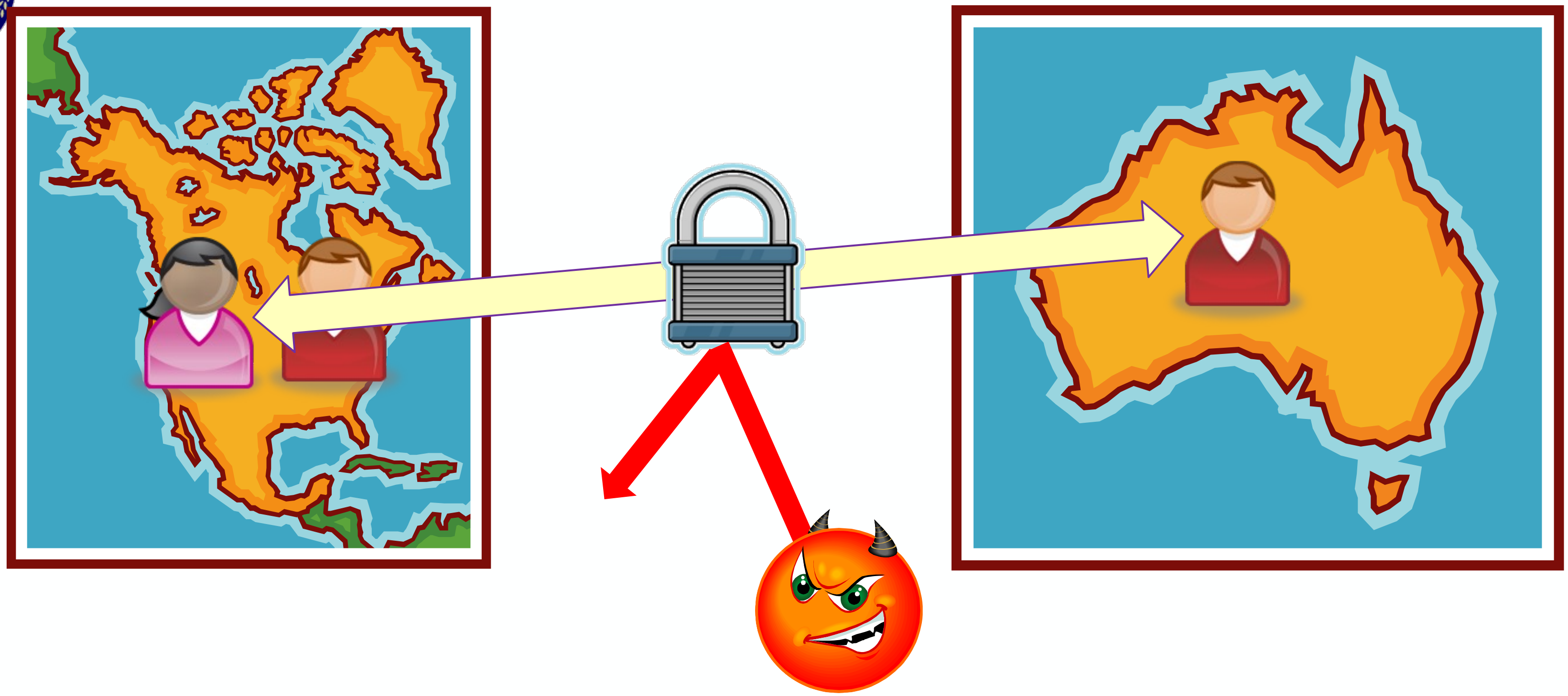
Applied Cryptography and Network Security

William Garrison
bill@cs.pitt.edu
6311 Sennott Square

Symmetric Key Cryptography

A Motivating Scenario



How can Alice and Bob communicate over an untrustworthy channel?

Need to ensure that:

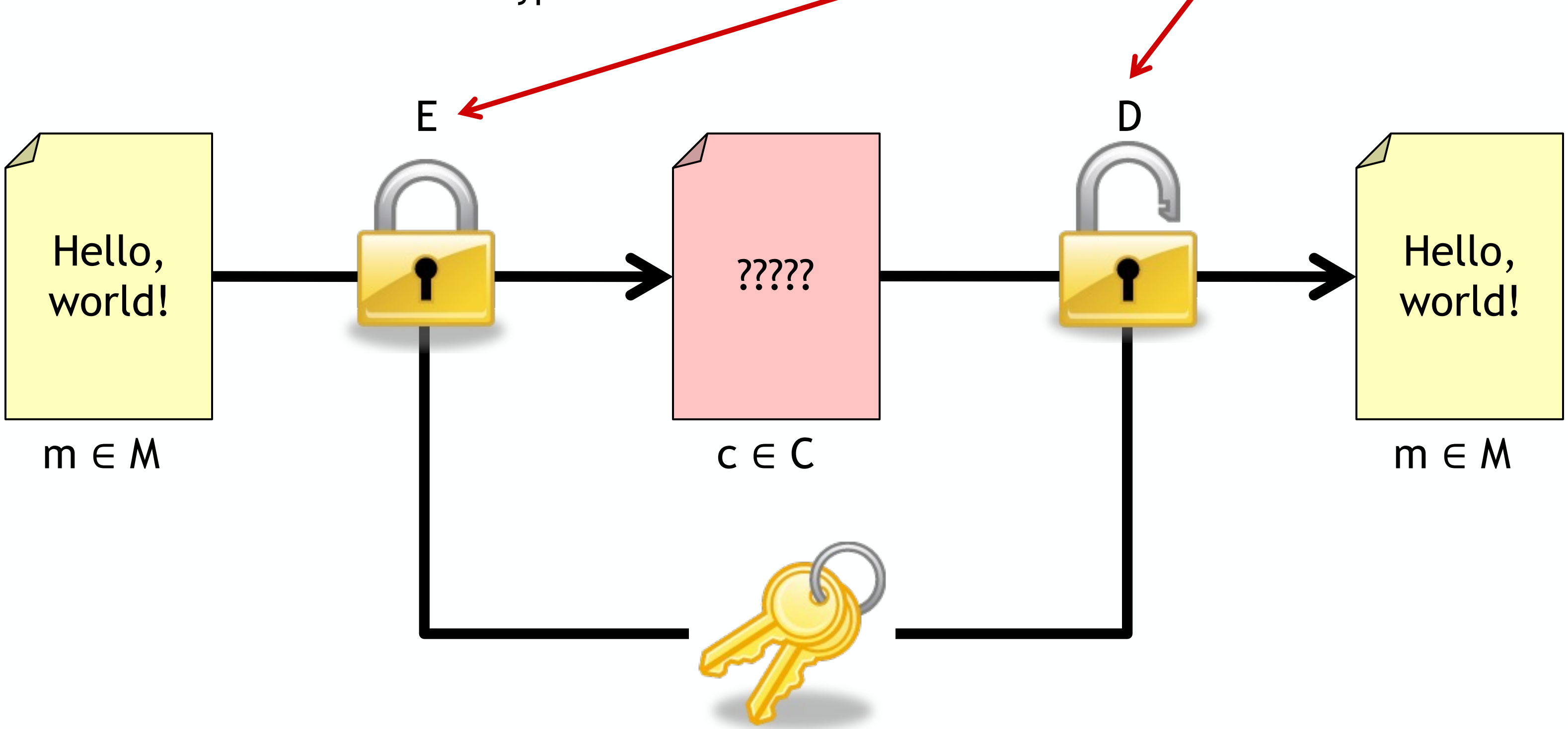
1. Their conversations remain secret (confidentiality)
2. Modifications to any data sent can be detected (integrity)

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

Today's focus is on symmetric key encryption



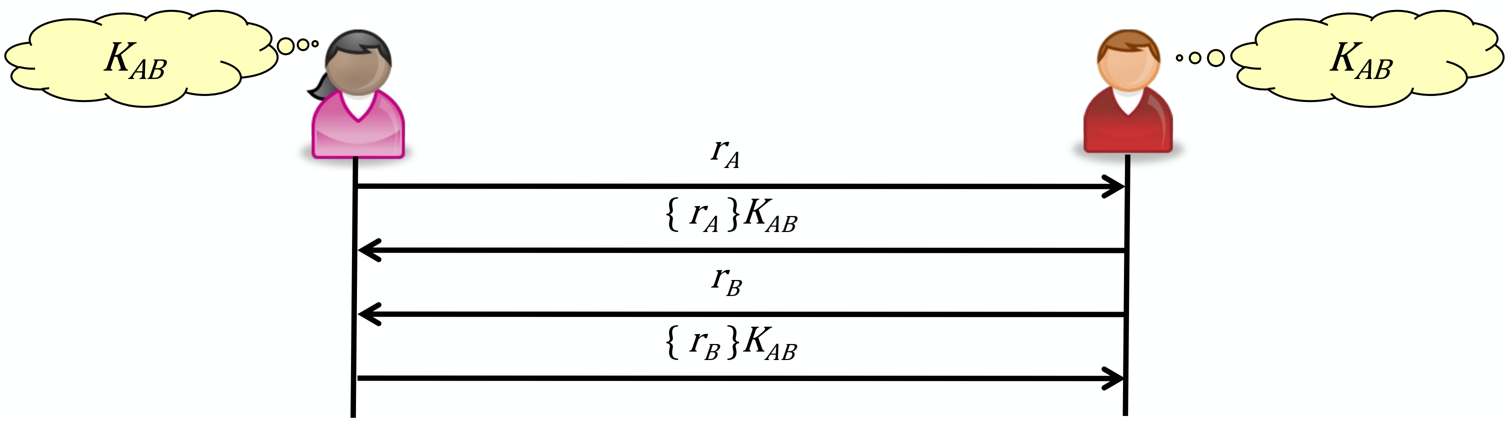
Why study symmetric key cryptography?

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 much detail soon…

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



The classical algorithms that we studied last time are examples of 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, $P(m) = P(m|c)$

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?

In modern cryptography, algorithms use a 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!

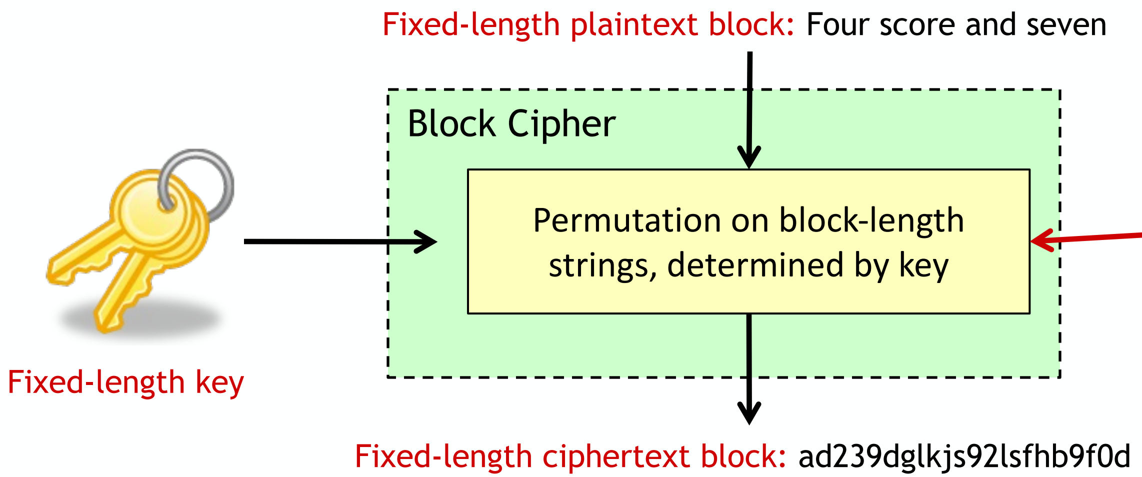
However, modern cryptographers have developed many algorithms that give good security using very small keys

We'll study two classes of symmetric key algorithms

- First: Block ciphers
  - DES, 3DES, AES, Blowfish, etc.
- Later: Stream ciphers
  - RC4, ChaCha, SEAL, etc.

BLOCK CIPHERS

Block ciphers are the most common symmetric cryptographic cipher



Designing good block ciphers is as much of an art as it is a science…

Block ciphers operate on fixed-length blocks of plaintext

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

Often, block ciphers apply several rounds of a simpler function

- Many 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

Block ciphers compared with substitution ciphers

Ideal Cipher Model: Each input is mapped to a random output

- Monoalphabetic substitution cipher: letter by letter
  - 26 options for each of 26 input letters
  - Full mapping can be written down in dozens of bits
- Block cipher with (say) 64-bit blocks
  - 2^{64} options for each of 2^{64} input blocks
  - Full mapping would require $\sim 2^{70}$ bits (a zettabyte) to store

The ideal cipher model is infeasible

- Provably incompatible with being efficiently computable
- Can be approximated: Pseudorandom permutation (PRP)
  - Outputs should look random to someone without the key

Structure of a practical block cipher

Similar components to classical ciphers with more repetition and complexity

- Usually in multiple rounds: mini-ciphers that are not necessarily secure individually

Each round transformation uses a per-round key

- Key expansion: Deriving per-round keys from main key
- This key schedule is often cached if multiple blocks are to be encrypted

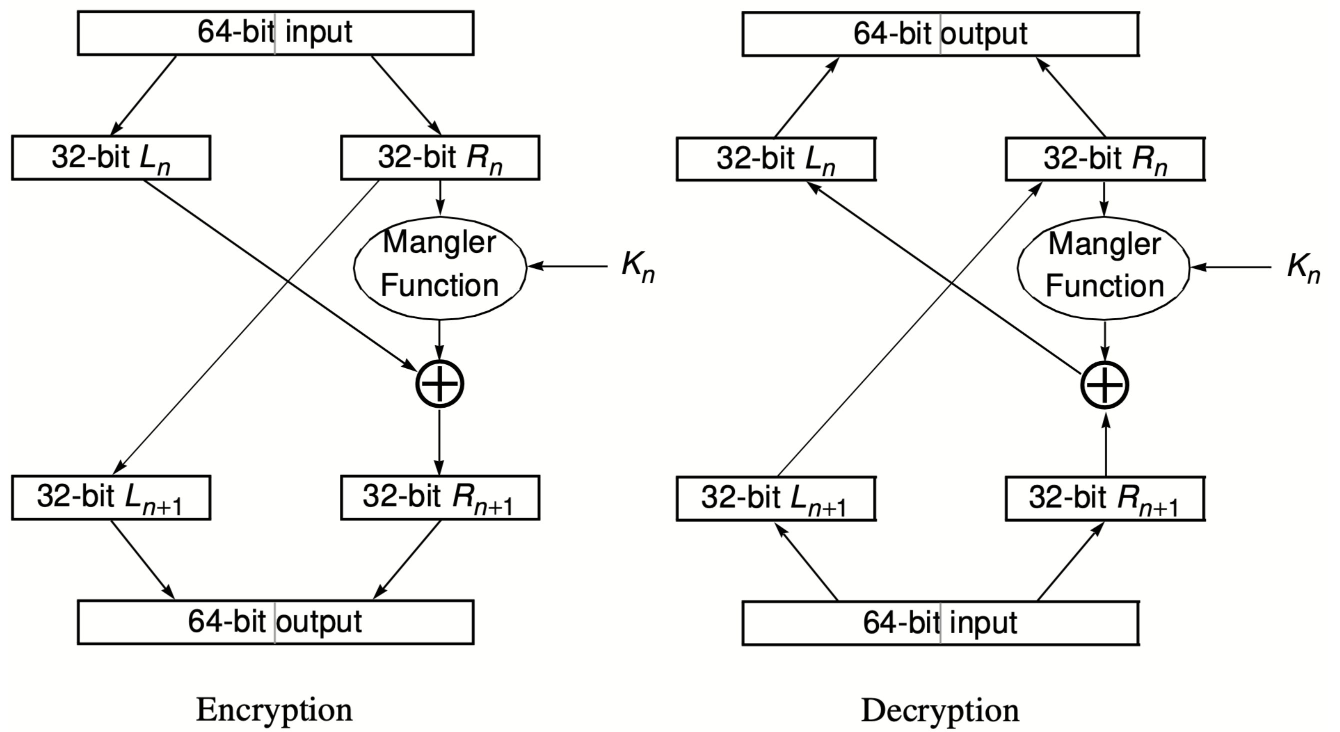
Each round is composed of replacements and shufflings

- More complex versions of what we saw in historical substitution and transposition ciphers (respectively)
- S(ubstitution)-boxes: Replace each possible input value with an output value
- P(ermutation)-boxes: Rearrange the positions of various bits
- The per-round key alters the specific details of these substitutions and permutations

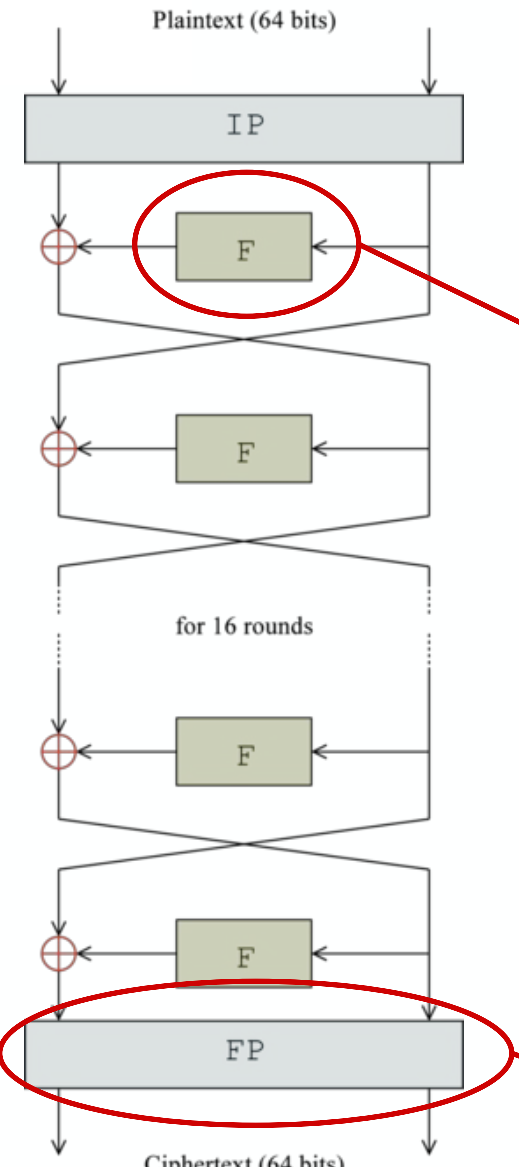
Feistel networks: Making block ciphers reversible

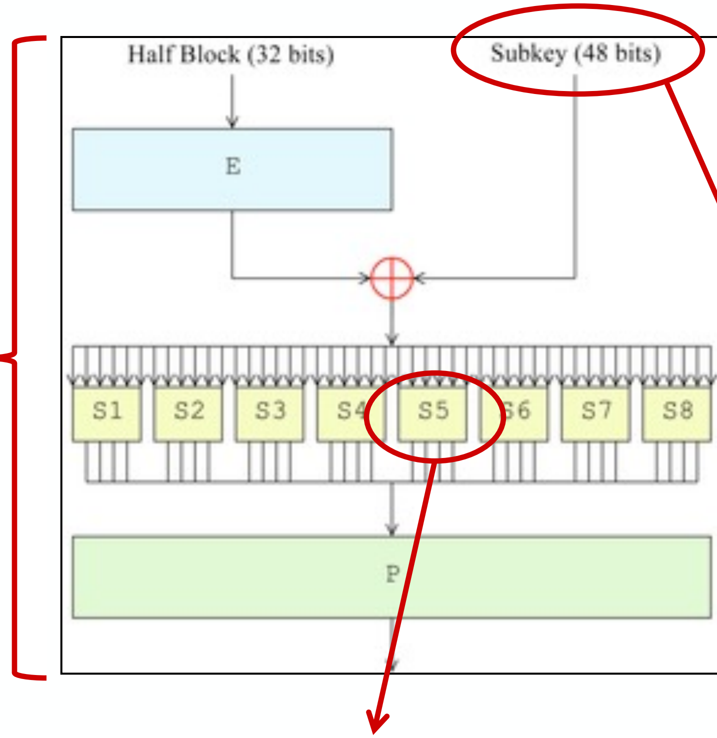
Horst Feistel developed the Feistel cipher structure

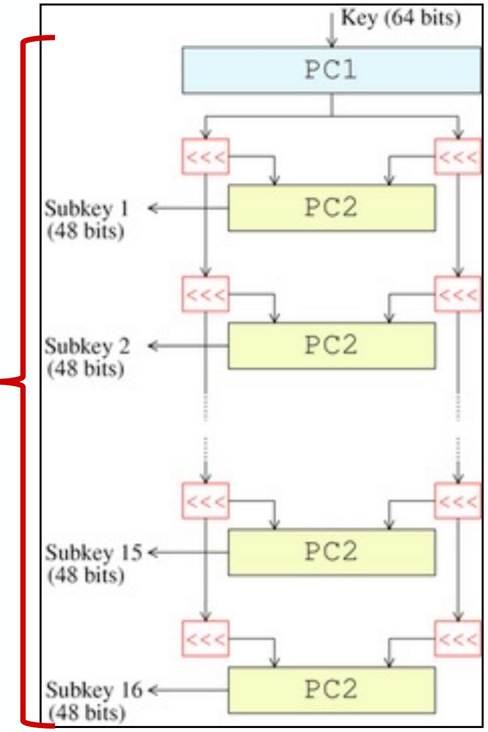
- ~1971 at IBM, along with other key ideas for block ciphers
- Key idea: Split the block in half, "mangle" and swap
  - (This is the structure of each round)



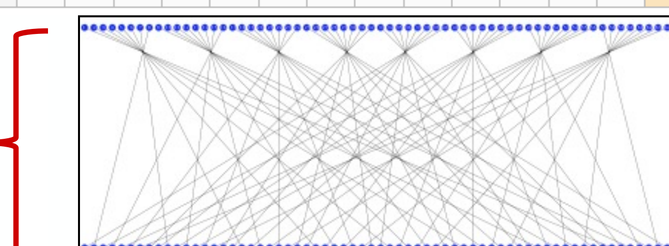
Example: DES







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

Some of the history behind DES

NIST (then NBS) standardized DES in 1976

- Based on Lucifer cipher developed at IBM in early 1970s
- Substitution boxes changed from IBM's design
- (Effective) key length reduced from 64 bits to 56 bits

Suspicion was immediately raised about the changes

- Changing S-boxes without explanation caused concern about backdoors
  - Might there be attacks against specific combinations?
  - 1990 differential cryptanalysis discovered, DES was strengthened
- Decreased key length weakens the cipher against brute-force attacks
  - 1991 Quisquater and Desmendt discussed the possibility of a "Chinese lottery"
  - 1993 Weiner proposed $1M machine, 7 hours
  - 1997 RSA Security held contest, distributed DESCHALL broke DES
  - 1998 EFF built Deep Crack, $250,000, about 2 day

Bandages that kept DES in use longer despite brute-force weakness

Triple-DES (3DES) was proposed to overcome DES's weakness to brute-force

- To encrypt, use DES three times
- Initially, 2 keys (K_1 then K_2 then K_1 again)
  - Later, 3 keys (K_1 then K_2 then K_3)
- EDE: Use encrypt mode, then decrypt mode, then encrypt mode
  - One reason: hardware implementations could be (wastefully) backward compatible by setting $K_1 = K_2 = K_3$
- Less security than expected from the long runtime and combined key length

This was clearly not a long-term solution, and development started on a replacement

- 1997 NIST design competition, Rijndael $\rightarrow$ AES
- Goal: At least as strong as DES, but more efficient and flexible

Between DES and AES came Blowfish, by Schneier et al.

1993, no patents, 64-bit block size, huge subkeys, slow to switch keys

Still no serious cryptographic breaks

- (Some weak keys)
- Twofish is a more modern version

Some interesting and unique properties

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

What is the key schedule for Blowfish?

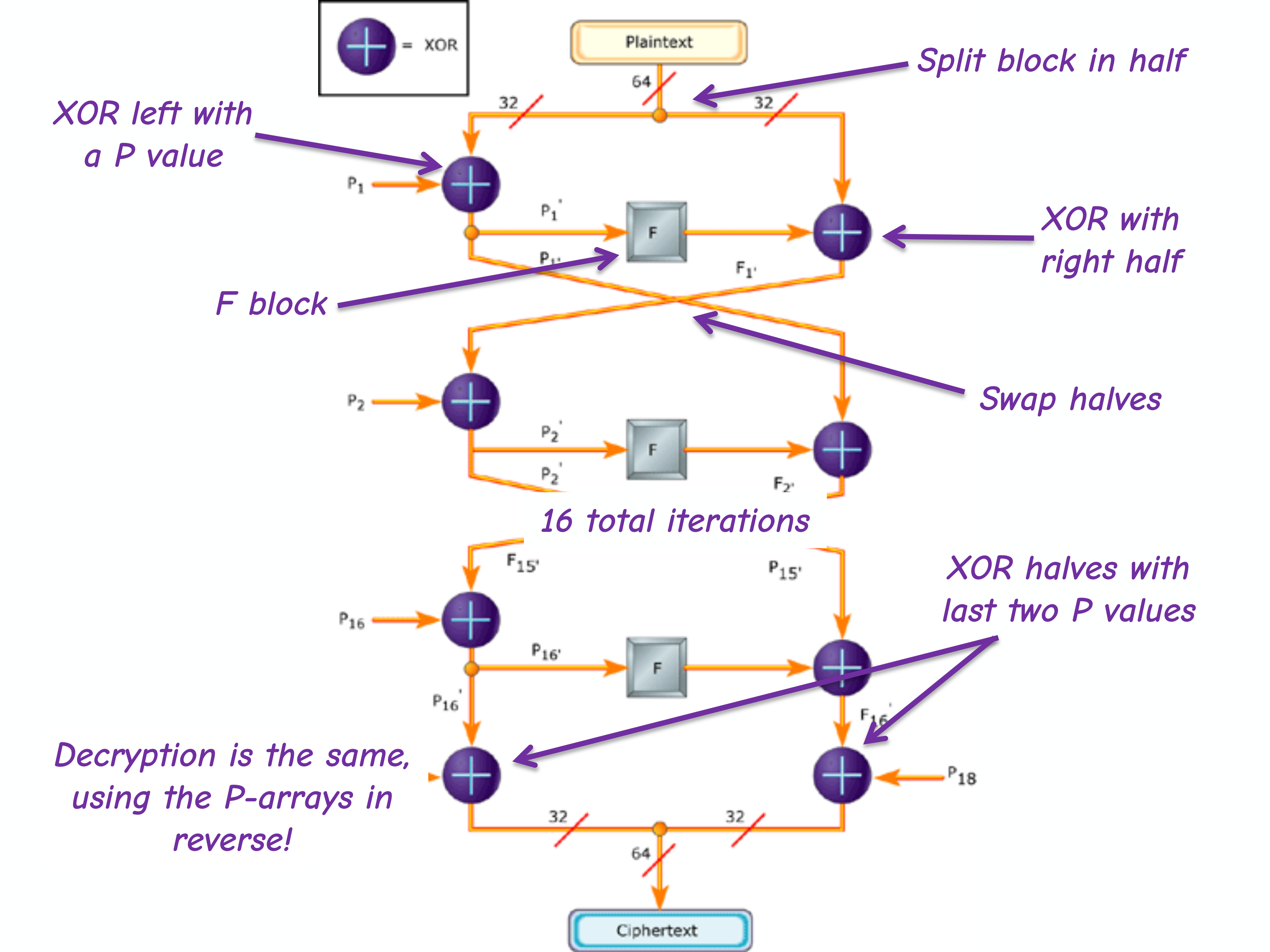
(Recall: Subkeys are data computed from the key, used in encryption)

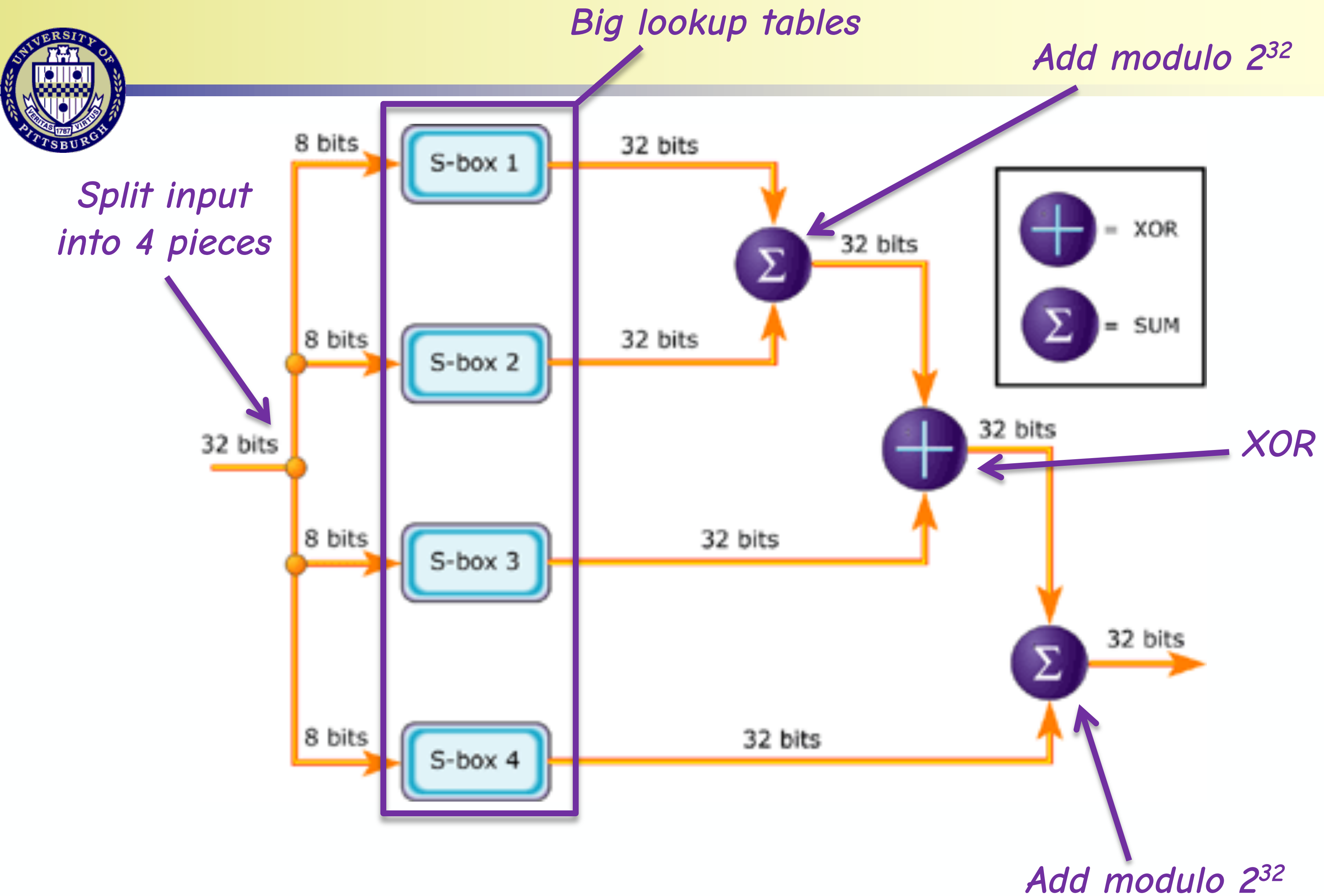
P-array
- 18 subkeys
- 32 bits each
- $P_1, P_2, \ldots, P_{18}$

72 bytes

S-boxes
- 4 lookup tables
- 8 bits → 32 bits
- $S_{1,0}, S_{1,1}, \ldots, S_{1,255};$
  $S_{2,0}, S_{2,1}, \ldots, S_{2,255};$
  $S_{3,0}, S_{3,1}, \ldots, S_{3,255};$
  $S_{4,0}, S_{4,1}, \ldots, S_{4,255};$

4096 bytes


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

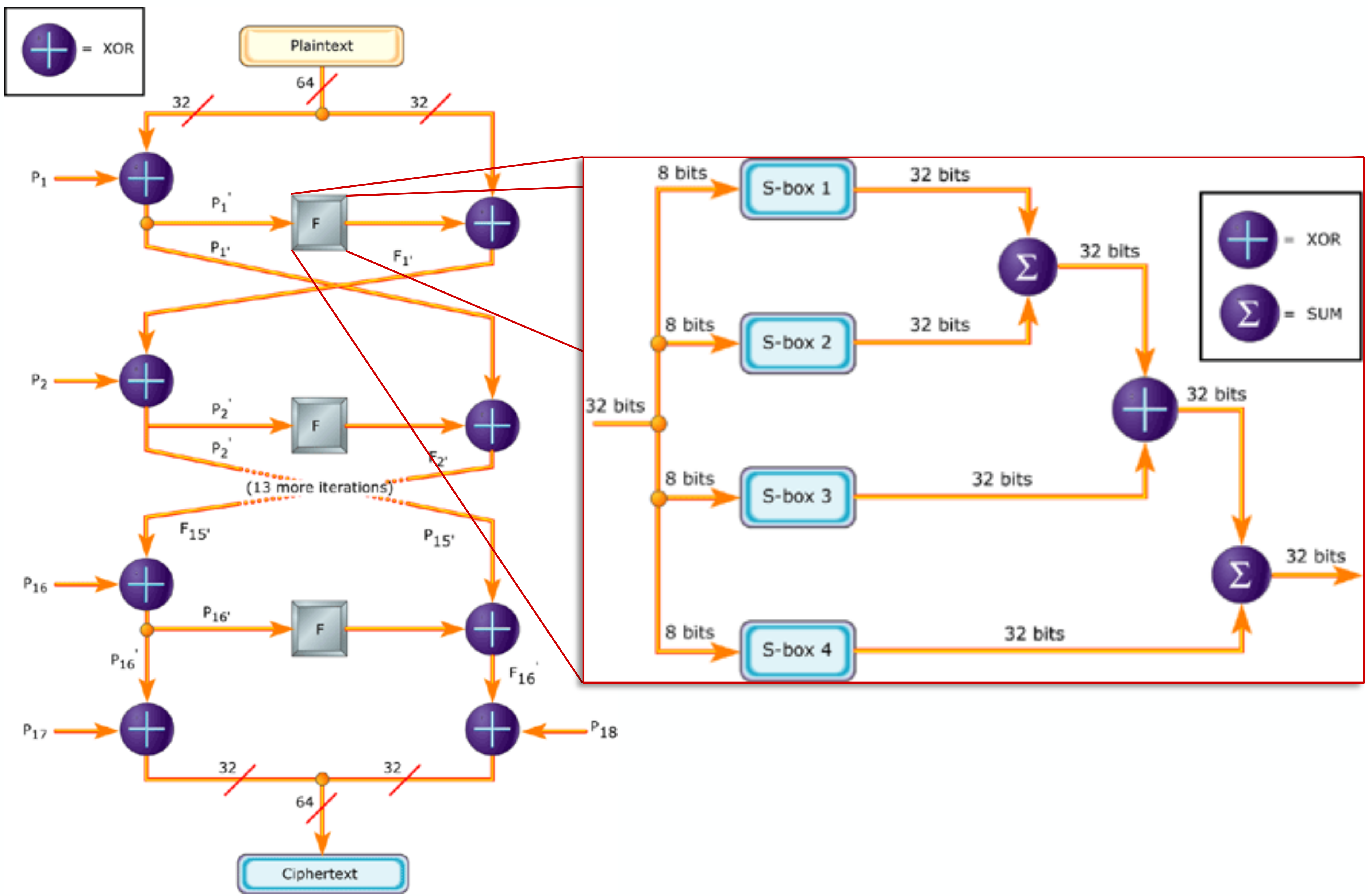

Big lookup tables
Add modulo 2^{32}
Split input into 4 pieces
32 bits
8 bits
8 bits
8 bits
8 bits
S-box 1
S-box 2
S-box 3
S-box 4
32 bits
32 bits
32 bits
32 bits
32 bits
32 bits
32 bits
= XOR
= SUM
XOR
Add modulo 2^{32}

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
   - $P_1 = P_1 \oplus$ first 32 bits of key
   - $P_2 = P_2 \oplus$ 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 full encryptions!!



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

A few questions to think about…

Why initialize the constants with the digits of pi?

Which step in Blowfish is very inefficient?

- In what way is that a good thing?

Unique components of AES

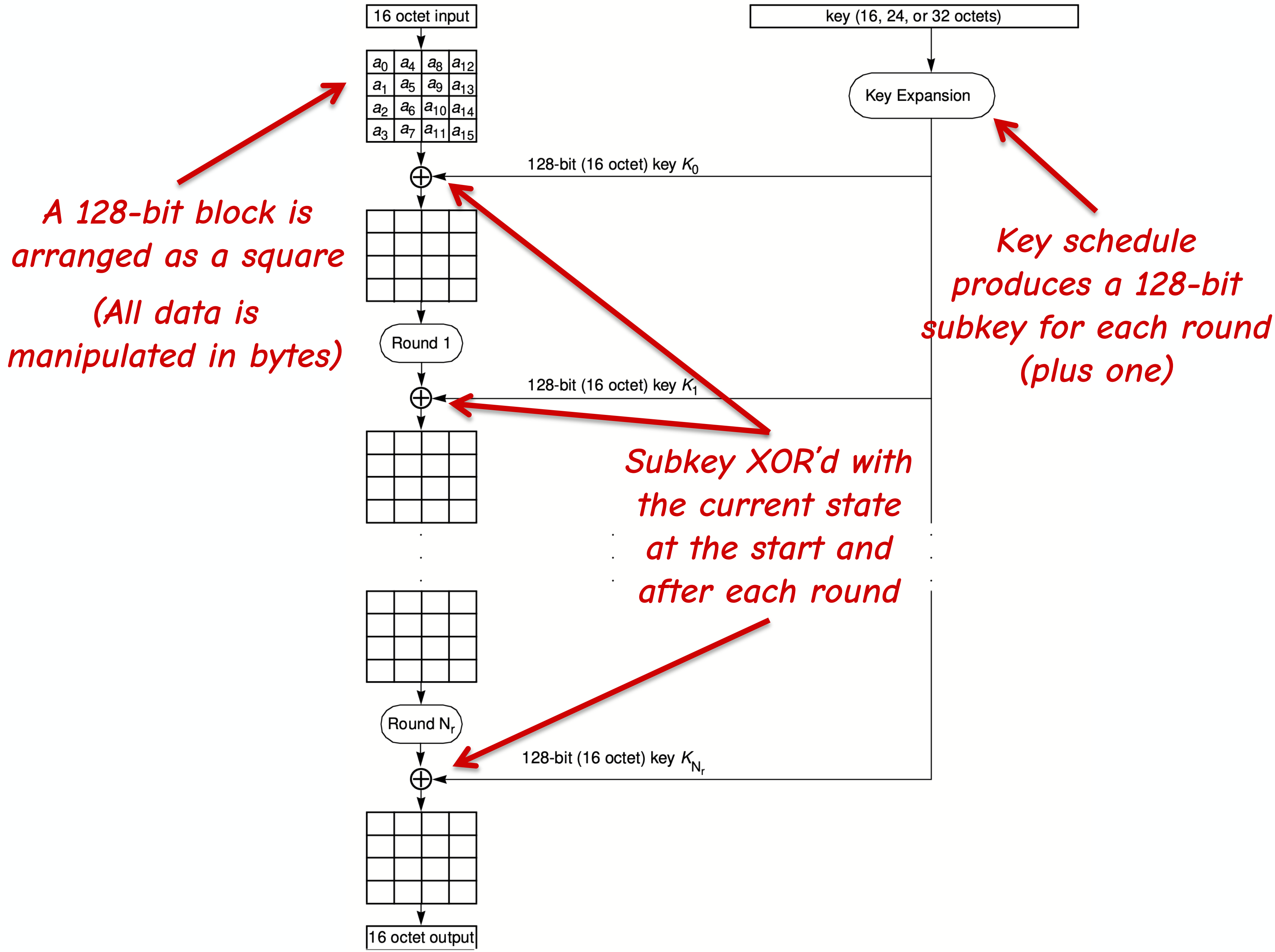
We've seen that most block ciphers have linear and non-linear steps in each round

- Linear: preserved by XOR, i.e., $F(a \oplus b) = F(a) \oplus F(b)$
  - In DES, the P-box permutations
- Non-linear: not preserved by XOR, often a series of S-boxes
  - These steps prevent the use of linear equations to cryptanalyze

In AES, the non-linear 8-bit S-box is based on multiplicative inverse over GF(2^8)

- Good non-linearity properties, compact description
- "Nothing up my sleeve"
- Otherwise, arbitrary; many other options would work well

Note that AES is not a Feistel network; all steps must be 1-to-1, but all bits can be mangled in each round (vs. half)



16 octet input
key (16, 24, or 32 octets)
Key Expansion
a_0 a_4 a_8 a_{12}
a_1 a_5 a_9 a_{13}
a_2 a_6 a_{10} a_{14}
a_3 a_7 a_{11} a_{15}
128-bit (16 octet) key K_0
Round 1
128-bit (16 octet) key K_1
Round N_r
128-bit (16 octet) key K_{N_r}
16 octet output
A 128-bit block is arranged as a square
(All data is manipulated in bytes)
Key schedule produces a 128-bit subkey for each round (plus one)
Subkey XOR'd with the current state at the start and after each round

The non-linear layer in AES: SubBytes

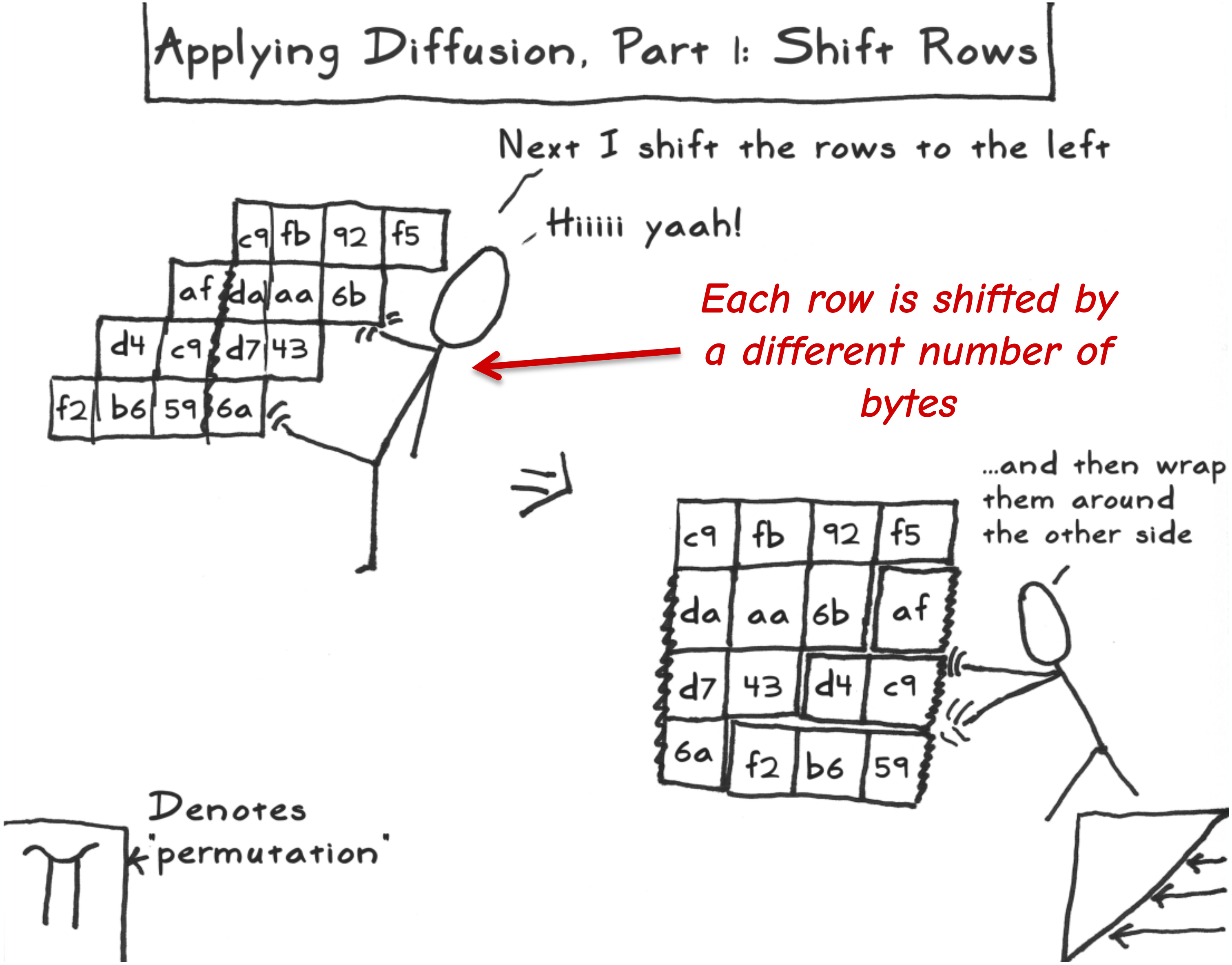
SubBytes implements the S-box in AES

- Each of the 16 bytes in the square is replaced with another
- Substitution was derived from inverses in Galois Fields
  - … but can be implemented as a hard-coded lookup table
- The same lookup table is used for all bytes

AES S-box

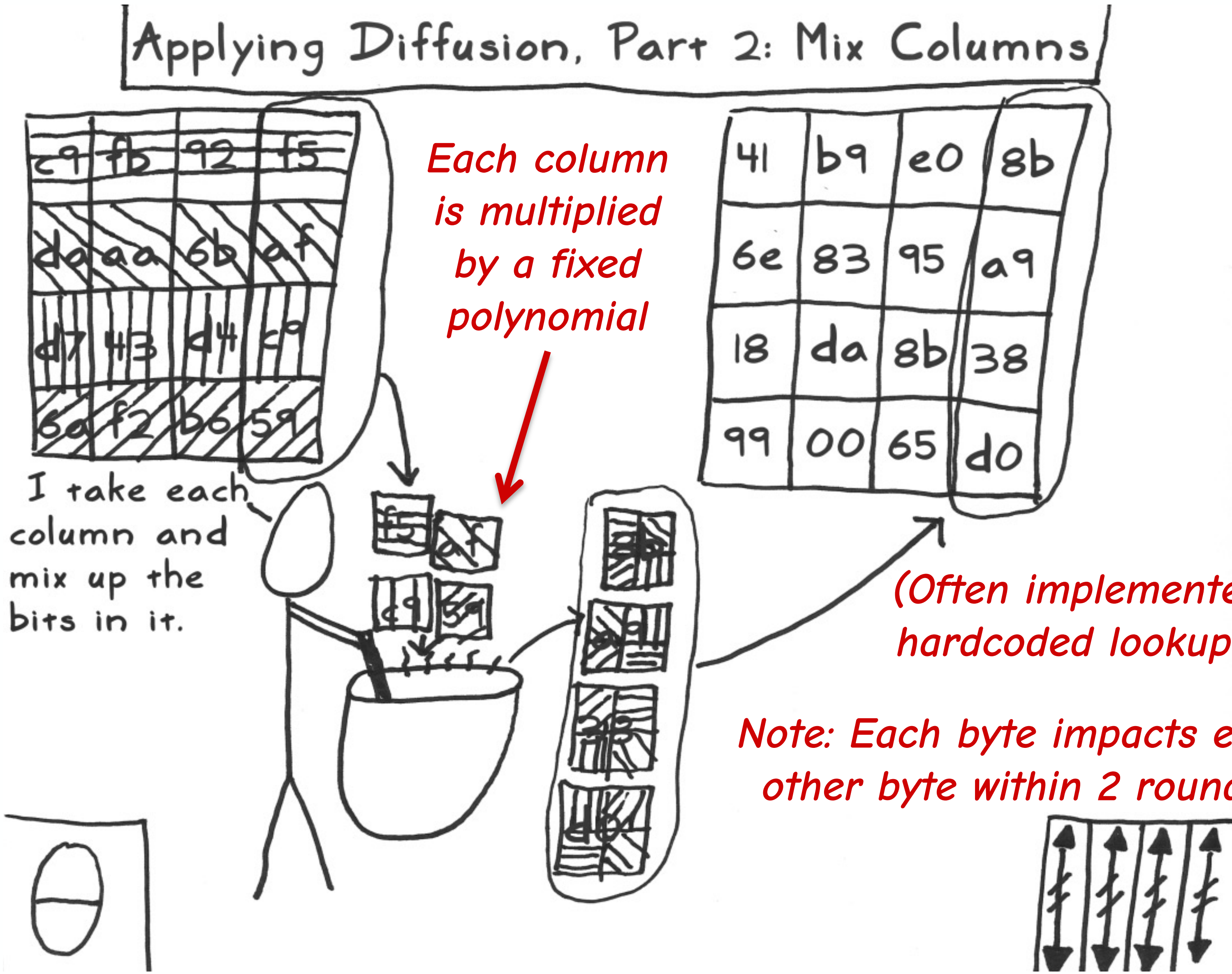
	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84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

The linear layer in AES: ShiftRows and MixColumns



Each row is shifted by a different number of bytes

The linear layer in AES: ShiftRows and MixColumns



Each column is multiplied by a fixed polynomial

(Often implemented as a hardcoded lookup table)

Note: Each byte impacts each other byte within 2 rounds!

Back to the key schedule: How are subkeys calculated?

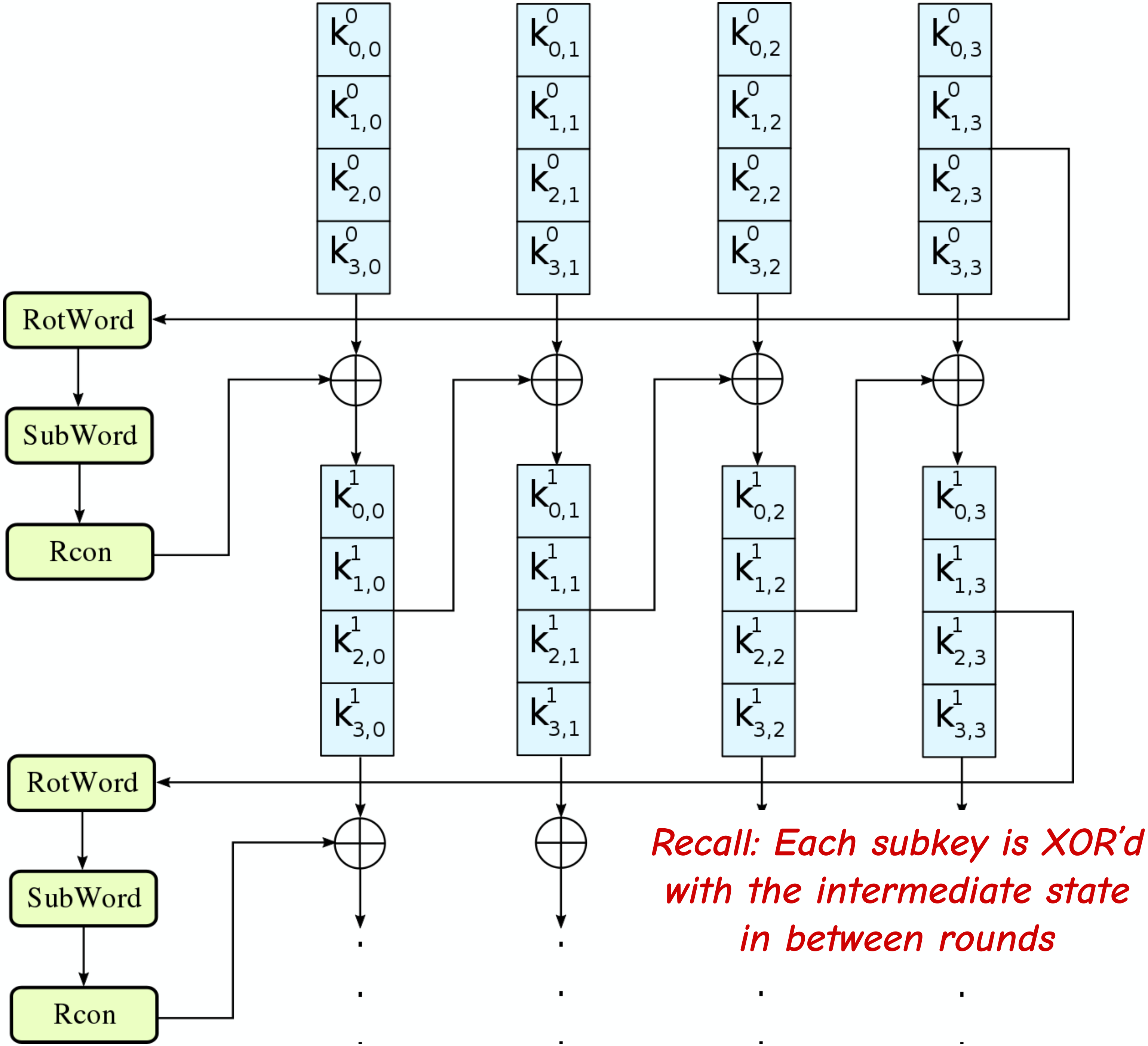
(We will focus on 128-bit keys, though others are related)

Initial key populates a square, column-wise

- Same as we did with a data block

Later subkeys are generated from earlier ones

- RotWord: In the last column, rotate the top byte to the bottom
- SubWord: Use the S-box to substitute each byte in this column
- Rcon: XOR this column with a round-specific constant
- This column becomes the first in the new round key
- Other columns are computed by XORing the old value with the column to the left in the new value


RotWord
SubWord
Rcon
RotWord
SubWord
Rcon
Recall: Each subkey is XOR'd
with the intermediate state
in between rounds

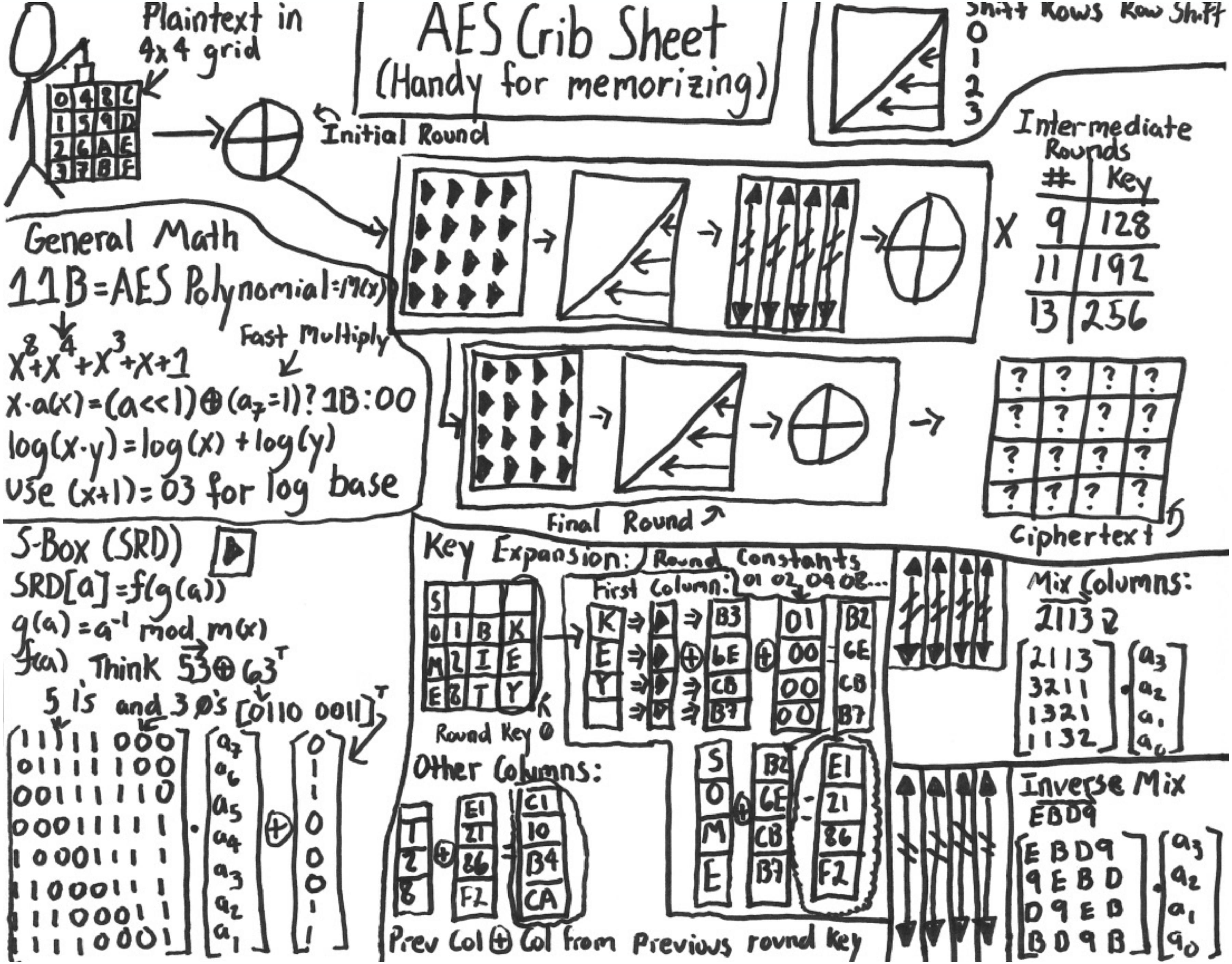
Implementation details in AES

As mentioned, AES is not a Feistel network!

- Feistel networks are easily reversible, but can only mangle half the data in each round
- Instead, each step was carefully designed to be independently reversible
- Decryption executes the inverse of each step, in reverse order

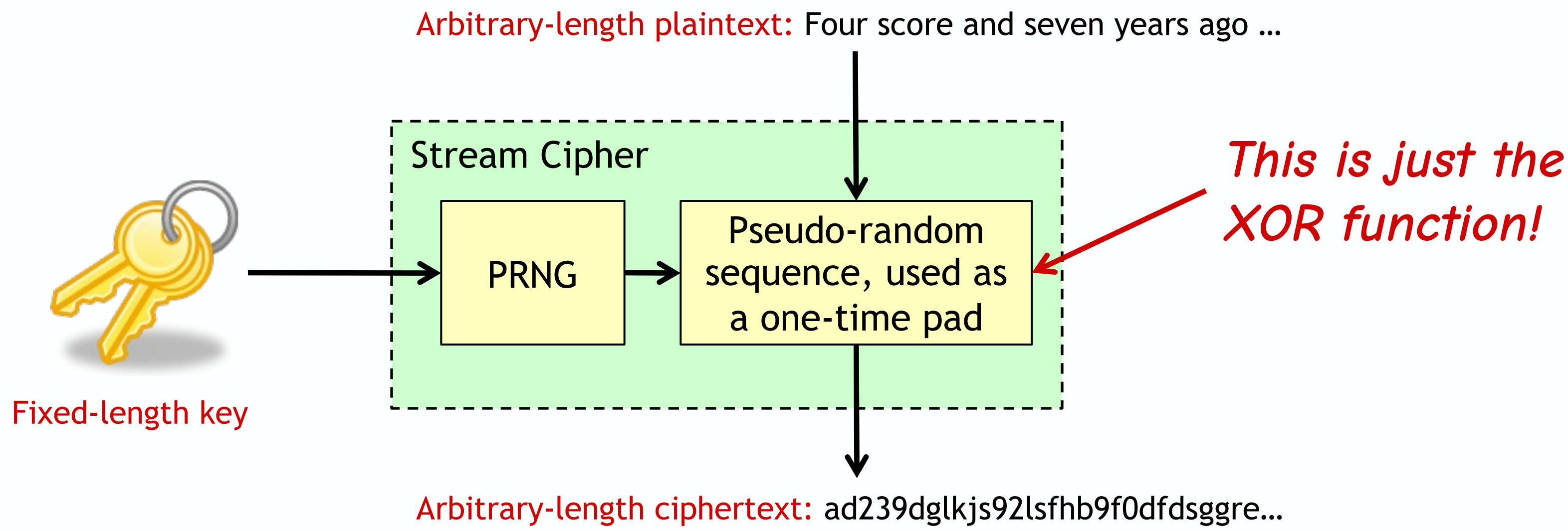
Note that many steps use “hard” math conceptually but can be implemented with much simpler operations

- e.g., XOR, table lookup
- Today, AES is (almost) always hardware-accelerated
  - CPU vendors add special AES-specific instructions that combine multiple steps and execute very efficiently


AES Crib Sheet
(Handy for memorizing)
Plaintext in 4x4 grid
0 4 8 C
1 5 9 D
2 6 A E
3 7 B F
Initial Round
Shift Rows Row Shift
0
1
2
3
X
Intermediate Rounds
| Key
9 | 128
11 | 192
13 | 256
Final Round
? ? ? ?
Ciphertext
General Math
11B = AES Polynomial = m(x)
x^8 + x^4 + x^3 + x + 1
Fast Multiply
x·a(x) = (a<<1) ⊕ (a7 = 1) ? 1B : 00
log(x·y) = log(x) + log(y)
Use (x+1) = 03 for log base
S-Box (SRD)
SRD[a] = f(g(a))
g(a) = a^-1 mod m(x)
f(a) Think 53 ⊕ 63^T
5 1's and 3 0's
[0110 0011]^T
11111000
01111100
00111110
00011111
10001111
11000111
11100011
11110001
a7 a6 a5 a4 a3 a2 a1 a0
0 1 1 0 0 0 1 1
Key Expansion:
Round Constants
First Column:
01 02 04 08 ...
S
0 1 B K
M 2 I E
E 8 T Y
Round Key 0
K
E
Y
B3 6E CB B7
01 00 00 00
B2 6E CB B7
S O M E
E1 21 86 F2
Other Columns:
T 2 8
C1 10 B4 CA
Prev Col ⊕ Col from Previous round key
Mix Columns:
2113
2 1 1 3
3 2 1 1
1 3 2 1
1 1 3 2
a3 a2 a1 a0
Inverse Mix
EBD9
E B D 9
9 E B D
D 9 E B
B D 9 B
a3 a2 a1 a0

STREAM CIPHERS

Stream ciphers are inspired by one-time pad



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

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

RC4: Key schedule

Start with the identity array

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Use the key to scramble the mapping

RC4: Generating key stream and updating state

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    t := (S[i] + S[j]) mod 256
    K := S[t]
    output K
endwhile
```

Another step of scrambling, like before but without using the key again

K is a byte of keystream, XOR'd with a byte of plaintext

Salsa20, a more modern stream cipher

Like in AES, internal state is a square

- In this case, a 4×4 square of 32-bit words
- A 256-bit key is broken into 8 words, arranged in the square along with constants and stream position / nonce values
- The function QR(a, b, c, d) can operate on a row or column:

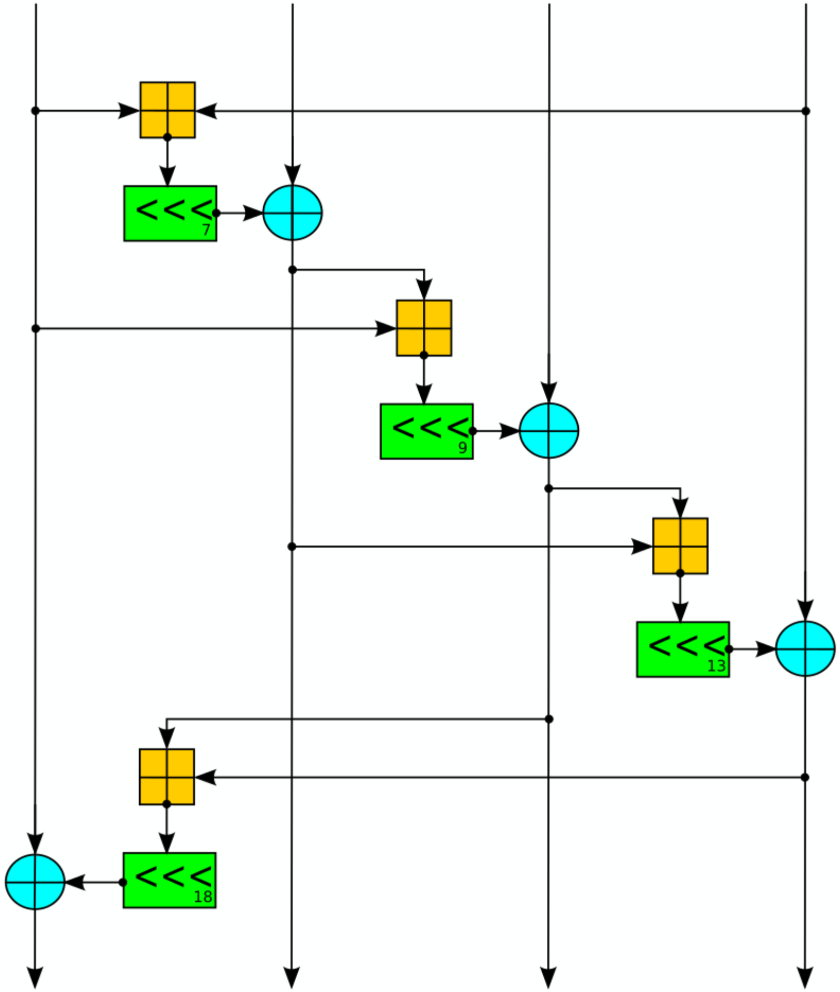
```
b ^= (a + d) <<< 7;
c ^= (b + a) <<< 9;
d ^= (c + b) <<< 13;
a ^= (d + c) <<< 18;
```

- Generation requires 20 rounds of mixing
  - Odd rounds use QR on each column, even rounds use QR on each row
- After all rounds, the new square is added to the previous
- The resulting square is 512 bits of keystream!

Salsa20 diagrams

Initial state of Salsa20

"expa"	Key	Key	Key
Key	"nd 3"	Nonce	Nonce
Pos.	Pos.	"2-by"	Key
Key	Key	Key	"te k"



One quarter-round

Two types of stream ciphers constructions

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

- ***Advantages***
  - Do not propagate transmission errors
  - Prevent insertion attacks
  - Key stream can be pre-generated
- ***Disadvantage:*** May need to change keys often if periodicity of PRNG is low

In a self-synchronizing stream cipher, the key stream is a function of some number of ciphertext bits

- ***Advantages***
  - Decryption key stream automatically synchronized with encryption key stream after receiving *n* ciphertext bits
  - Less frequent key changes, since key stream is a function of key and ciphertext
- ***Disadvantage:*** Vulnerable to replay attack

All is well?

Today we learned how symmetric-key cryptography can protect the confidentiality of our communications

So, the security problem is solved, right?

- What about integrity?

Unfortunately, symmetric key cryptography doesn't solve everything…

1. How do we get secret keys for everyone that we want to talk to?
2. How can we update these keys over time?

In about a week: Public key cryptography will help us with problem 1

Later in the semester: We'll look at key exchange protocols that help with problem 2

Next: Block modes of operation, integrity mechanisms