Block Ciphers

A **block cipher** encodes a plaintext in blocks of N bits

as opposed to a stream cipher, which can work on a stream of bits

Each N-bit plaintext becomes an N-bit ciphertext

Which is better, a *stream cipher* or *block cipher*?

- Neither
- It's complicated
- Just use AES, which is a block cipher

Block Ciphers

A **block cipher** encodes a plaintext in blocks of N bits

as opposed to a stream cipher, which can work on a stream of bits

Each N-bit plaintext becomes an N-bit ciphertext

We'll look at two block ciphers:

Data Encryption Standard (DES): older, broken at original key size

Advanced Encryption Standard (AES): newer, very widely used

DES

Developed in 1970s at IBM, standardized with input from NSA

64-bit block with 56-bit key

Three main components:

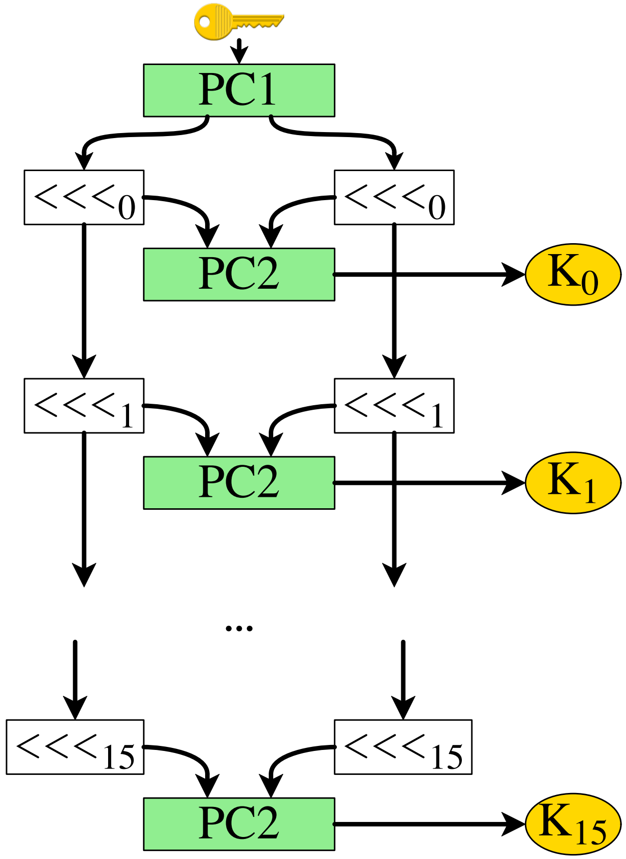
- **Key schedule** generated PRNG-like from the key

🔑 ⇒ $K_0, K_1, K_2, \ldots K_{15}$

- 16 rounds of *Feistel structure* mixing with key schedule as input
- Feistel function F to implement mixing

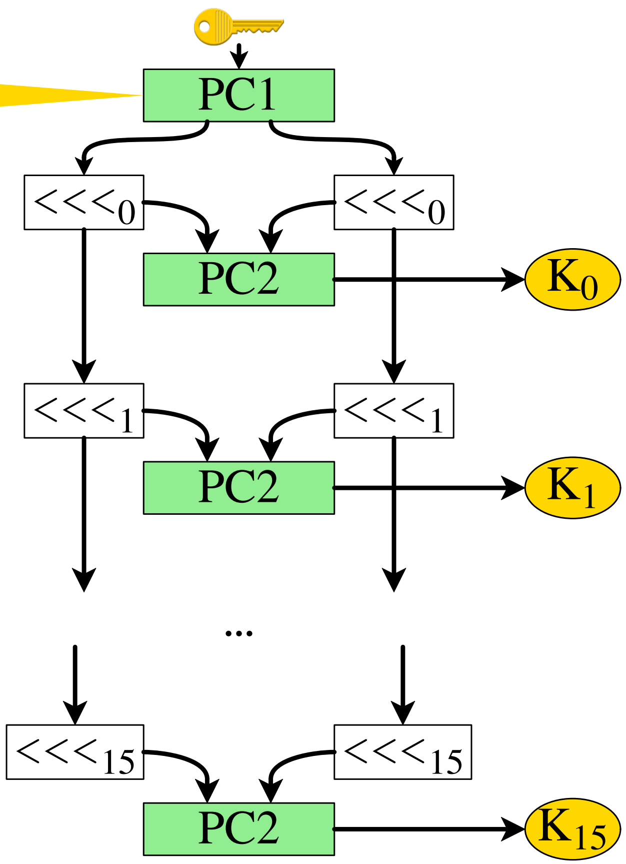
Following pictures are based on
https://en.wikipedia.org/wiki/Data_Encryption_Standard

DES Key Schedule



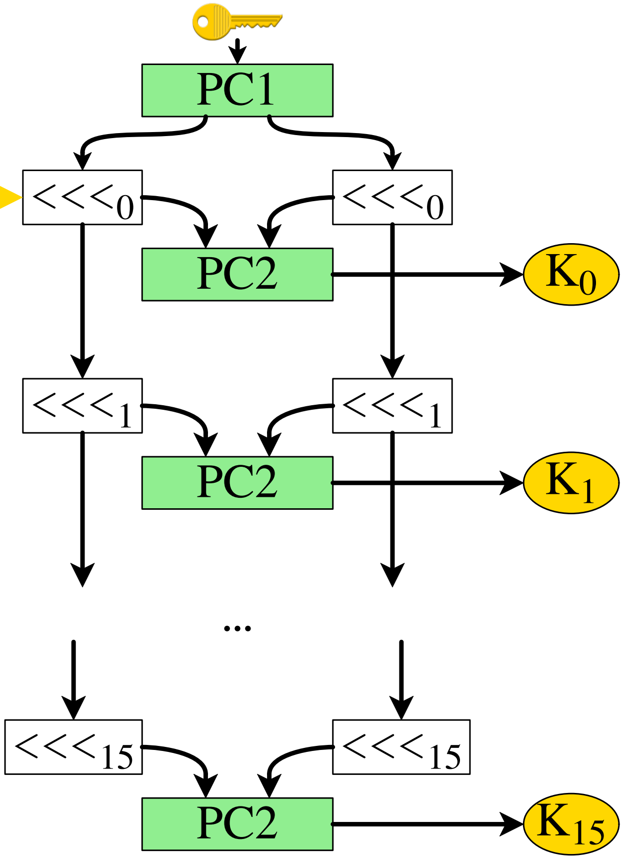
DES Key Schedule

Permuted Choice:
shuffle and pick 56 of 64 bits,
then split into two

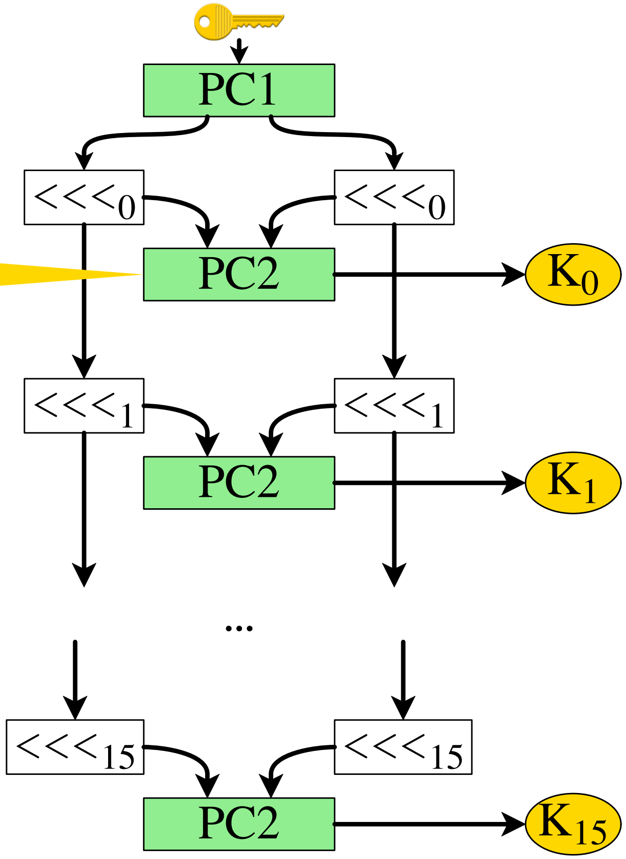


DES Key Schedule

Different rotation amount each step

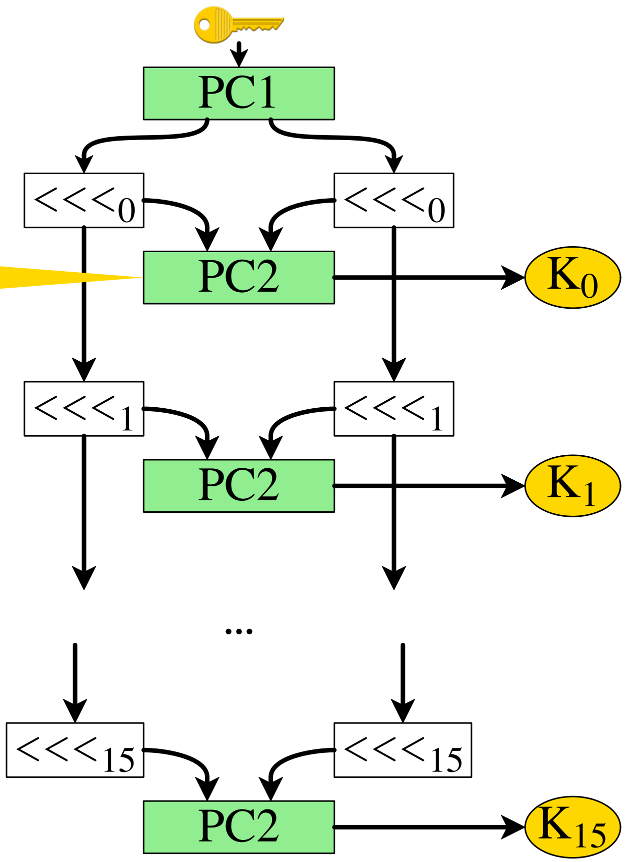


DES Key Schedule



shuffle and pick 48 of 56 bits

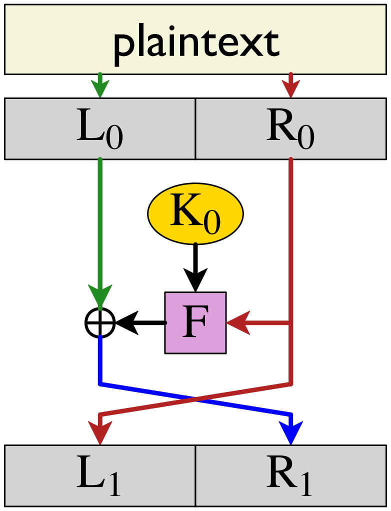
DES Key Schedule



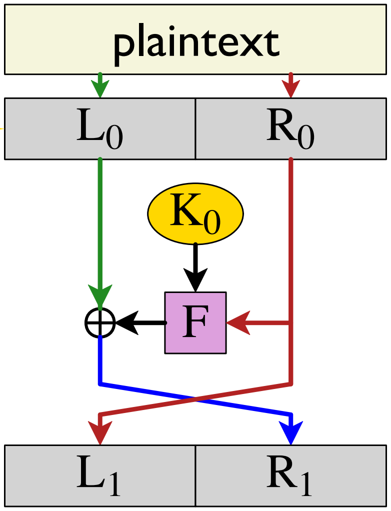
shuffle and pick 48 of 56 bits

Output is sequence of K_i, each 48 bits

DES Feistel Structure

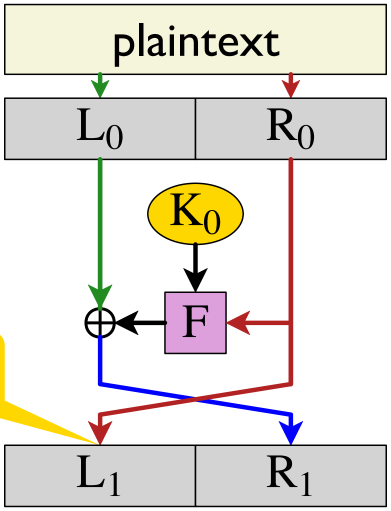


DES Feistel Structure



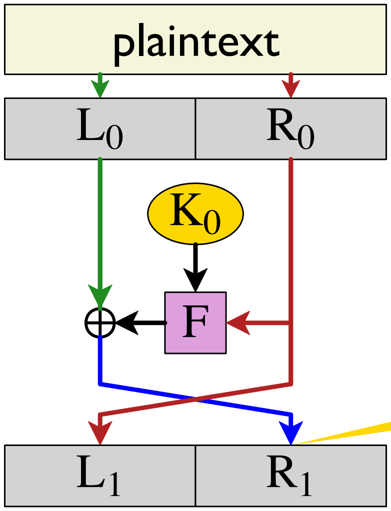
Split plaintext into 32-bit halves

DES Feistel Structure



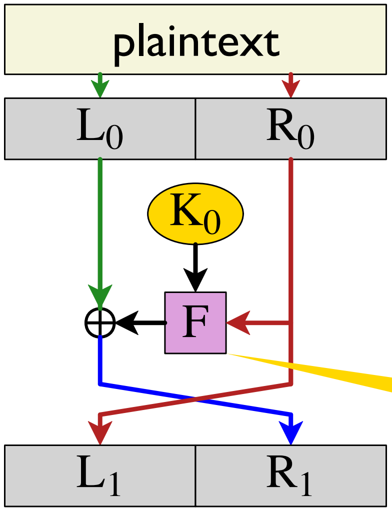
New left half is old right half

DES Feistel Structure



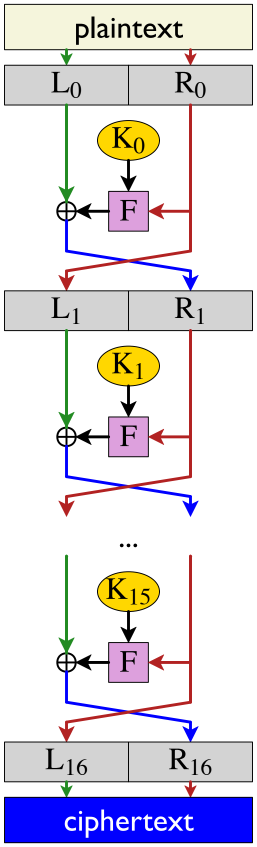
New right half depends on both old halves as mixed through K and F

DES Feistel Structure

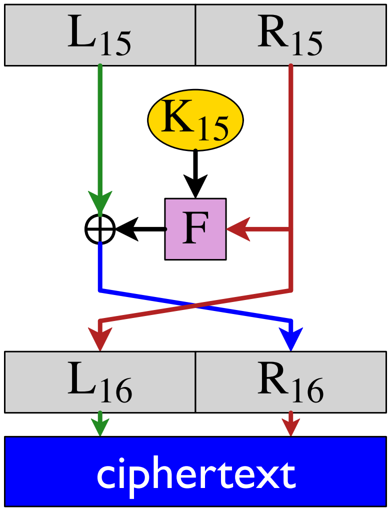


Feistel function F is *one-way*

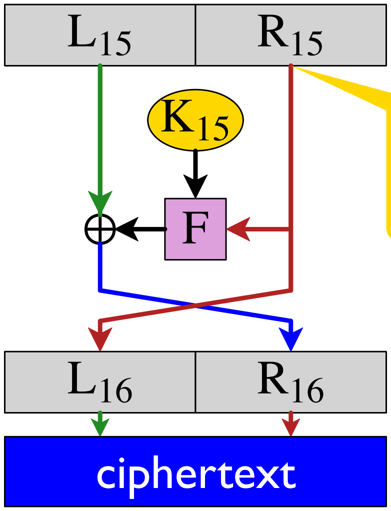
DES Feistel Structure



DES Feistel Structure

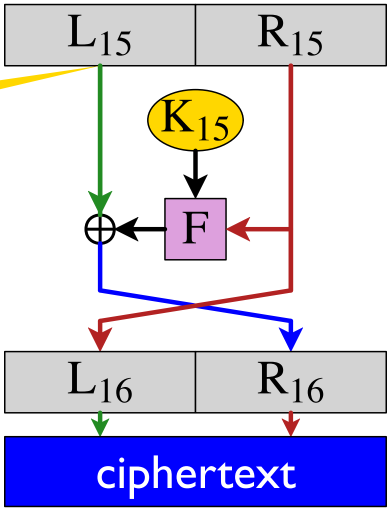


DES Feistel Structure



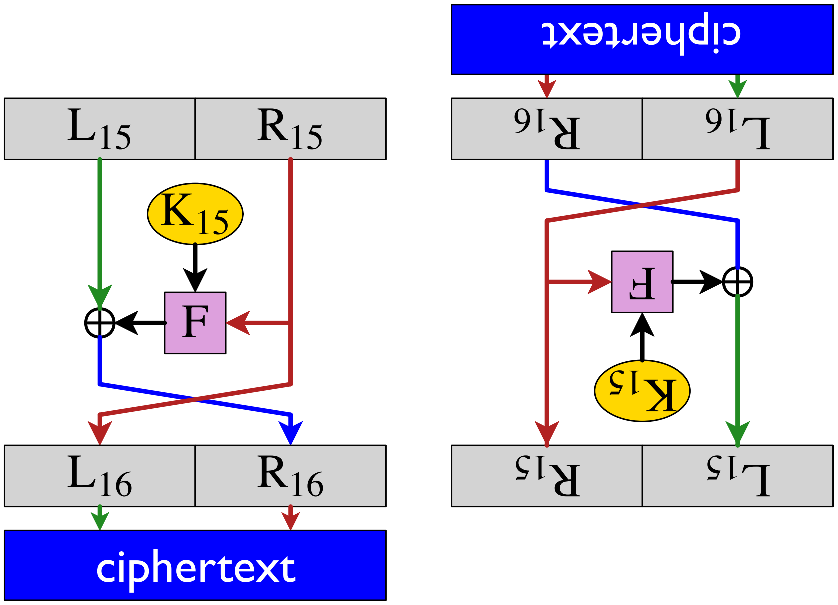
Decode-step right half is previous left half

DES Feistel Structure

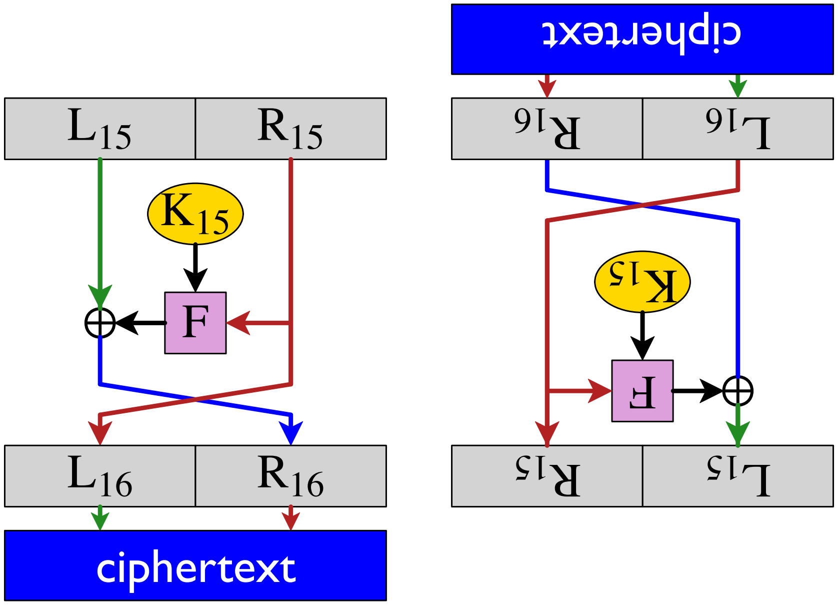


Decode-step left half depends on both previous halves as mixed through K and F

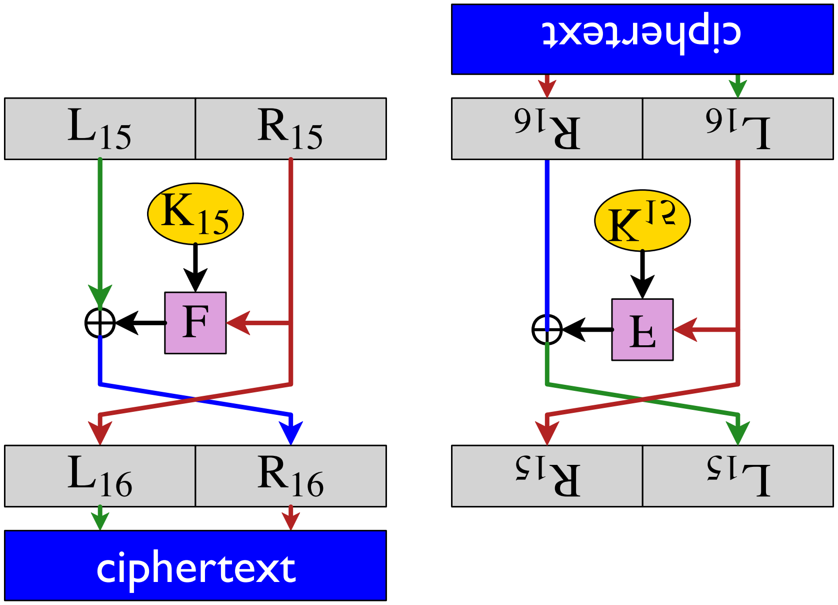
DES Feistel Structure



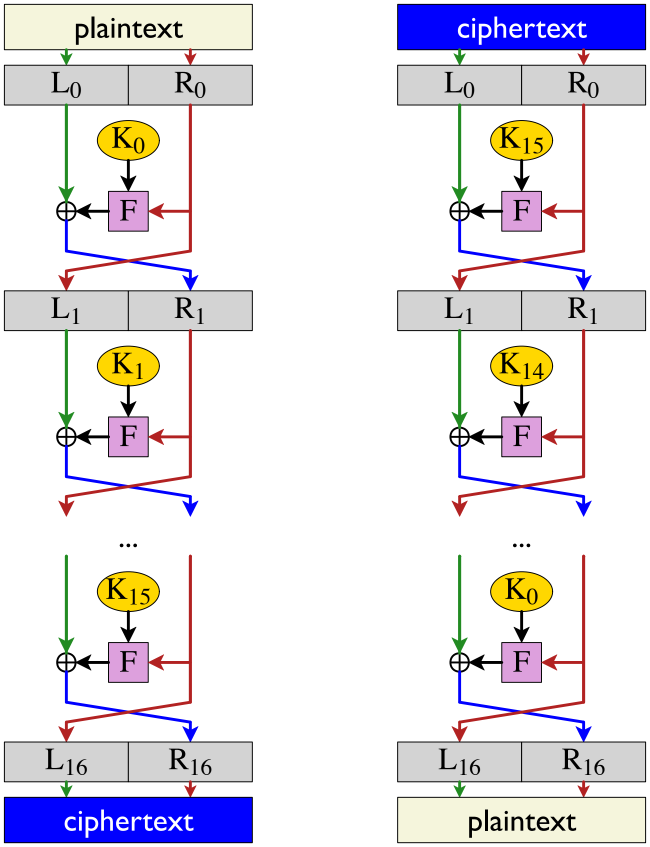
DES Feistel Structure



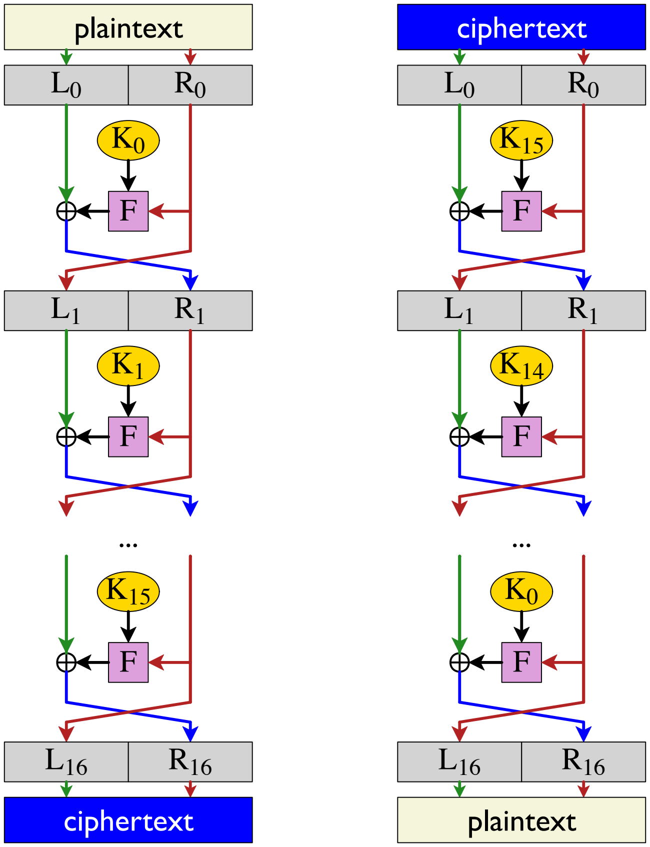
DES Feistel Structure



DES Feistel Structure

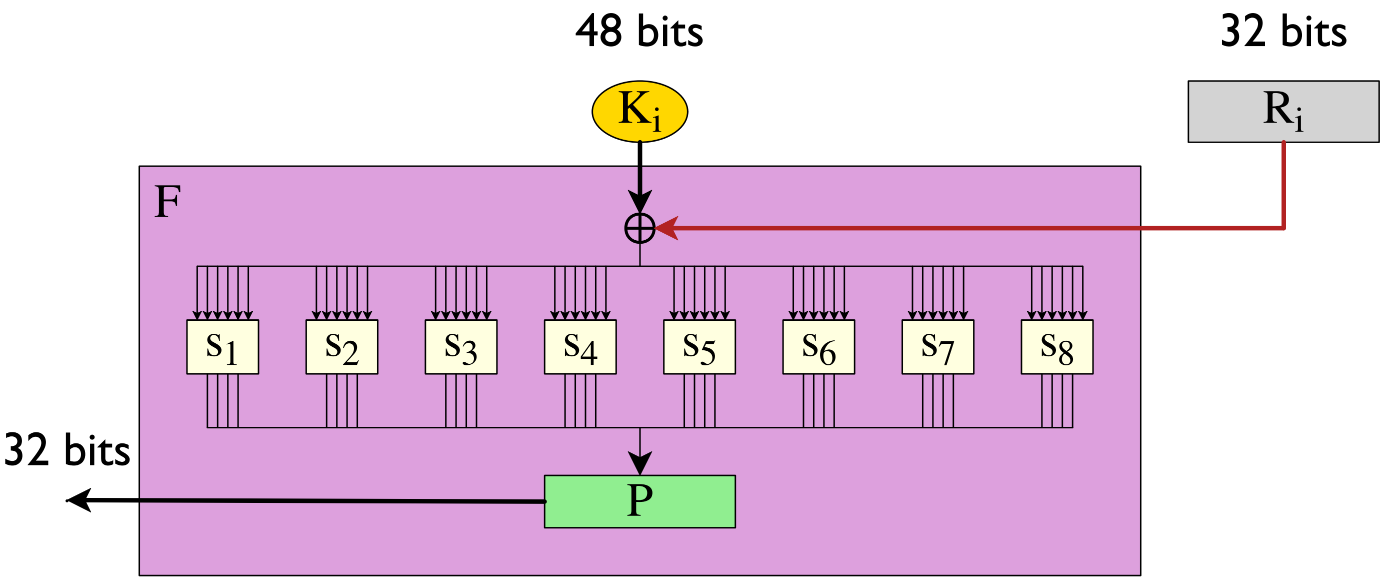


DES Feistel Structure

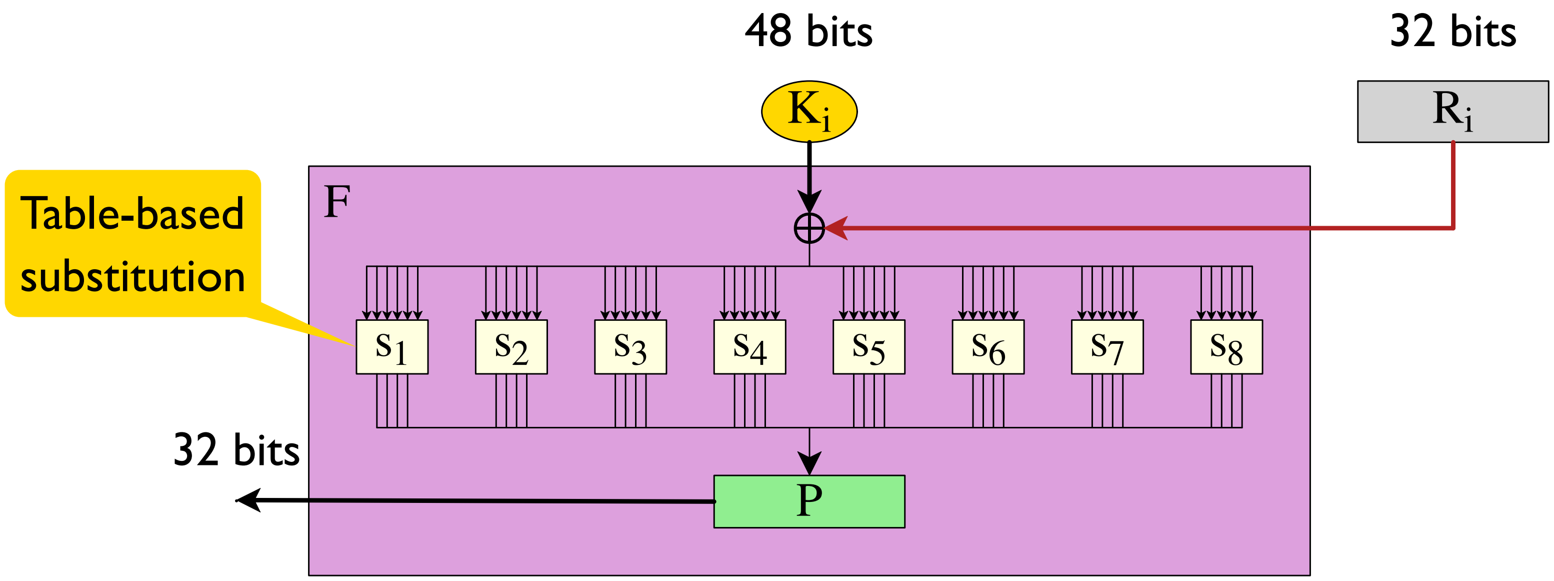


Encode and decode are the same function, just using the key schedule in opposite order

DES Feistel Function



DES Feistel Function



DES Feistel Function

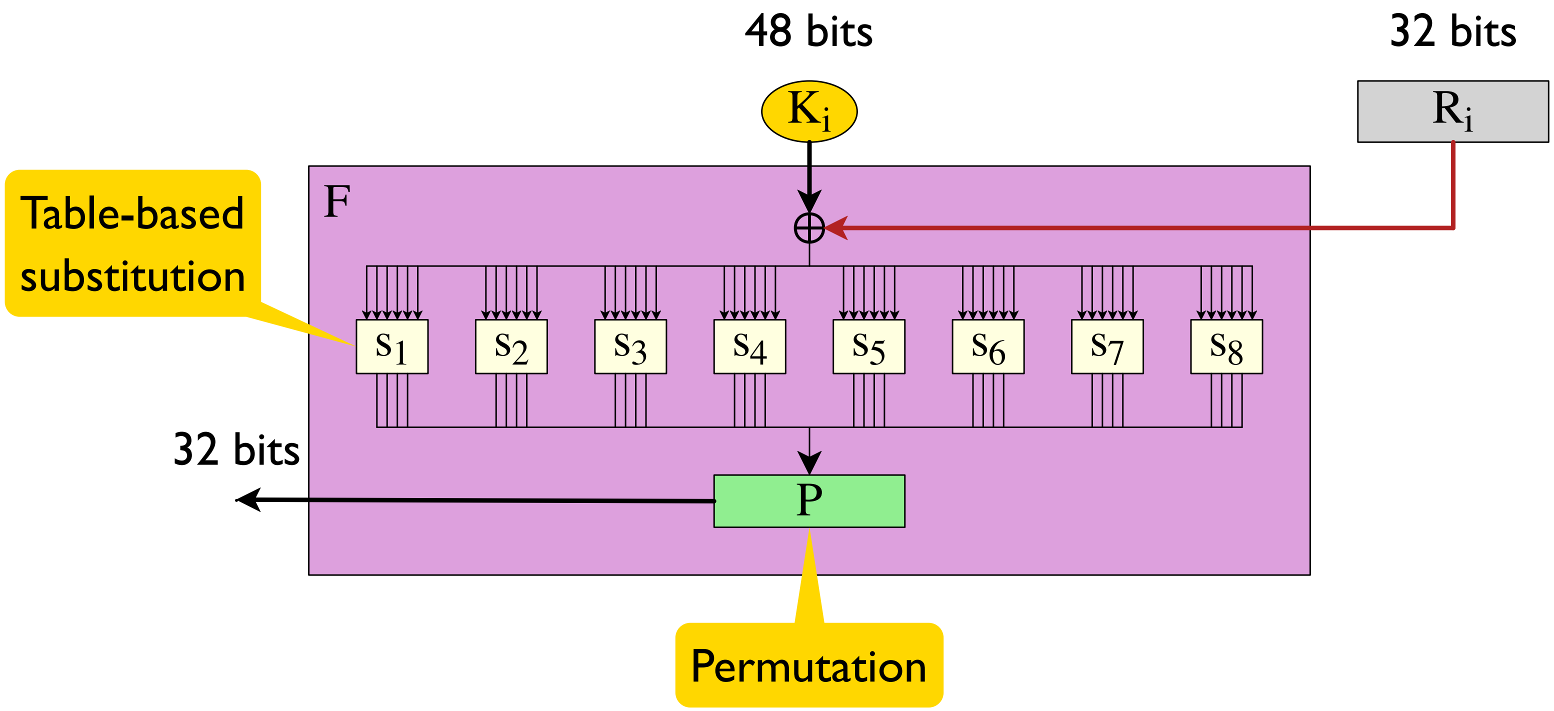
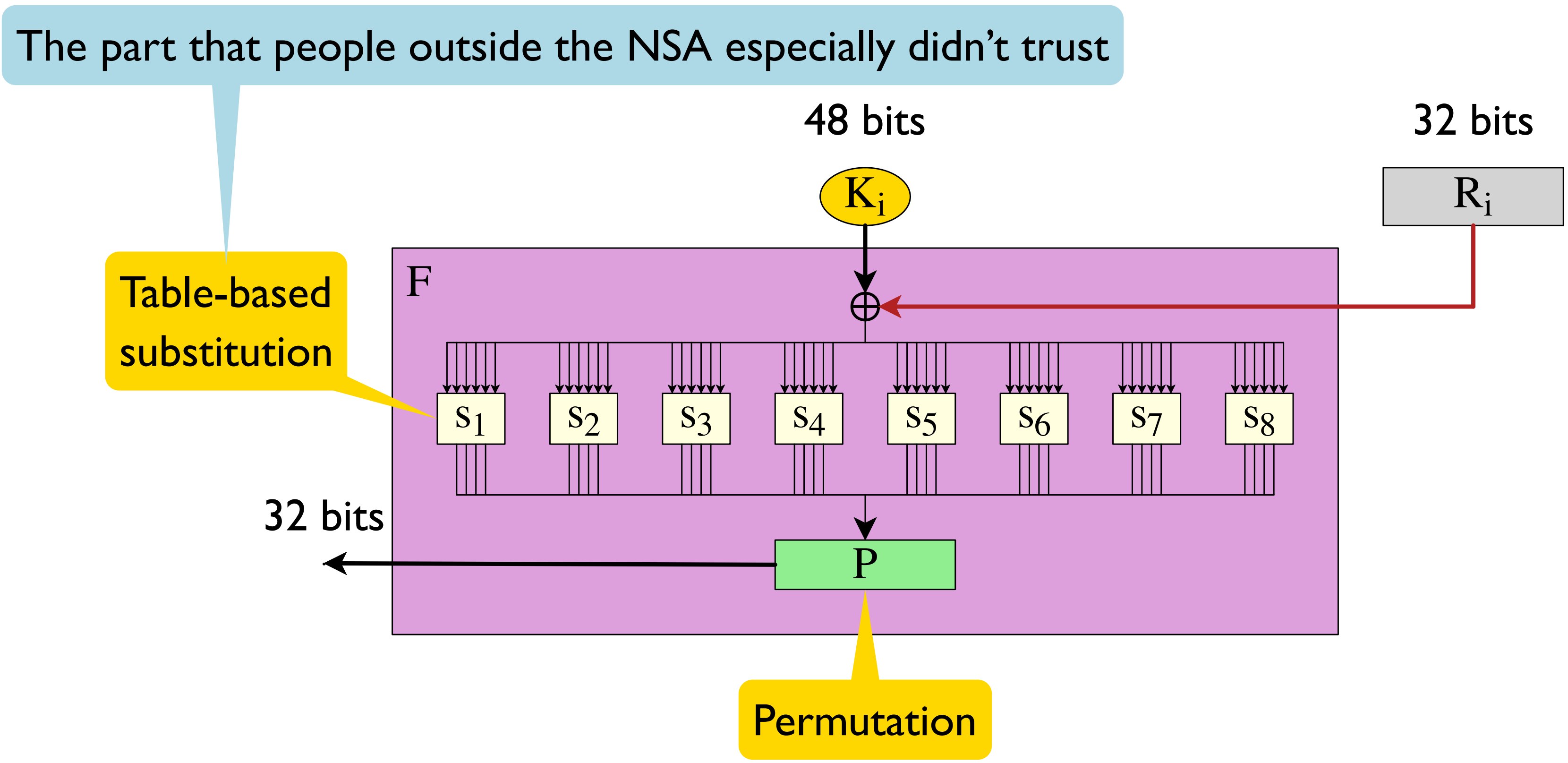


Table-based substitution

Permutation

DES Feistel Function



The part that people outside the NSA especially didn’t trust

Table-based substitution

Permutation

3DES

By the 1990s, a 56-bit key was too small

3DES is running DES three times:

$$\text{🔑} = \langle K_A, K_B, K_C \rangle$$

$$\text{Enc}_{\text{3DES}}(\text{🔑}, \boxed{\text{plaintext}}) = \text{Enc}_{\text{DES}}(K_A, \text{Dec}_{\text{DES}}(K_B, \text{Enc}_{\text{DES}}(K_C, \boxed{\text{plaintext}})))$$

DES Issues

Algorithm was designed for hardware

P bit permutations are a pain to implement in software with `and`, `or`, <<, and >>

Distrust of the secret design process

and especially the S_is

AES

Developed by an open competition in the 1990s run by NIST

Variant of an algorithm called **Rijndael**

128-bit block with 128-, 192-, or 256-bit key

Main components are analogous to DES:

- **Key schedule** generated from the key

  different PRNG-like generator

- 11, 13, or 15 rounds of mixing using key schedule as input

  different mixing function

- Reversible mixing function R (instead of Feistel structure)

  includes $\oplus$ of key from schedule

AES

Developed by an open competition in the 1990s run by NIST

Variant of an algorithm called **Rijndael**

128-bit block with 128-, 192-, or 256-bit key

Main components are analogous to DES:

- **Key schedule** generated from the key

Each K_i is 128 bits

different PRNG-like generator

- 11, 13, or 15 rounds of mixing using key schedule as input

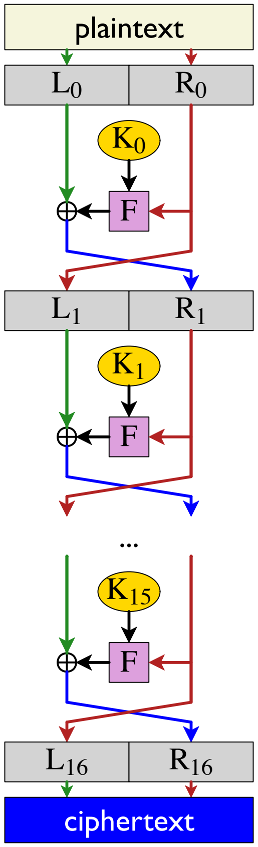
different mixing function

- Reversible mixing function R (instead of Feistel structure)

includes ⊕ of key from schedule

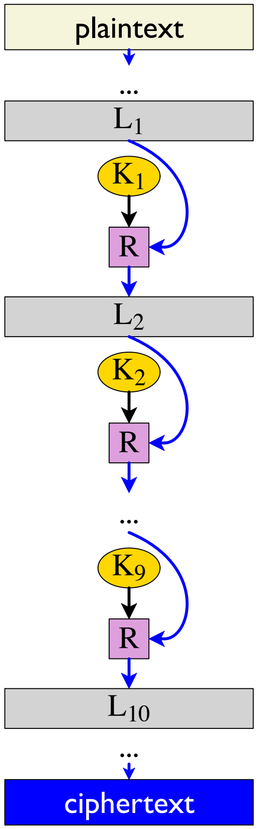
DES versus AES Structure

DES

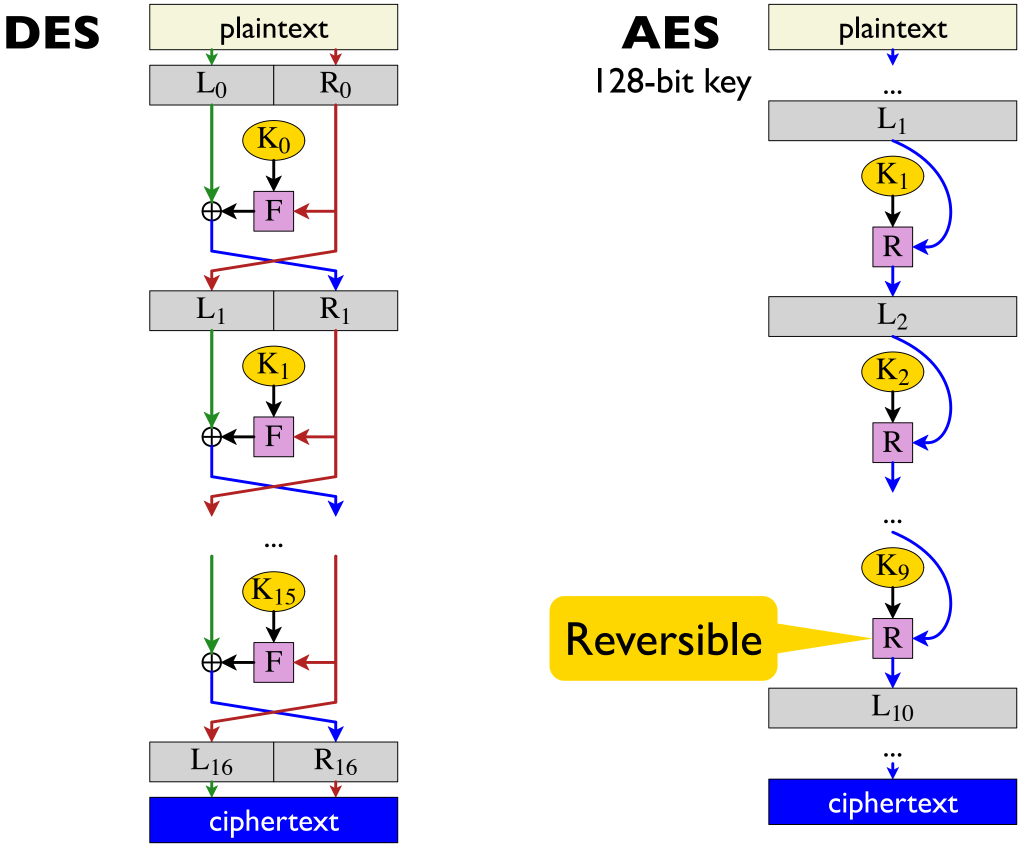


AES

128-bit key

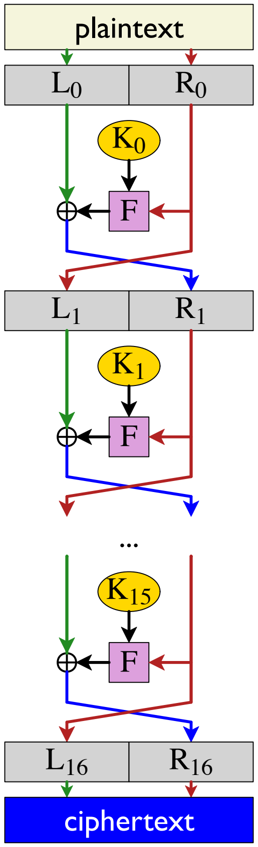


DES versus AES Structure



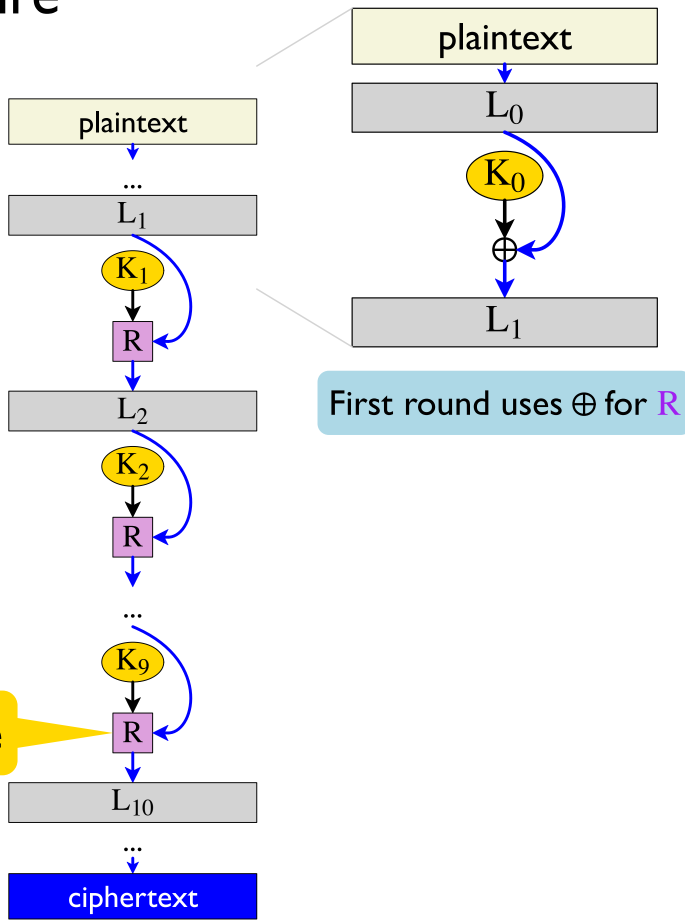
DES versus AES Structure

DES



AES

128-bit key

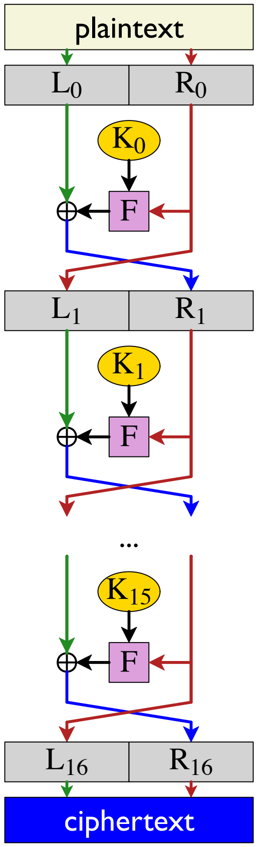


Reversible

First round uses ⊕ for R

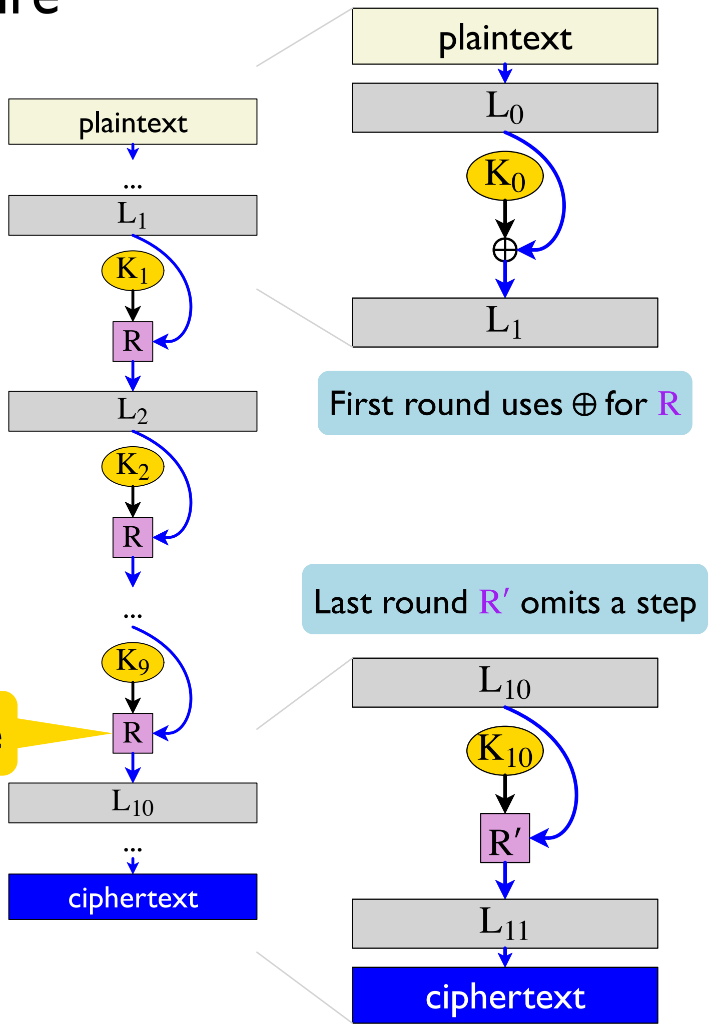
DES versus AES Structure

DES



AES

128-bit key



First round uses ⊕ for R

Last round R′ omits a step

Reversible

AES Round

View the state as an 4 × 4 array of bytes:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

AES Round

starts as plaintext

View the state as an 4 × 4 array of bytes:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

$$R(K_i, \text{state}) = \texttt{MixColumns}(\texttt{ShiftRows}(\texttt{SubBytes}(\text{state}))) \oplus K_i$$

$$R'(K_i, \text{state}) = \texttt{ShiftRows}(\texttt{SubBytes}(\text{state})) \oplus K_i$$

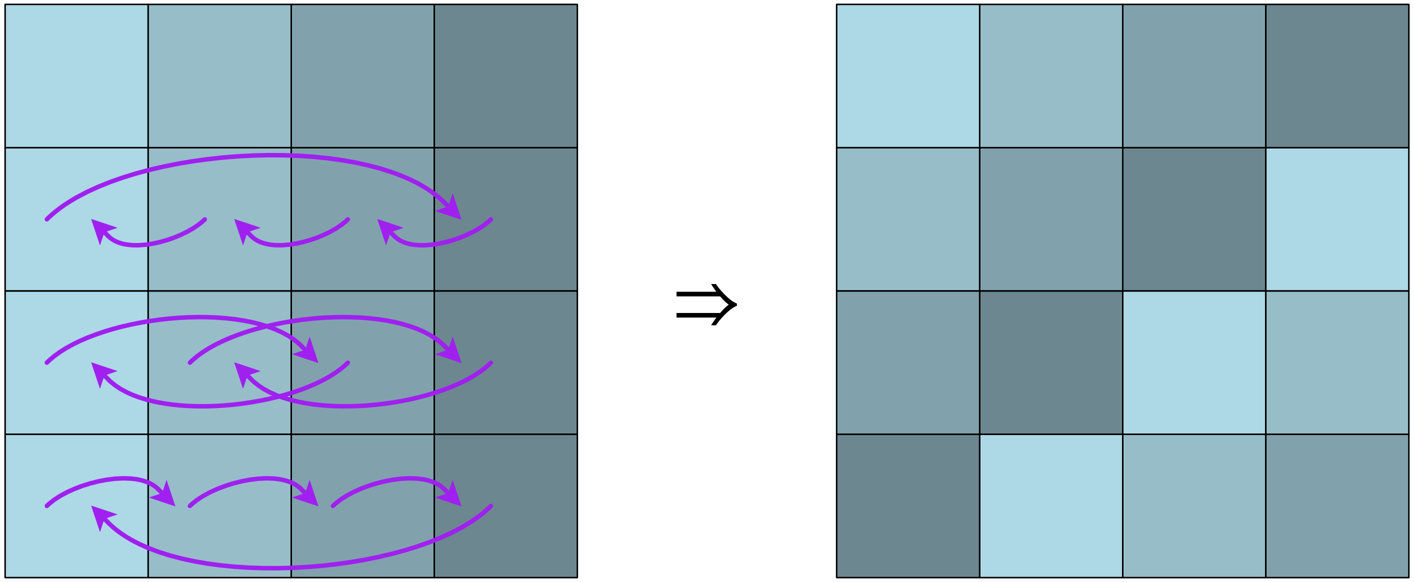
AES Substitution

`SubBytes` looks up a substitution in this table, which is based on a particular polynomial:

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

AES Shift Rows

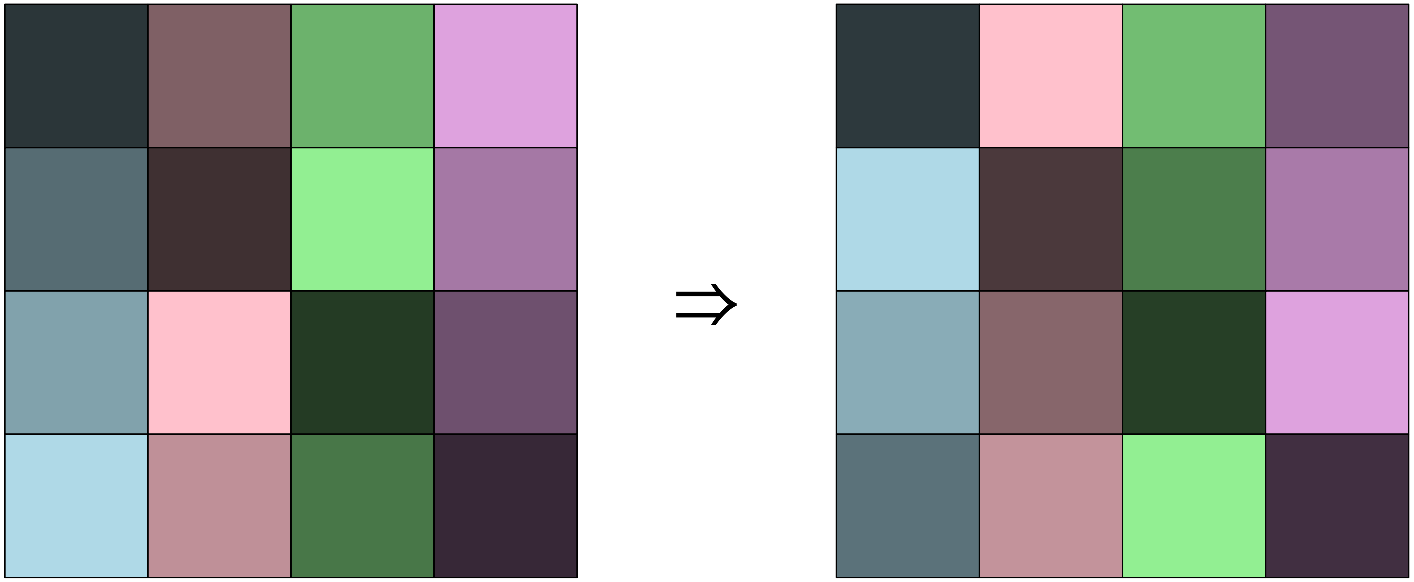
`ShiftRows` **rotates bytes within a row:**



AES Mix Columns

`MixColumns` “multiplies” each column by a fixed matrix

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

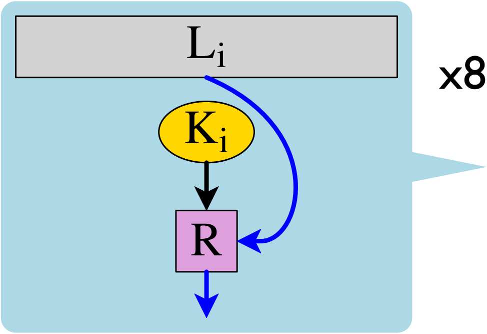


Processor Support for AES

x86 instructions for AES extension:

`AESENC`	Perform R
`AESENCLAST`	Perform R'
`AESDEC`	Perform inverse of R
`AESDECLAST`	Perform inverse of R'
`AESKEYGENASSIST`	Key sequence helper
`AESIMC`	Key sequence helper

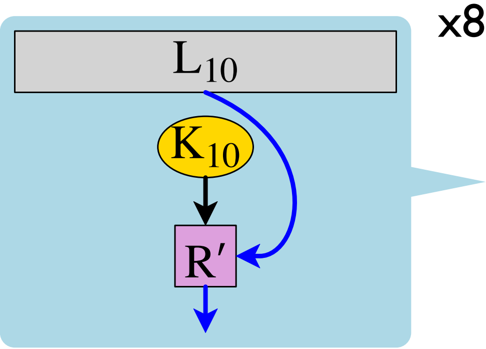
Processor Support for AES



x86 instructions for AES extension:

`AESENC`	Perform R
`AESENCLAST`	Perform R'
`AESDEC`	Perform inverse of R
`AESDECLAST`	Perform inverse of R'
`AESKEYGENASSIST`	Key sequence helper
`AESIMC`	Key sequence helper

Processor Support for AES

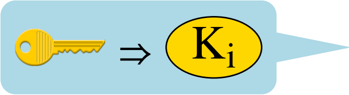


x86 instructions for AES extension:

`AESENC`	Perform R
`AESENCLAST`	Perform R'
`AESDEC`	Perform inverse of R
`AESDECLAST`	Perform inverse of R'
`AESKEYGENASSIST`	Key sequence helper
`AESIMC`	Key sequence helper

Processor Support for AES

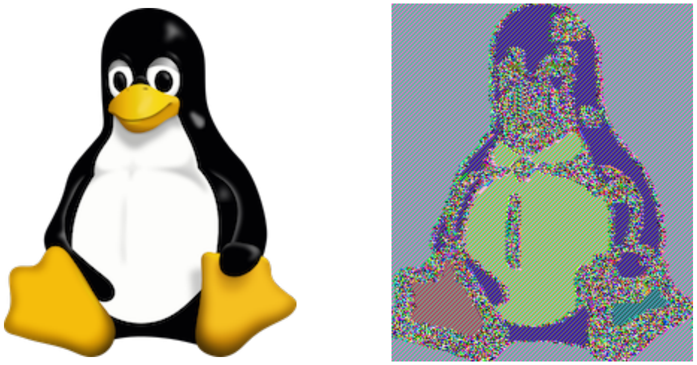
x86 instructions for AES extension:



`AESENC`	Perform R
`AESENCLAST`	Perform R'
`AESDEC`	Perform inverse of R
`AESDECLAST`	Perform inverse of R'
`AESKEYGENASSIST`	Key sequence helper
`AESIMC`	Key sequence helper

Block ciphers mix up individual blocks, but for a given 🔑, they always encode a plaintext block as a deterministic ciphertext block

What if your message has a lot of the same block repeated?



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining

Block ciphers mix up individual blocks, but for a given 🔑, they always encode a plaintext block as a deterministic ciphertext block

What if your message has a lot of the same block repeated?

Instead of

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i)$$

use

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i \oplus \text{ciphertext}_{i-1})$$

This is known as a **mode of operation**

Cipher Block Chaining

Block ciphers mix up individual blocks, but for a given 🔑, they always encode a plaintext block as a deterministic ciphertext block

What if your message has a lot of the same block repeated?

Instead of

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i)$$

use

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i \oplus \text{ciphertext}_{i-1})$$

Pick a random number to use with plaintext_0, and send that number first

This is known as a **mode of operation**

Cipher Block Chaining

Block ciphers mix up individual blocks, but for a given 🔑, they always encode a plaintext block as a deterministic ciphertext block

What if your message has a lot of the same block repeated?

Instead of

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i)$$

use

$$\text{ciphertext}_i = \text{Enc}_{\text{AES}}(\text{plaintext}_i \oplus \text{ciphertext}_{i-1})$$

This initial value is called an **initialization vector**

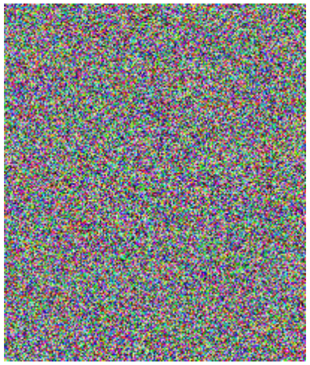
Pick a random number to use with plaintext_0, and send that number first

This is known as a **mode of operation**

Cipher Block Chaining

Block ciphers mix up individual blocks, but for a given 🔑, they always encode a plaintext block as a deterministic ciphertext block

What if your message has a lot of the same block repeated?



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Summary

Block ciphers encode chunks using a more complex combination with a random stream than $\oplus$

- **DES** — historical, key size was issue, expensive to compute
- **AES** — modern, large key sizes, fast on modern processors

Block ciphers still need a **mode of operation** to hide larger structure