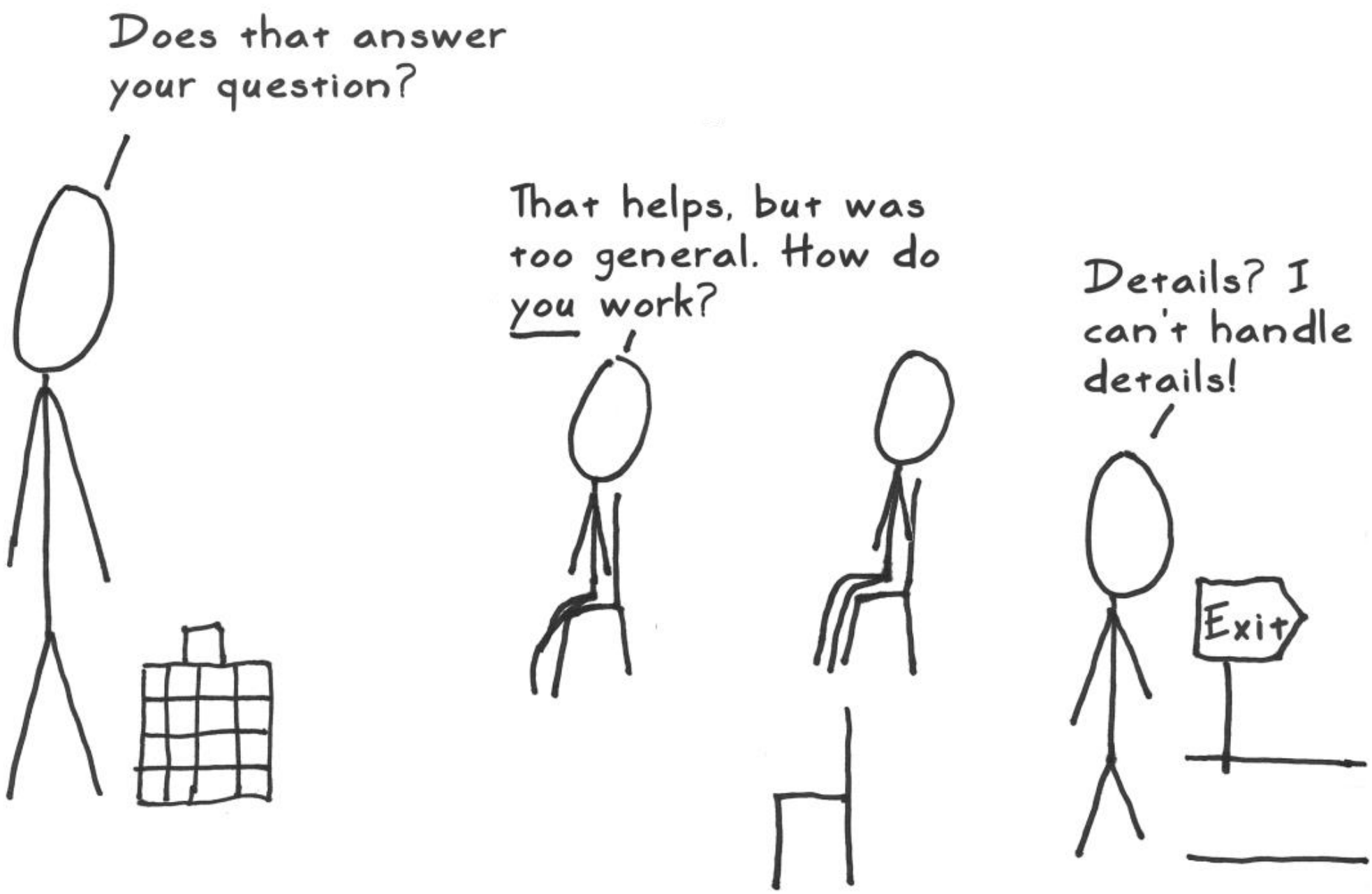
This is the “rail fence” cipher that we talked about last week

Big Idea #3: Secrecy Only in the Key

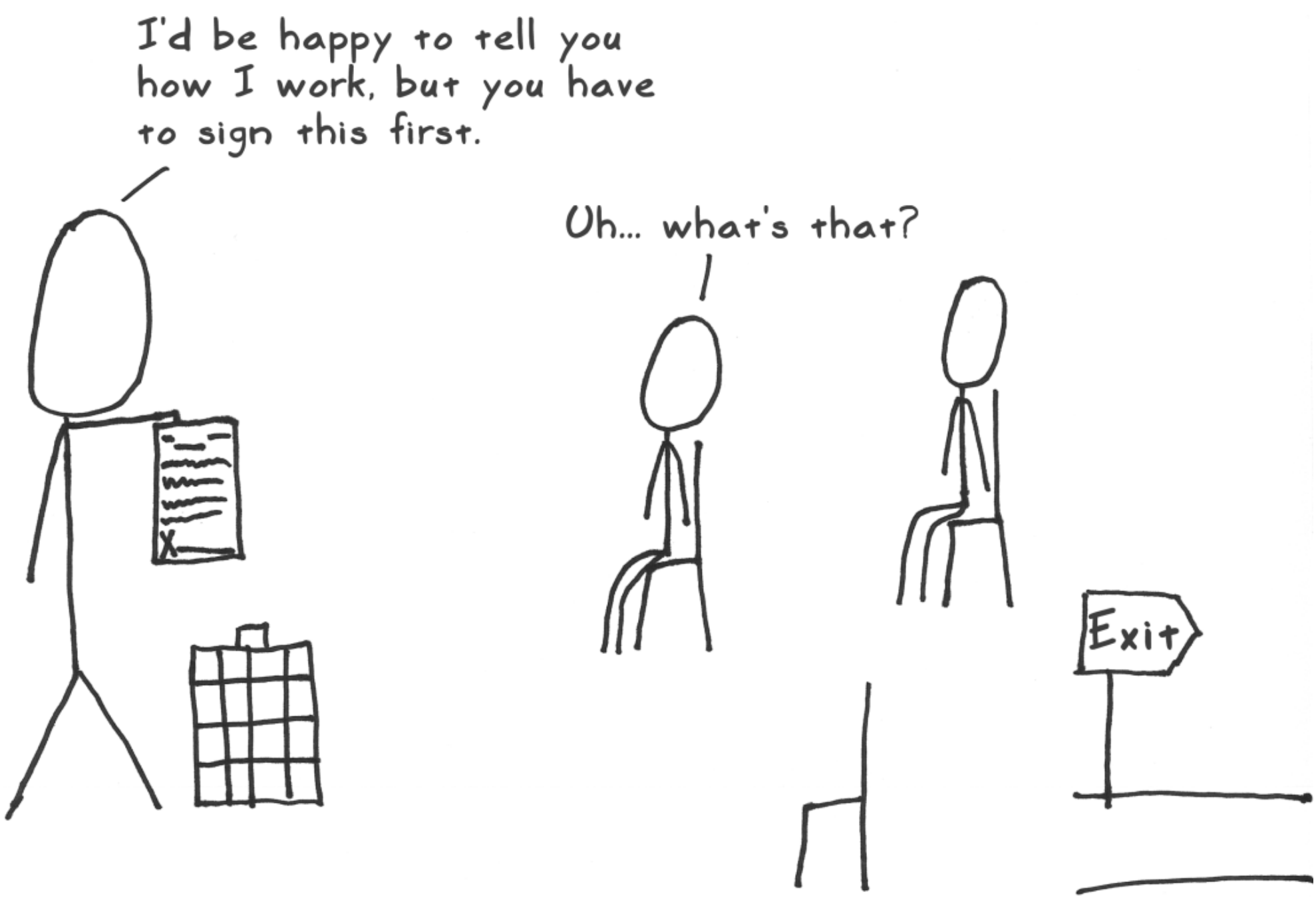
Kerchoffs' principle!

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!
Exit

Act 3: AES Basics

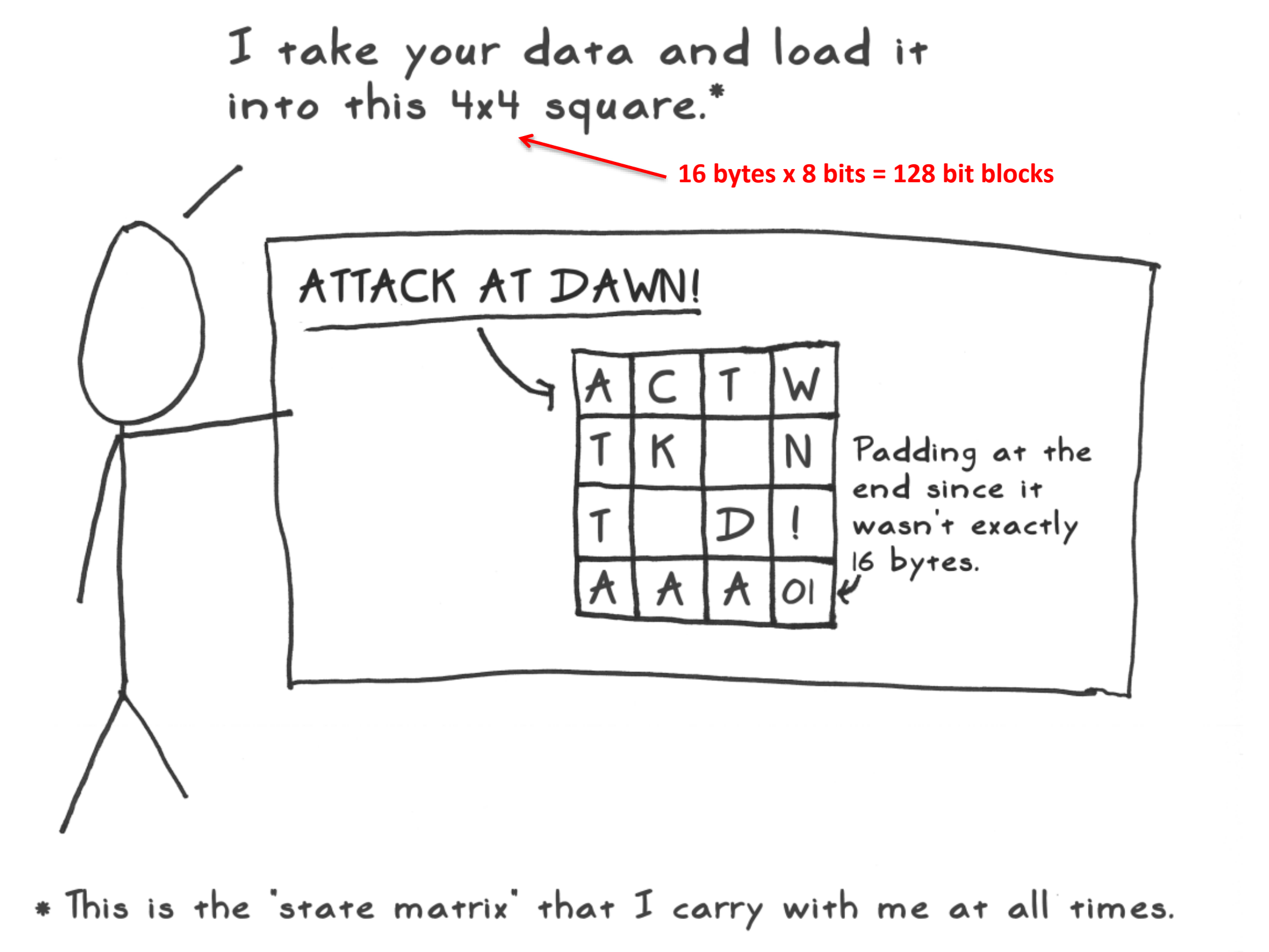

I'd be happy to tell you how I work, but you have to sign this first.
Uh... what's that?
Exit

Foot-Shooting Prevention Agreement

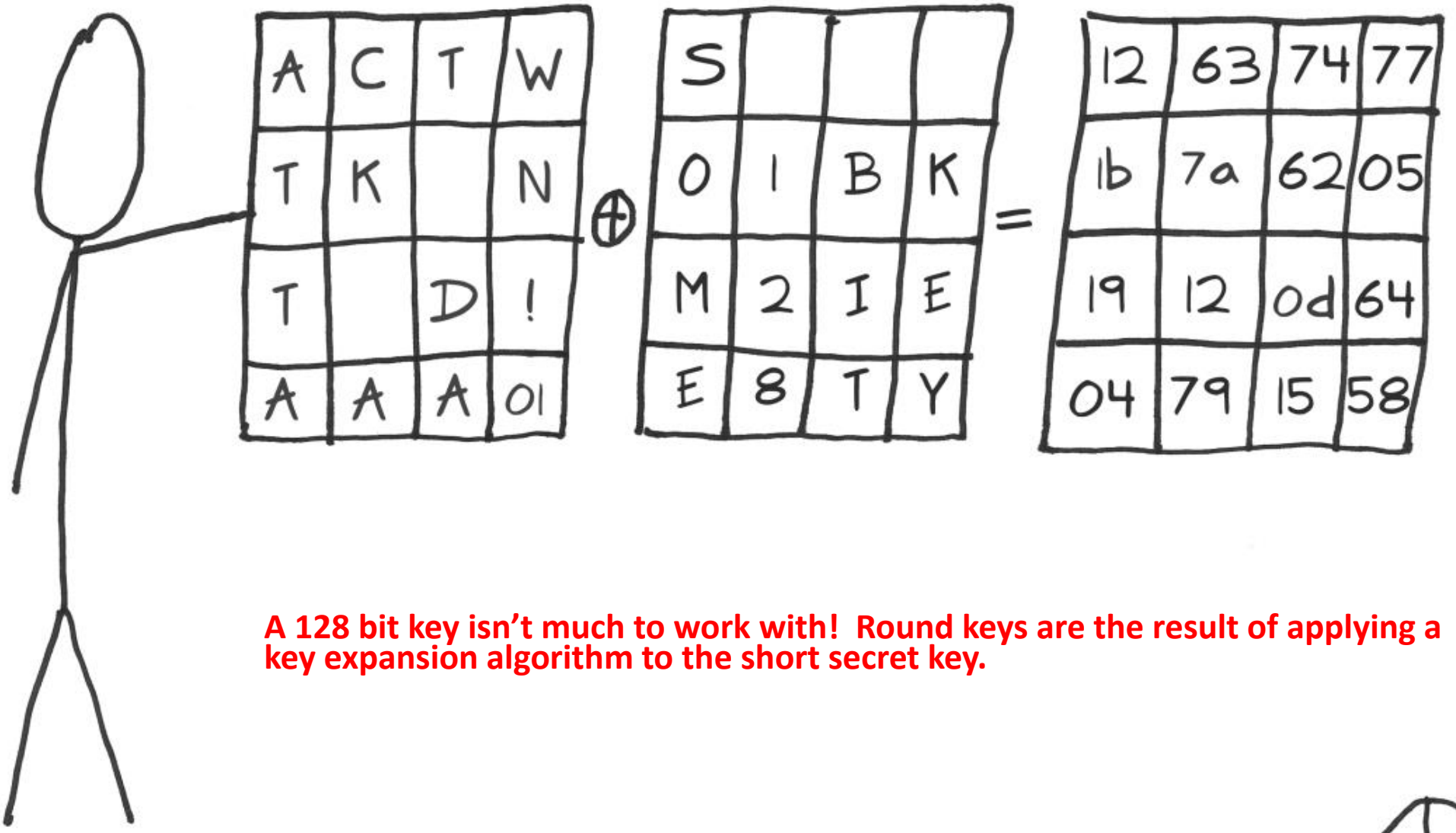
I, _____ , promise that once
Your Name
I see how simple AES really is, I will not implement it in production code even though it would be really fun.

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.*
16 bytes x 8 bits = 128 bit blocks
ATTACK AT DAWN!
A C T W
T K N
T D !
A A A 01
Padding at the end since it wasn't exactly 16 bytes.
* This is the "state matrix" that I carry with me at all times.

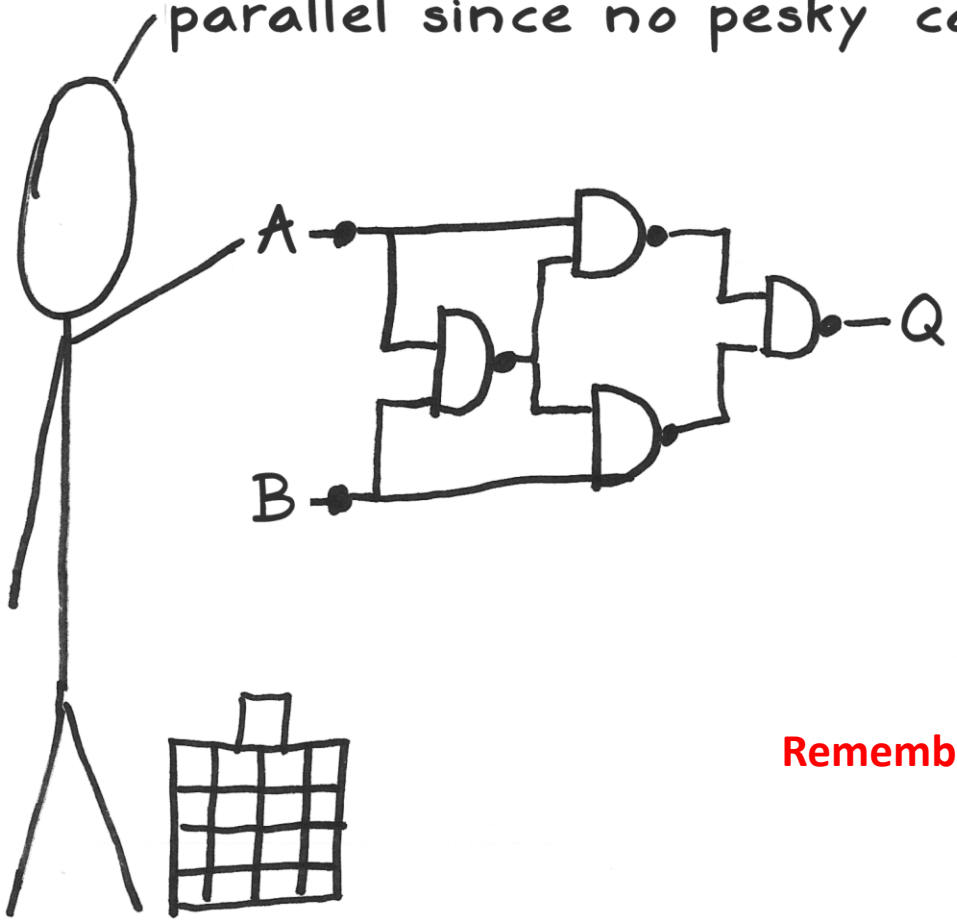
The initial round has me xor each input byte with the corresponding byte of the first round key.



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.

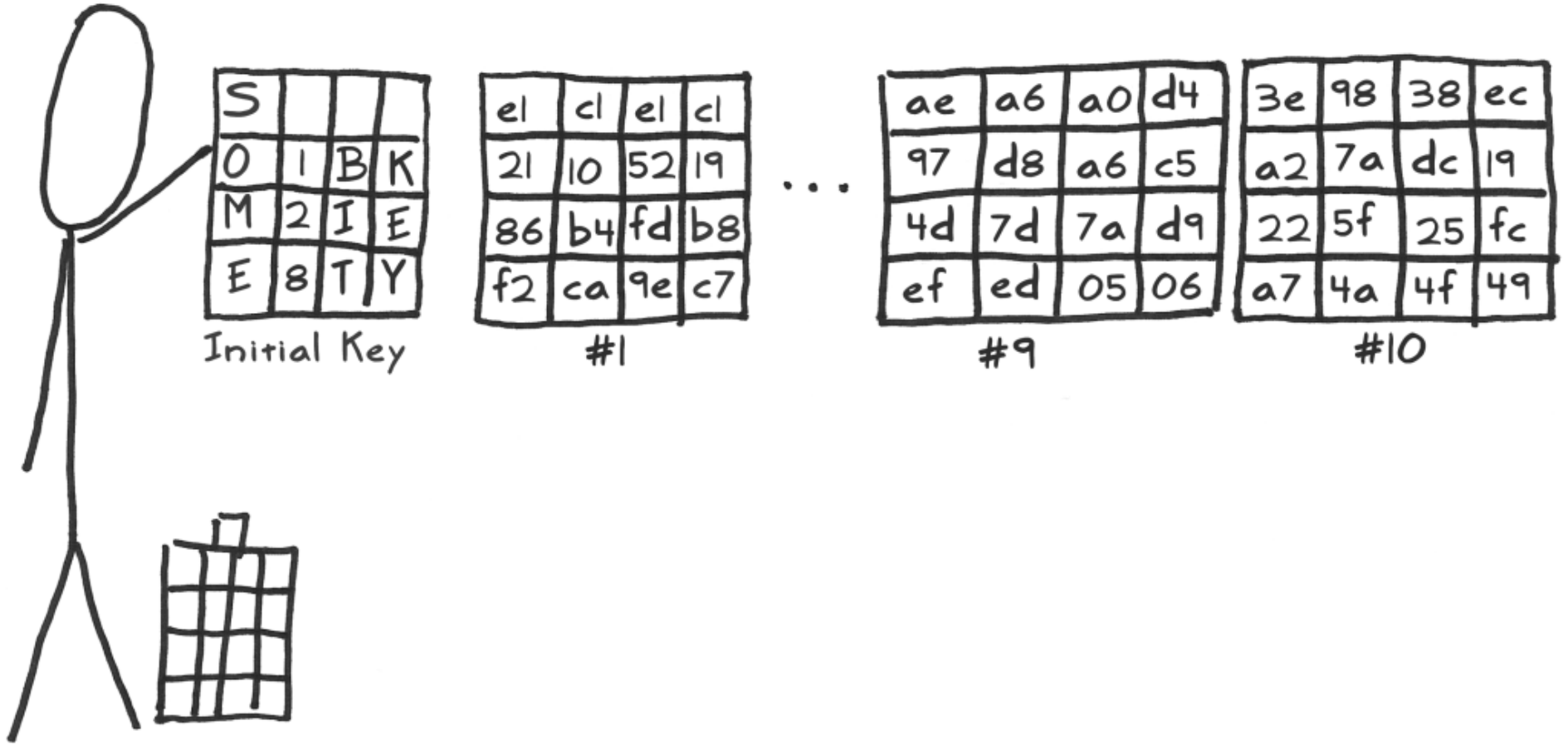


AES ♡ ⊕

Remember the performance focus of AES?

Key Expansion: Part 1

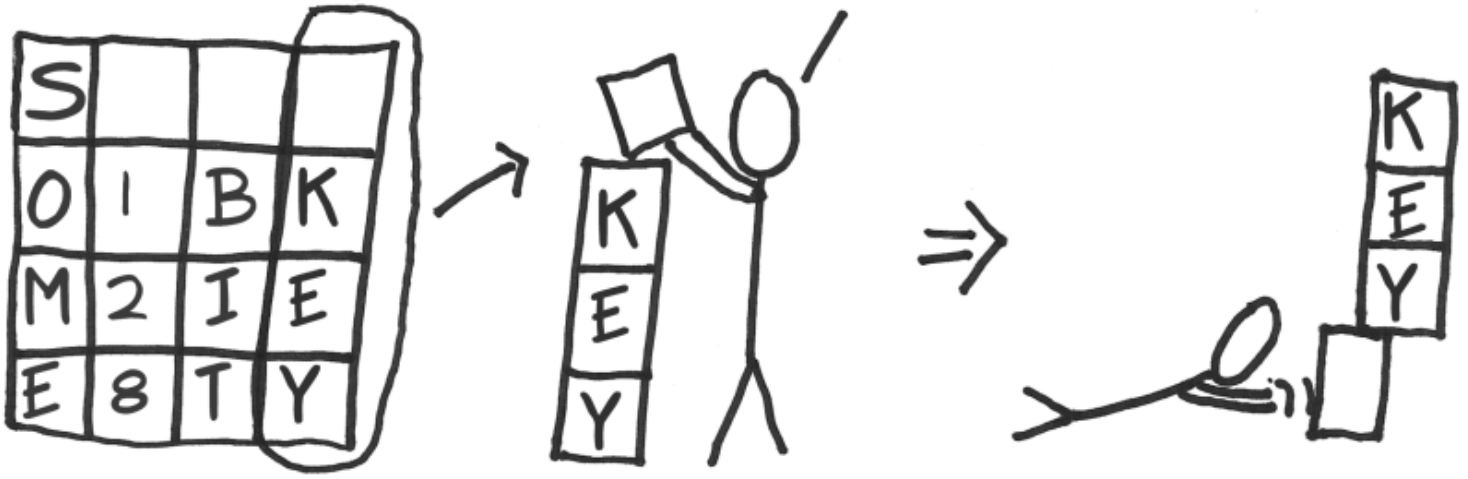
I need lots of keys for use in later rounds. I derive all of them from the initial key using a simple mixing technique that's really fast. Despite its critics,* it's good enough.



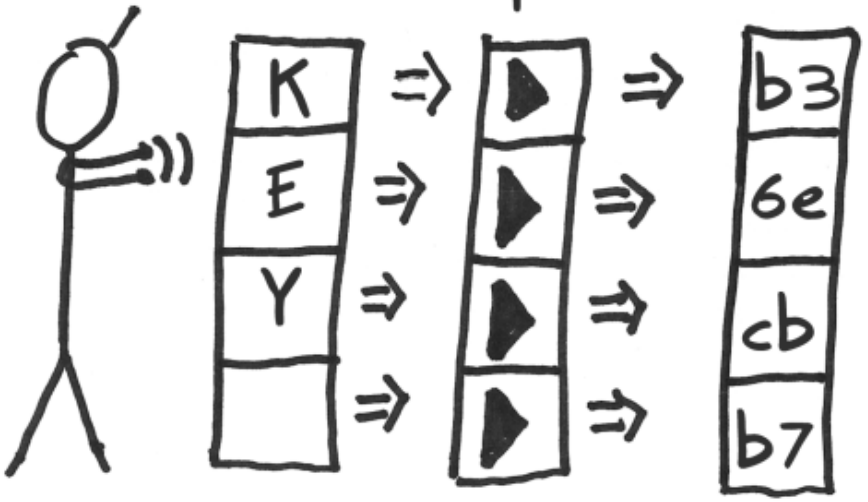
* By far, most complaints against AES's design focus on this simplicity.

Key Expansion: Part 2a

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

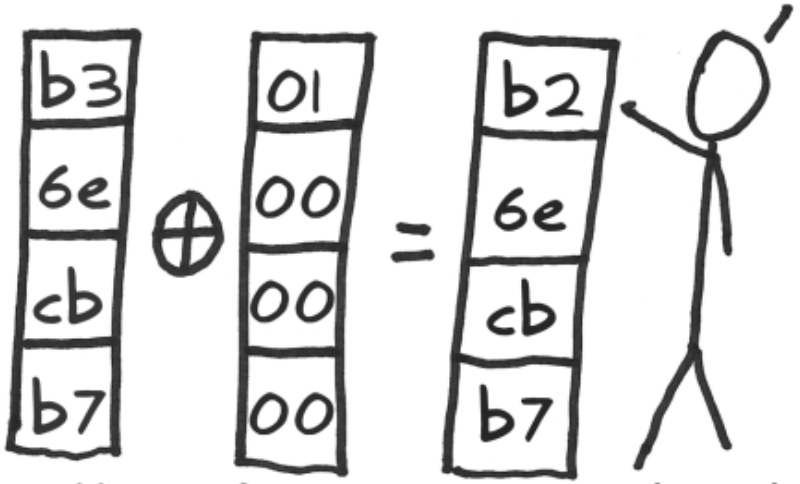


(2) Next, I run each byte through a substitution box that will map it to something else:

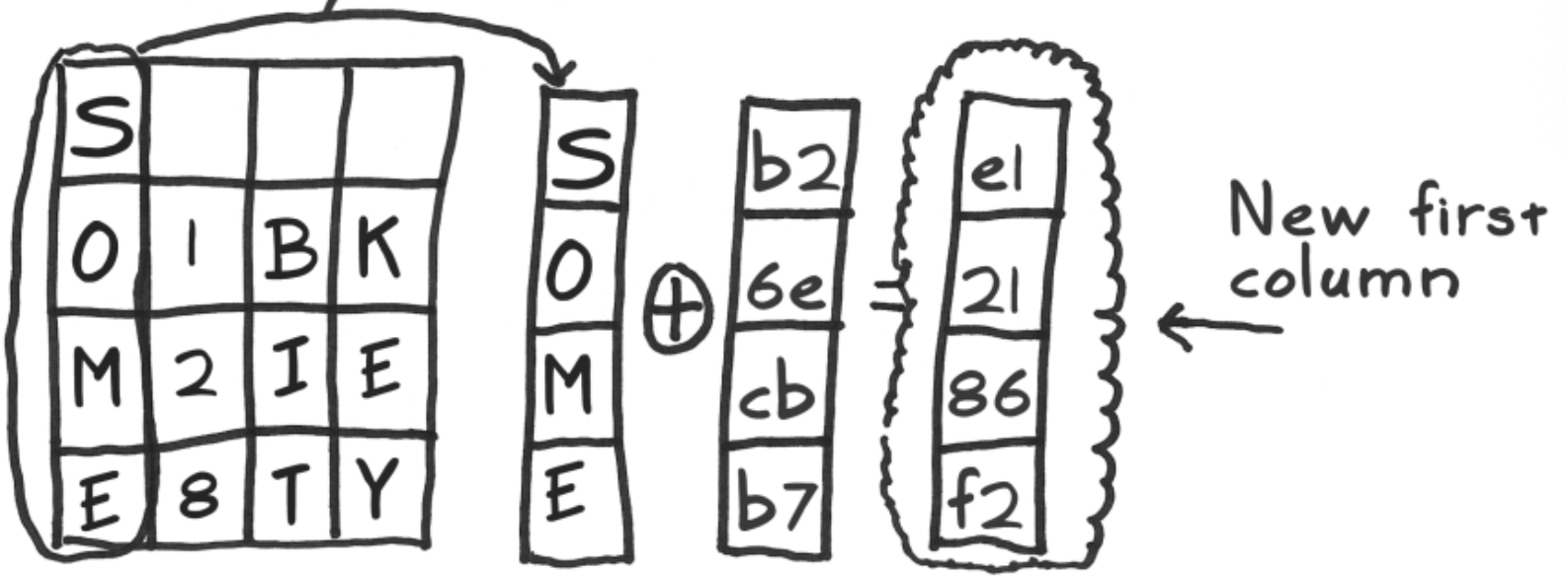


Key Expansion: Part 2b

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

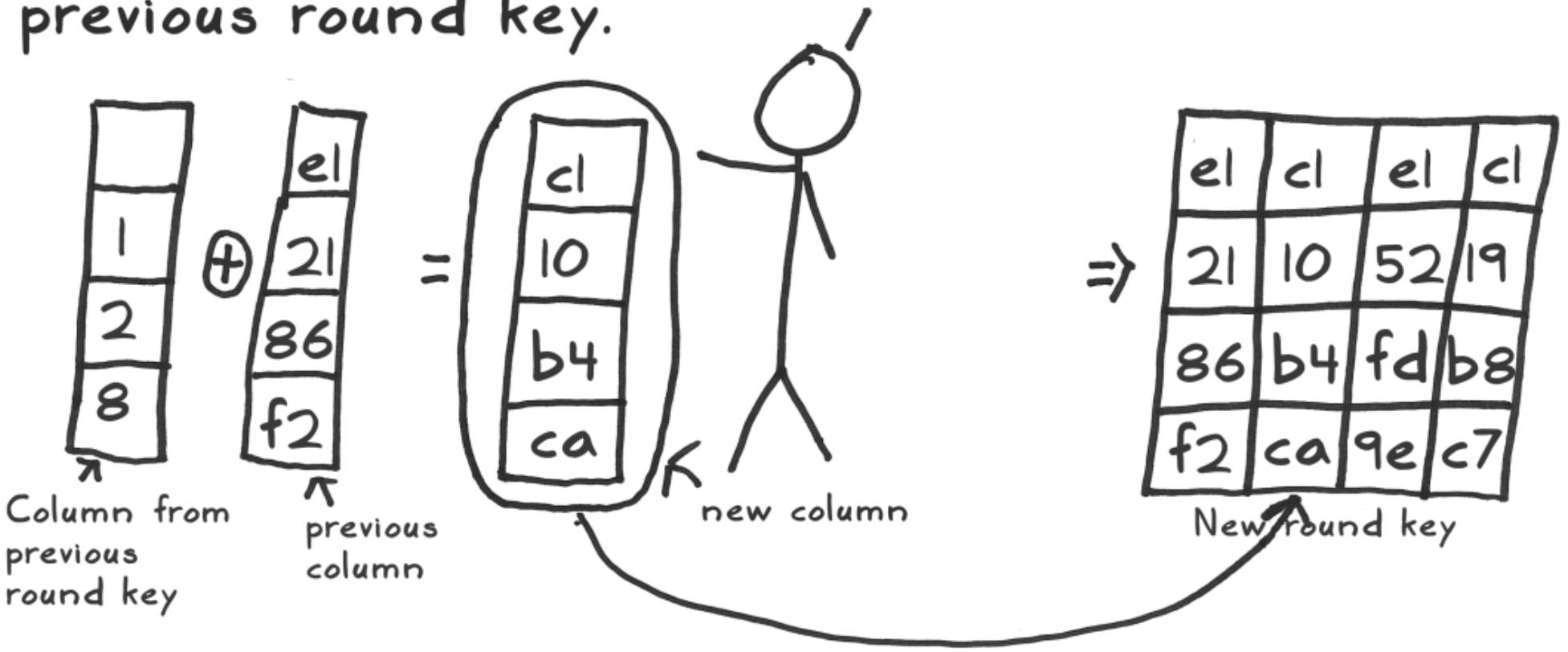


(4) Finally, I xor it with the first column of the previous round key:



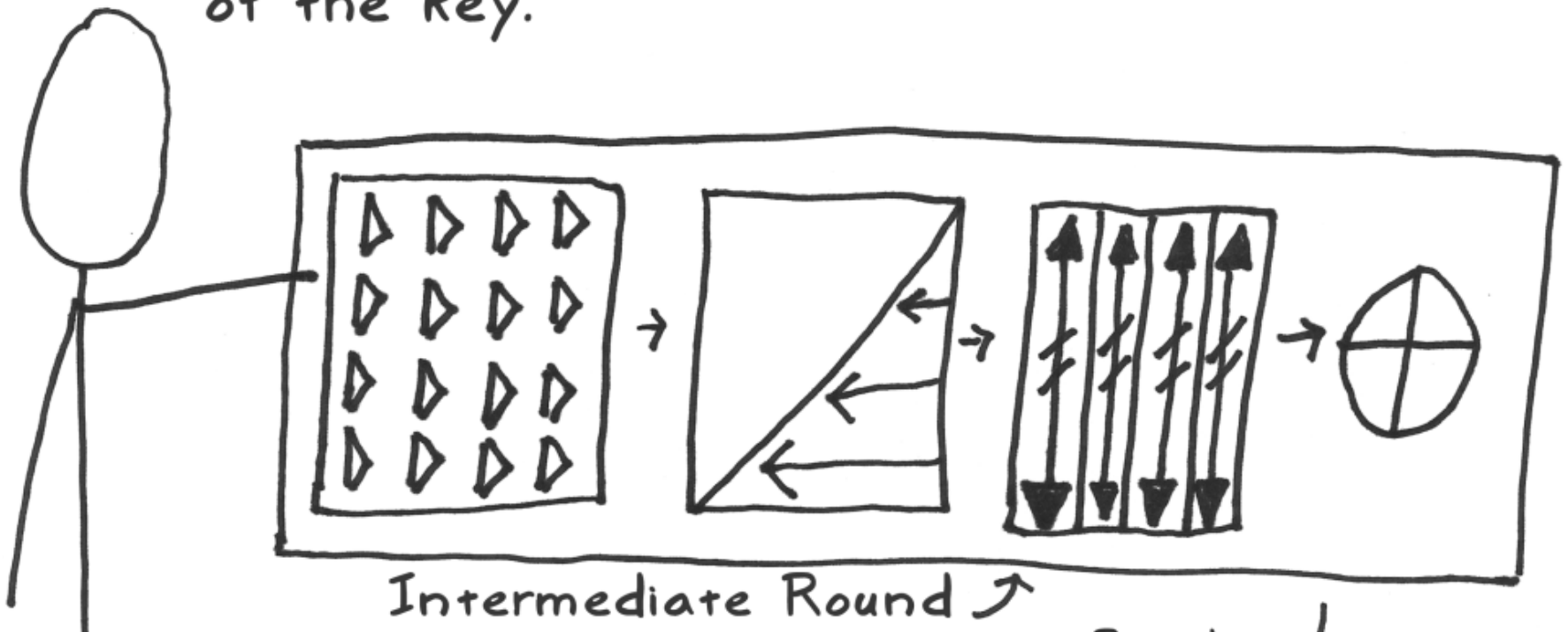
Key Expansion: Part 3

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

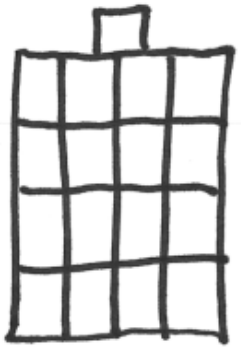


* Note that 256 bit keys are slightly more complicated.

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.

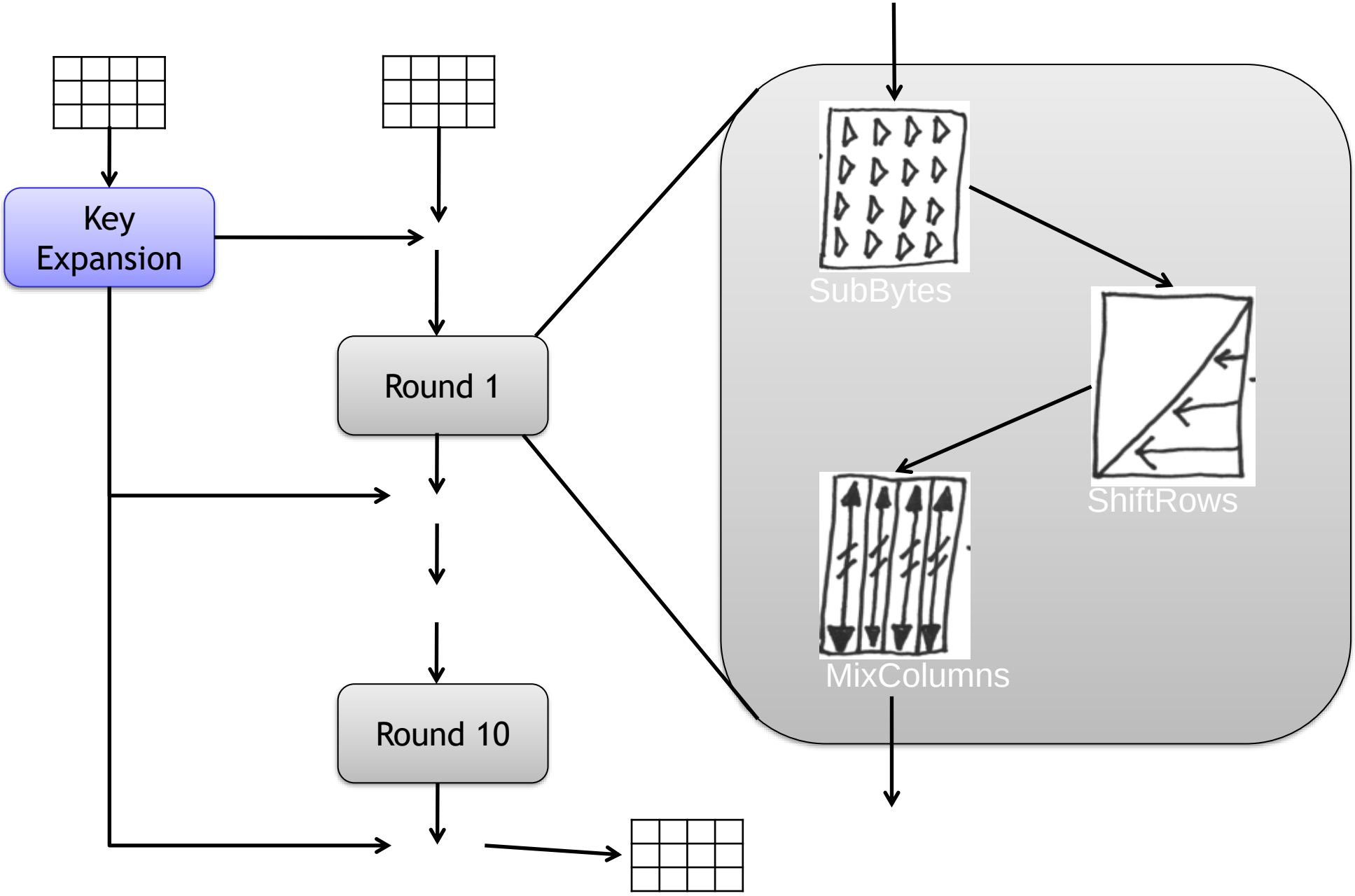


Intermediate Round



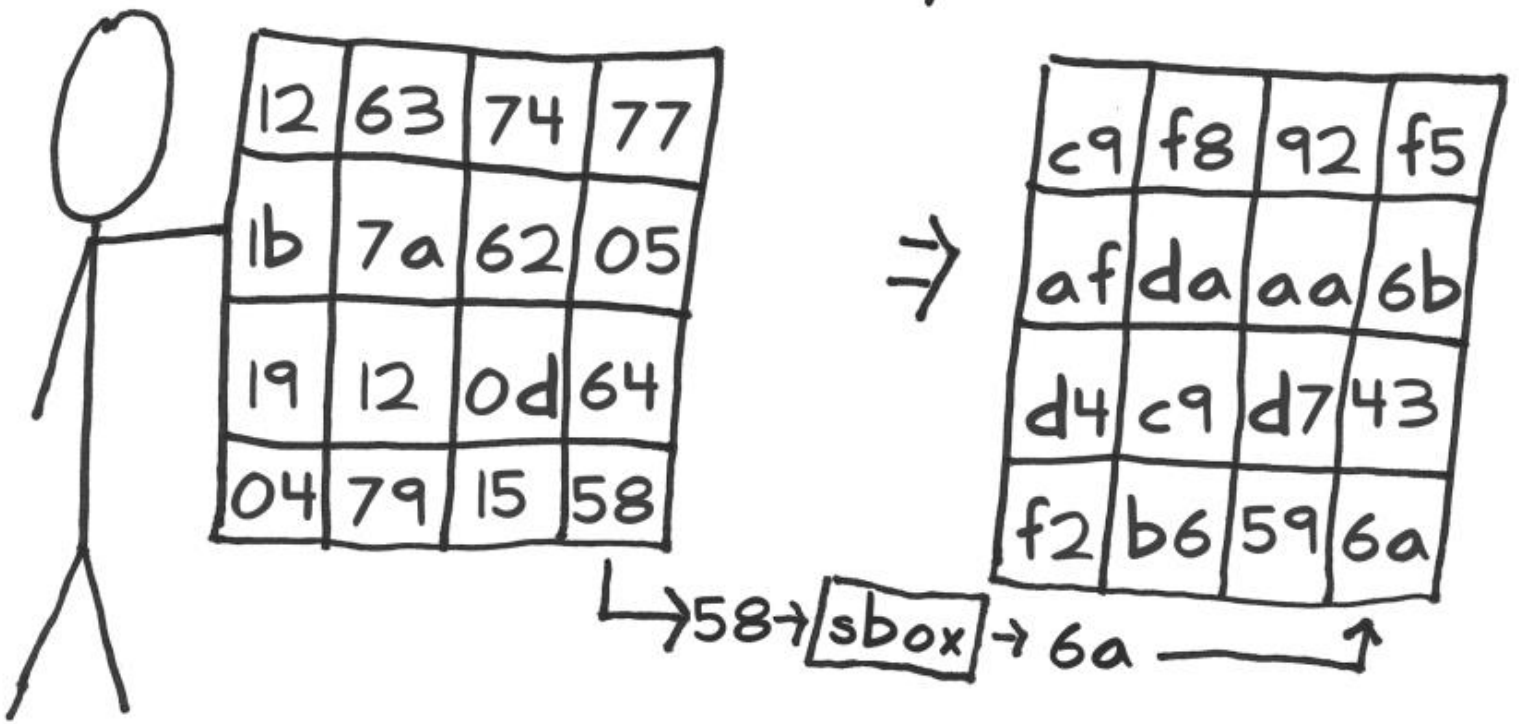
Round Repetitions	Key Size
9	128
11	192
13	256

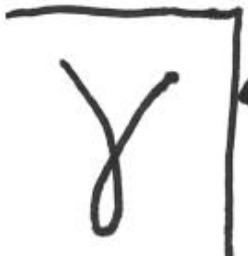
AES Round Diagram



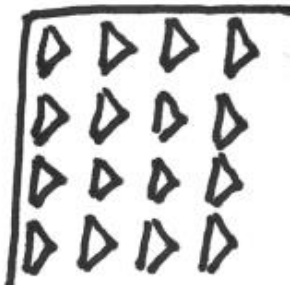
Applying Confusion: Substitute Bytes

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:





Denotes "confusion"

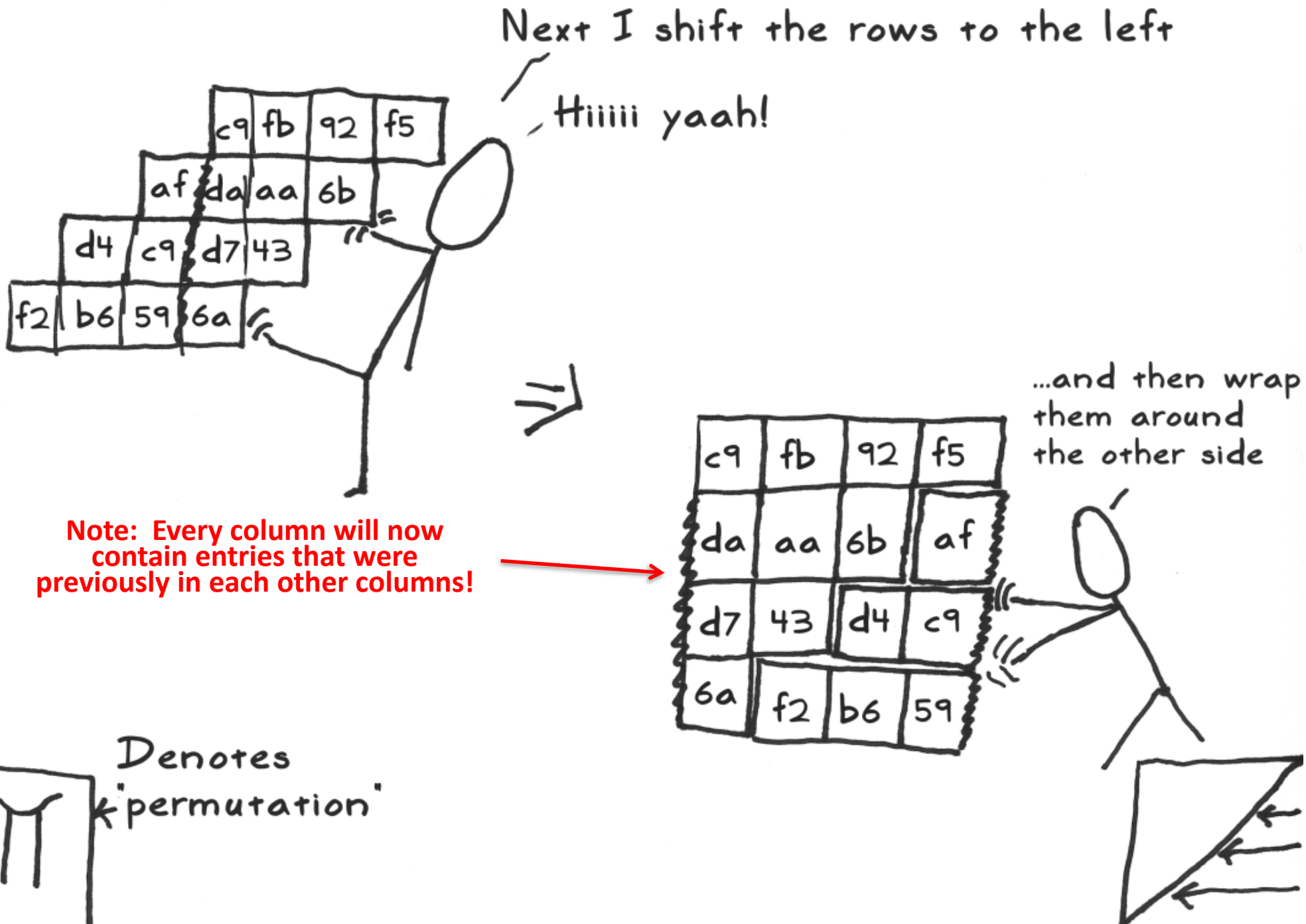


The S-Box is just a big lookup table!

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
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 1: Shift Rows

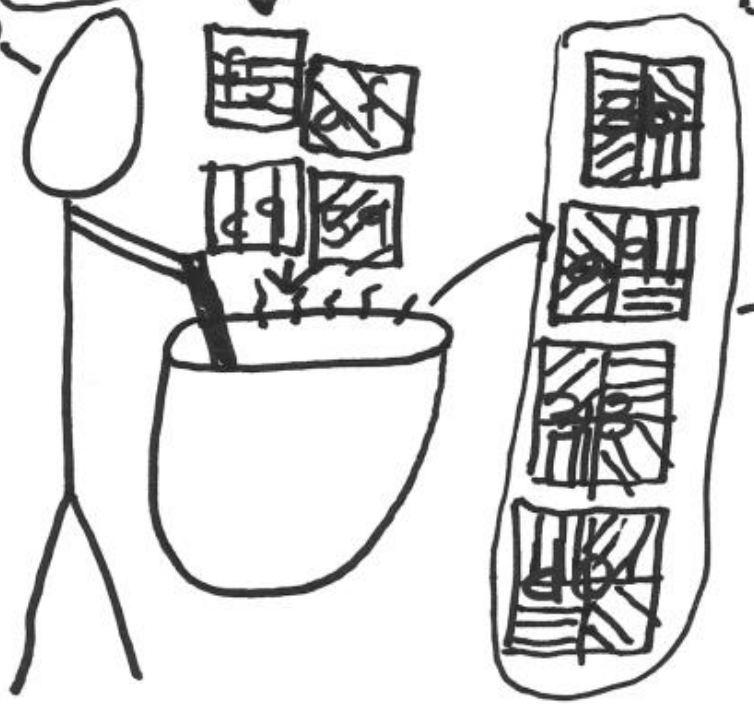


Note: Every column will now contain entries that were previously in each other columns!

Applying Diffusion, Part 2: Mix Columns

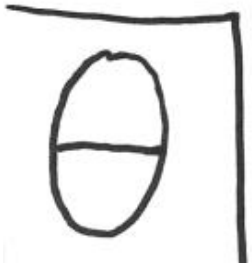
c9	fb	92	f5
da	aa	6b	af
d7	43	d4	c9
6a	f2	b6	59

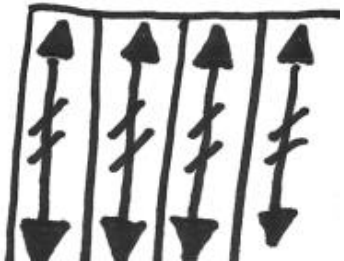
I take each column and mix up the bits in it.



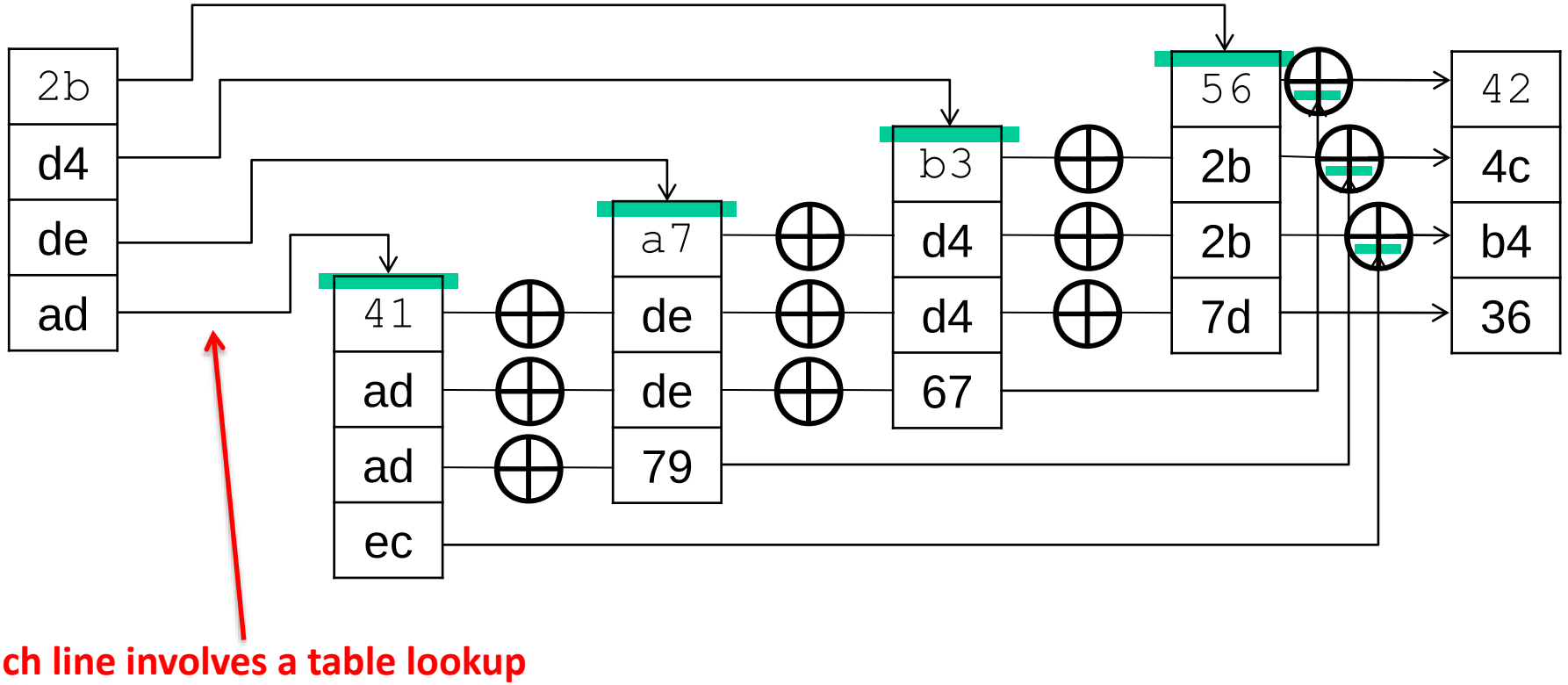
41	b9	e0	8b
6e	83	95	a9
18	da	8b	38
99	00	65	d0

Data begins to spill across rows

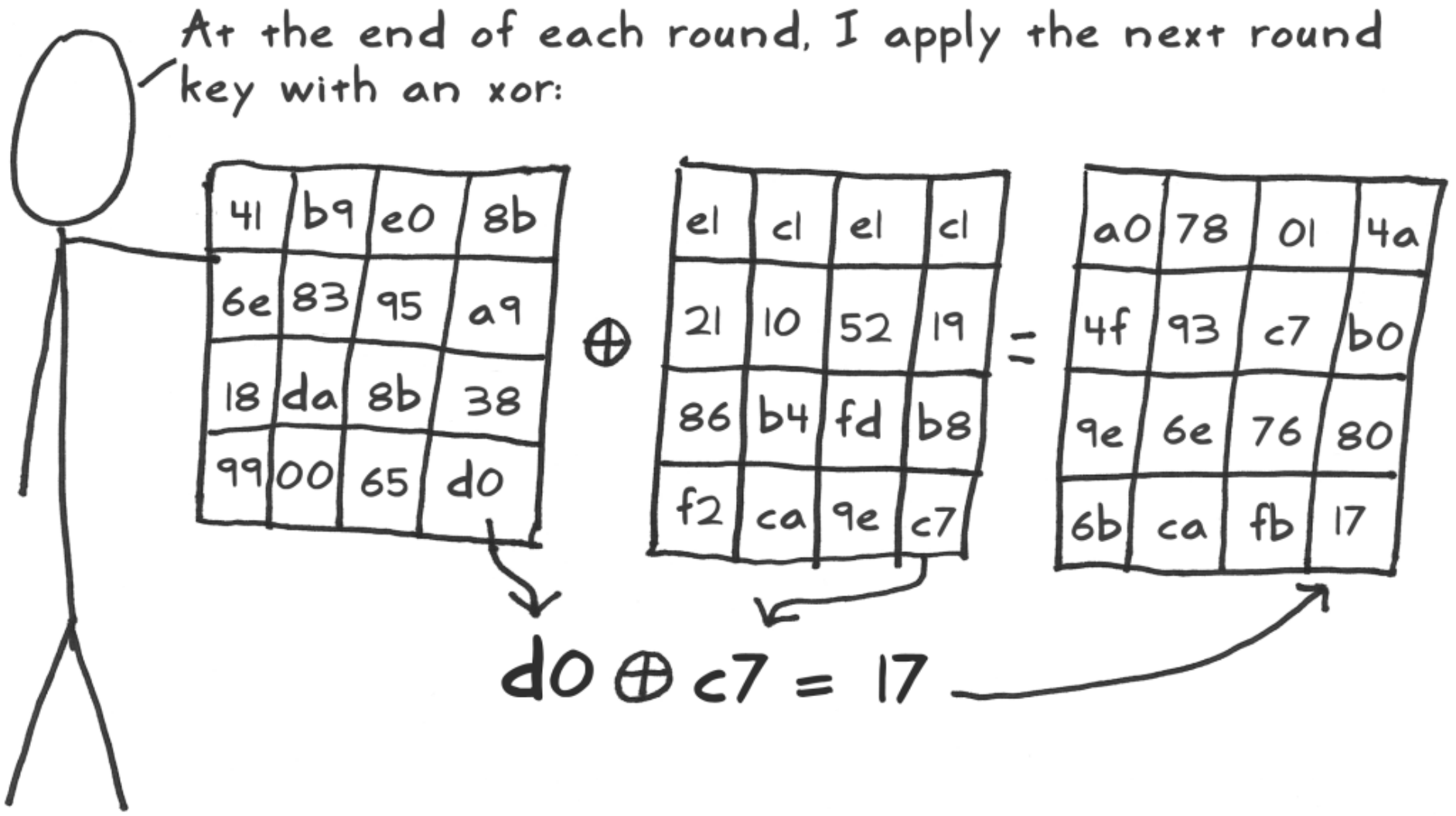




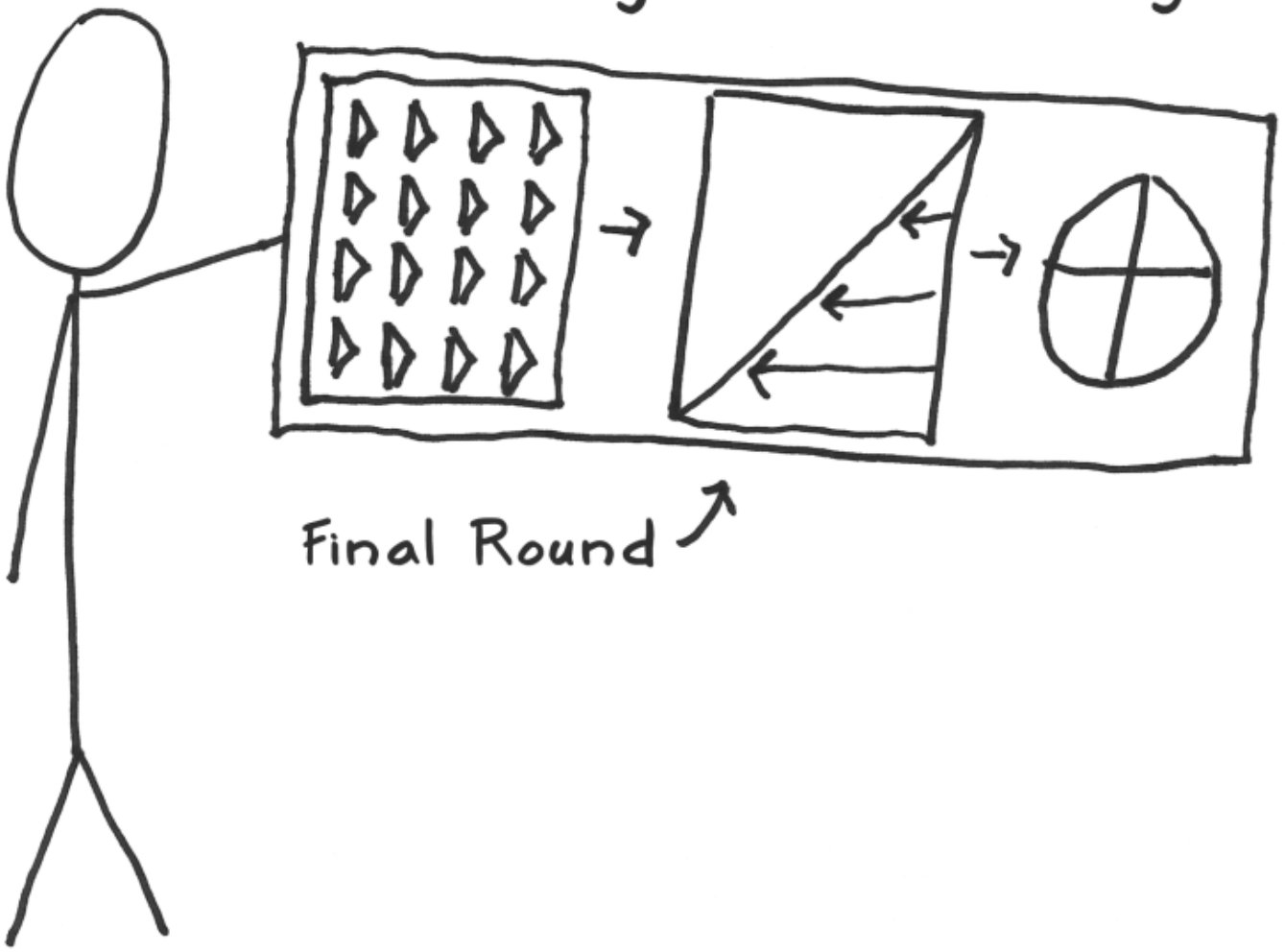
MixColumns is essentially table lookups and XORs



Applying Key Secrecy: Add Round Key

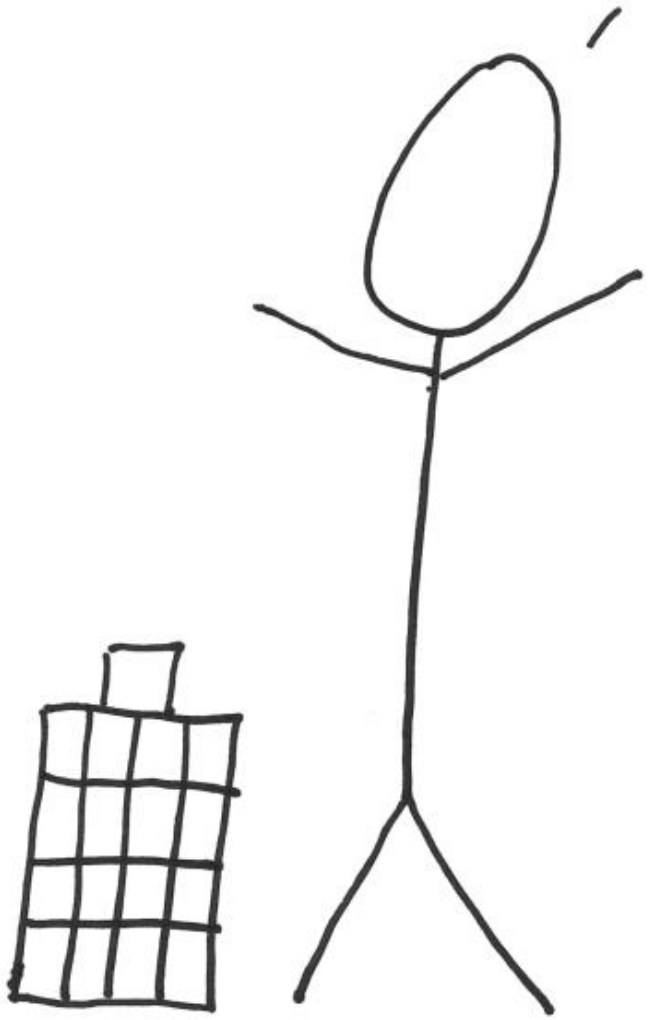


In the final round, I skip the "Mix Columns" step since it wouldn't increase security* and would just slow things down:

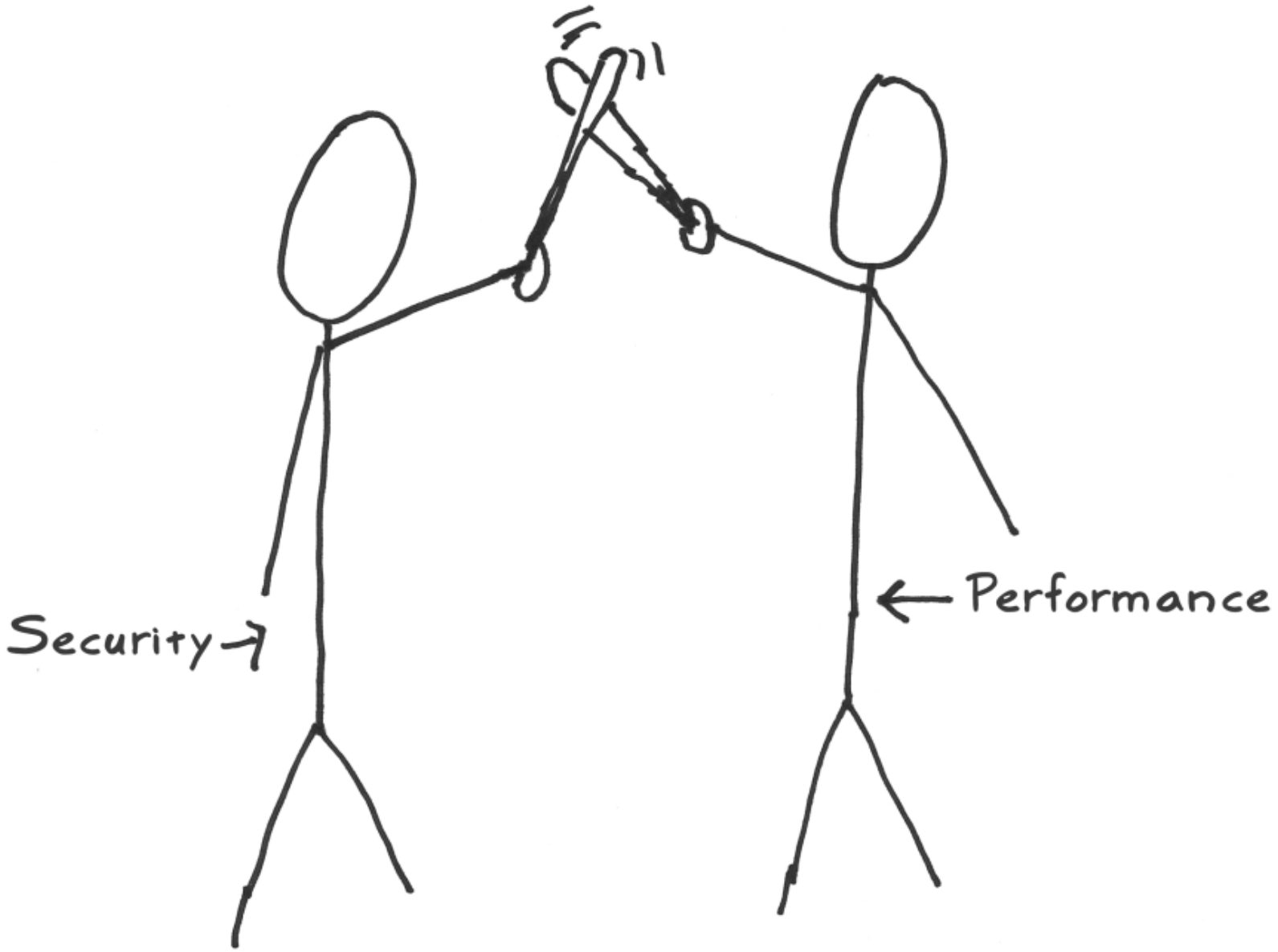


*The diffusion it would provide wouldn't go to the next round.

...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!

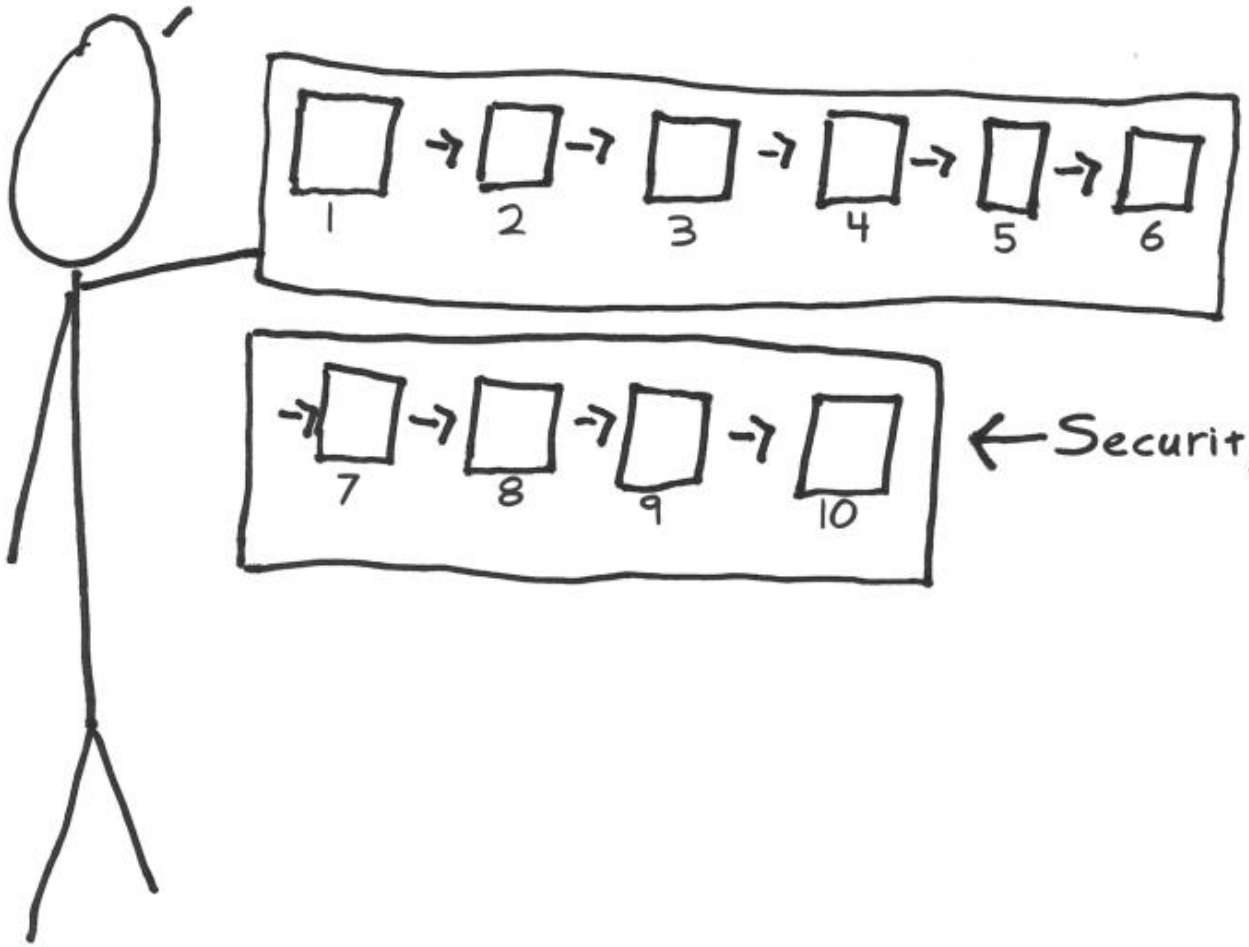


Determining the number of rounds always involves several tradeoffs.



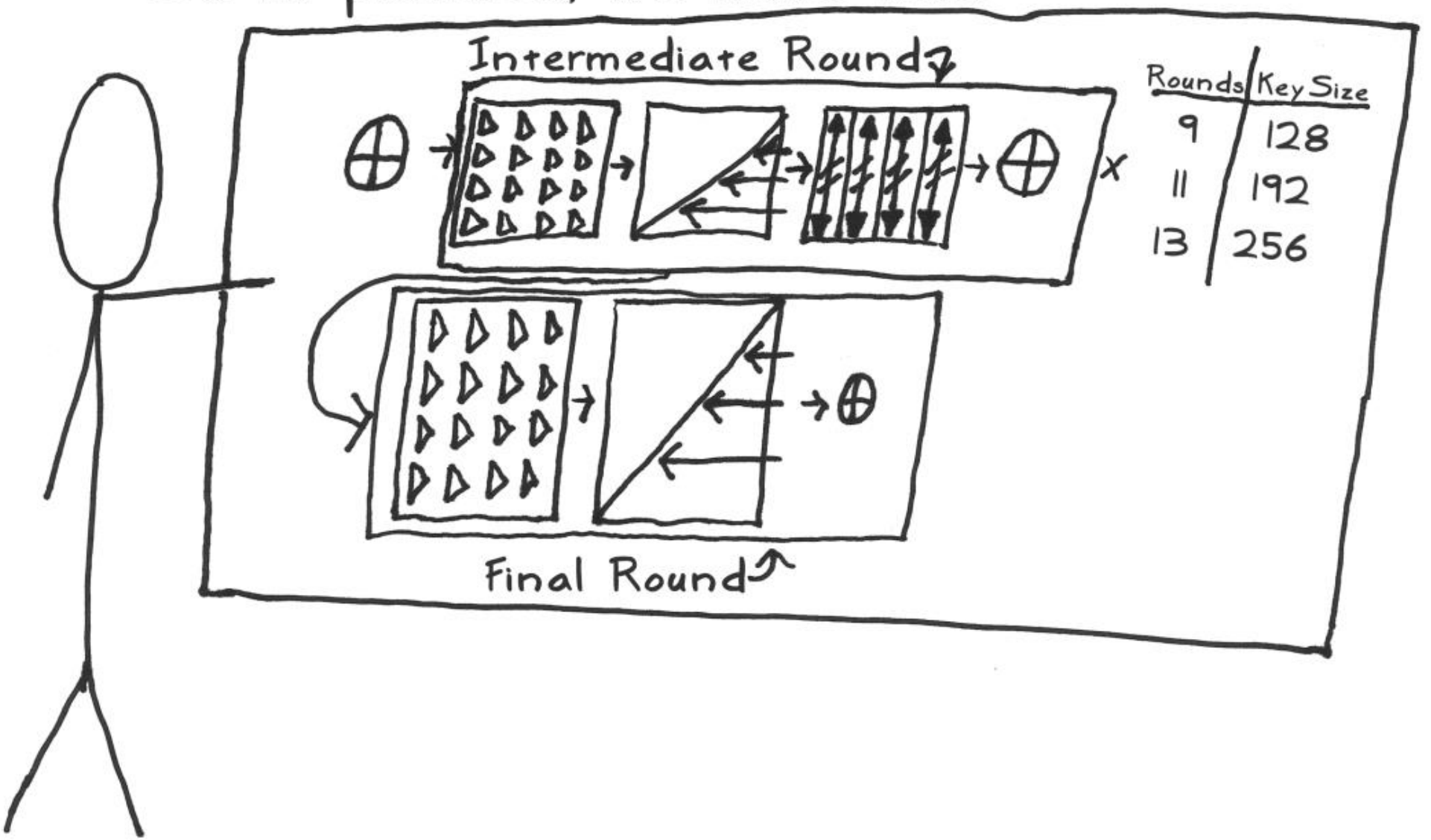
"Security always comes at a cost to performance" - Vincent Rijmen

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."

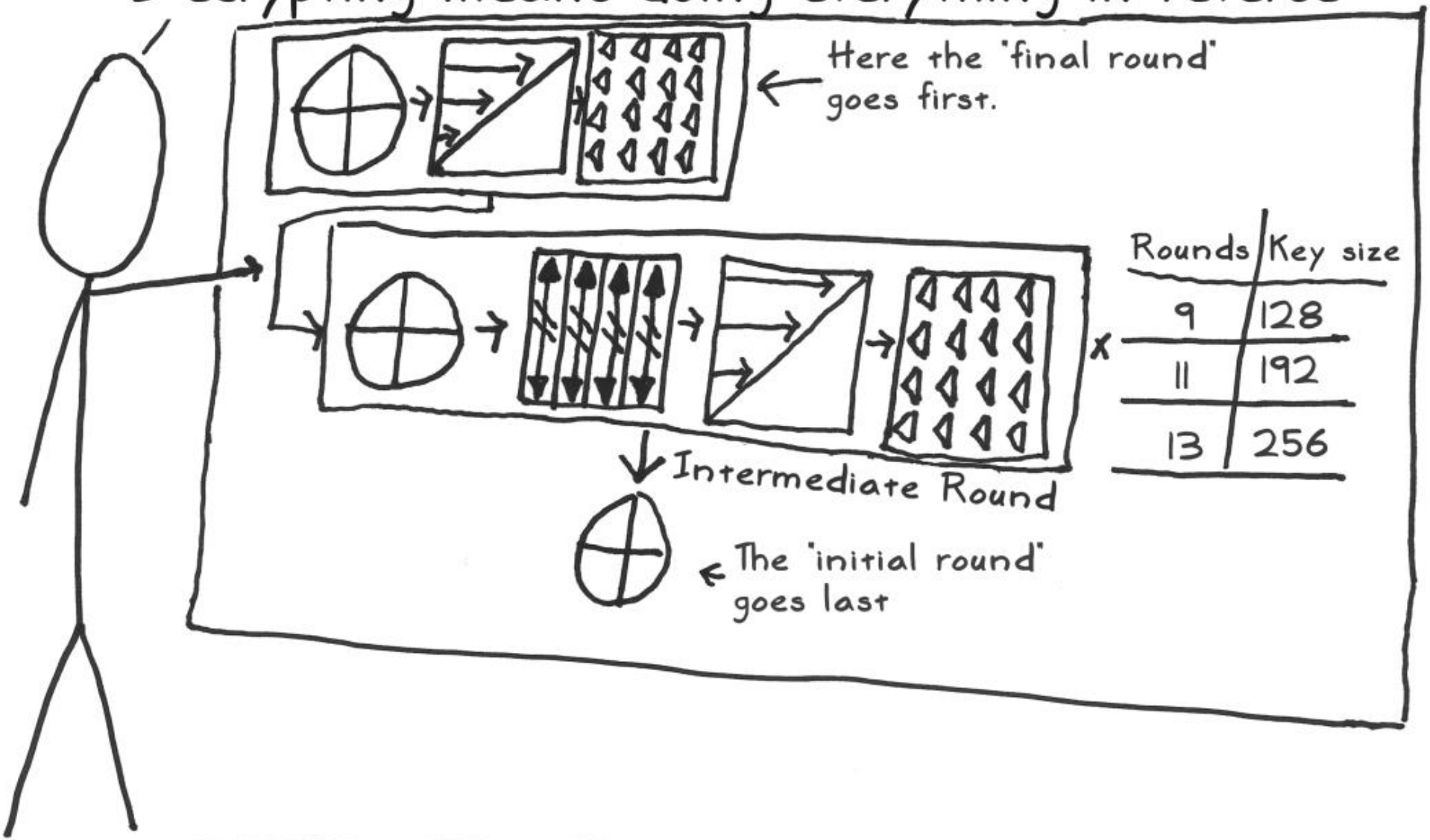


FIPS 197 Spec	
Key Size	Rounds
128	10
192	12
256	14

So in pictures, we have this:



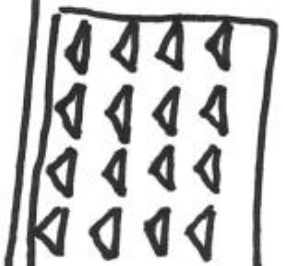
Decrypting means doing everything in reverse



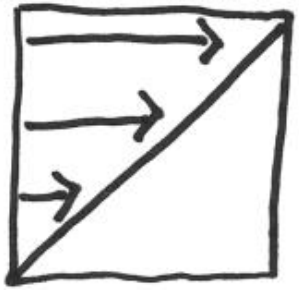
Rounds	Key size
9	128
11	192
13	256

Add Round Key Inverse

Inverse Substitute Bytes



Inverse Shift Rows



Inverse Mix Columns