

DES, Triple-DES, and AES



Sandy Kutin

CSPP 532

7/3/01

Symmetric Cryptography



- ⌘ Secure communication has two parts:
 - ☑ Establish a key (public key methods)
 - ☑ Encrypt message symmetrically using key
- ⌘ Symmetric encryption is faster
- ⌘ Cryptographic scheme is only as good as its “weakest link”
- ⌘ We need to understand strengths and weaknesses of symmetric encryption

DES: Data Encryption Standard



- ⌘ 1972: National Bureau of Standards begins search
- ⌘ 1975: DES: Lucifer by IBM, modified by NSA (key reduced from 128 to 56 bits)
- ⌘ Approved by NBS '76, ANSI '81
- ⌘ renewed every 5 years by NIST
- ⌘ now considered obsolete

DESiderata



⌘ Secure: hard to attack

- ☑ Classic case: given ciphertext, get plaintext
- ☑ Also: given both, get key
- ☑ Achieved through diffusion, confusion

⌘ Easy to implement (in hardware, software)

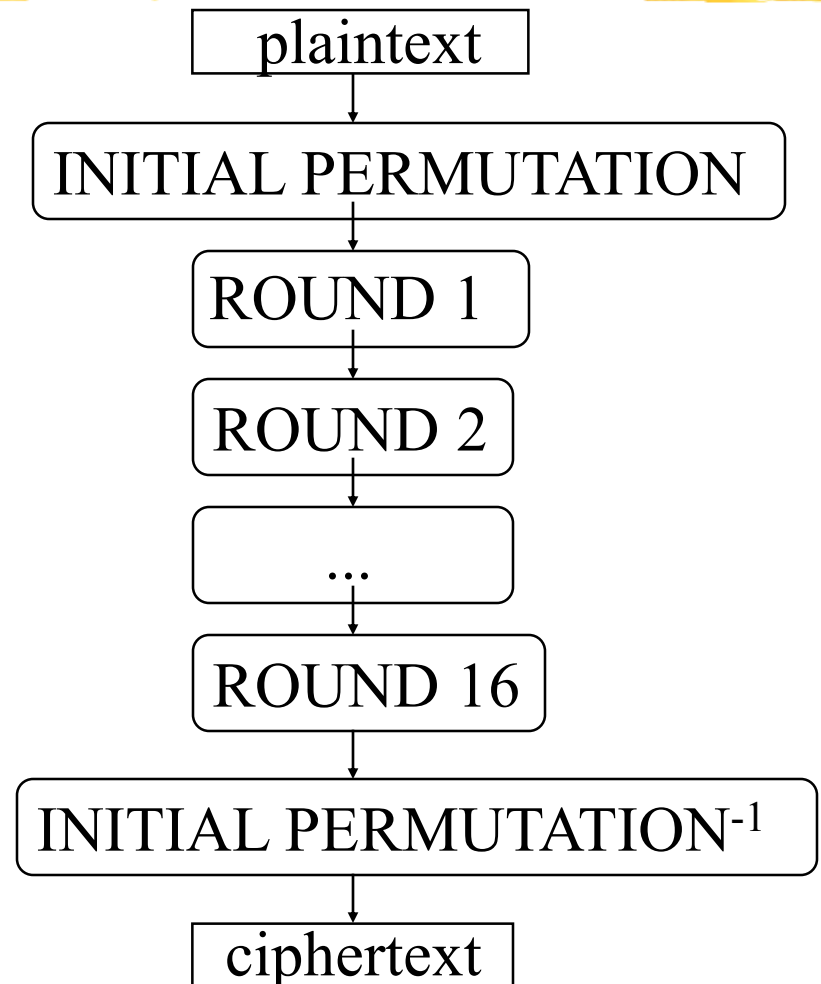
- ☑ Use a few fast subroutines
- ☑ Decryption uses same routines

⌘ Easy to analyze

- ☑ Prove that certain attacks fail

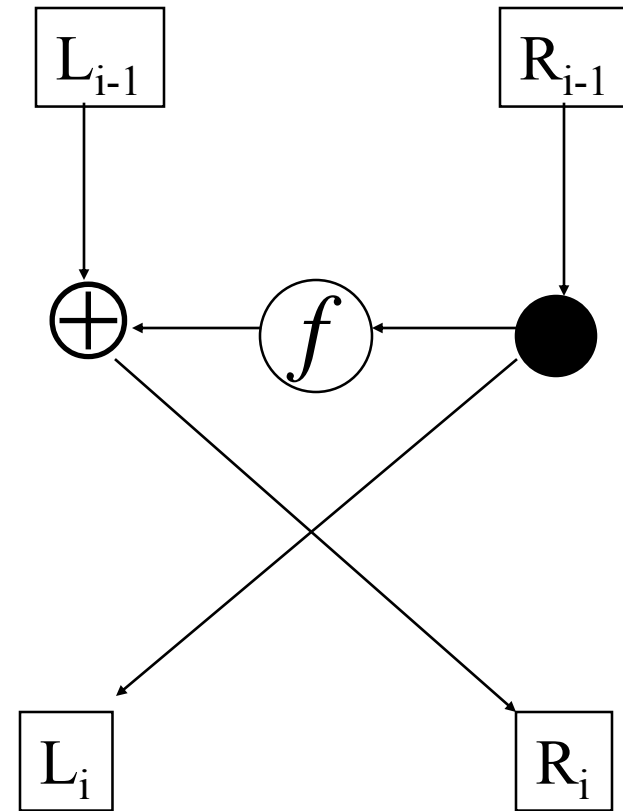
DEScriptor: Overview

- ⌘ Block cipher: 64 bits at a time
- ⌘ Initial permutation rearranges 64 bits (no cryptographic effect)
- ⌘ Encoding is in 16 rounds



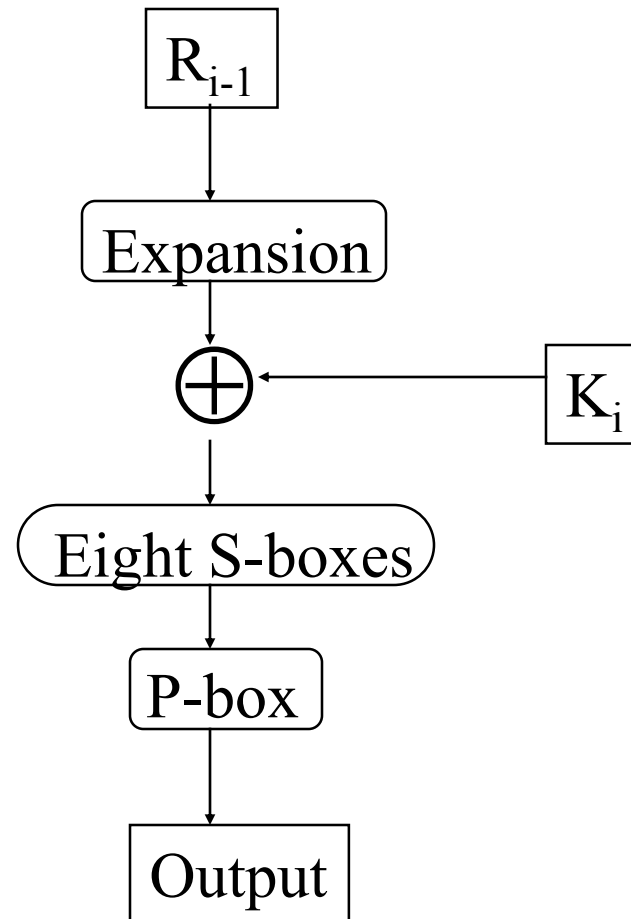
DEScriptio: One Round

- ⌘ 64 bits divided into left, right halves
- ⌘ Right half goes through function f , mixed with key
- ⌘ Right half added to left half
- ⌘ Halves swapped (except in last round)



DEScriptio: InsiDES

- ⌘ Expand right side from 32 to 48 bits (some get reused)
- ⌘ Add 48 bits of key (chosen by schedule)
- ⌘ S-boxes: each set of 6 bits reduced to 4
- ⌘ P-box permutes 32 bits



DESign Principles: Inverses

⌘ Equations for round i :

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1})$$

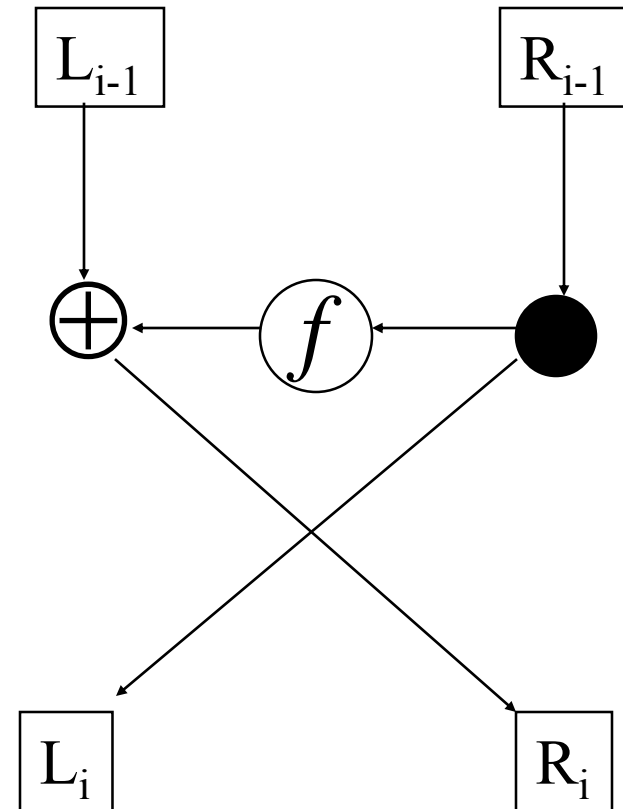
⌘ In other words:

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(L_i)$$

⌘ So decryption is the same as encryption

⌘ Last round, no swap:
really is the same



MoDES of Operation

⌘ ECB: Electronic CodeBook mode:

- ☑ Encrypt each 64-bit block independently
- ☑ Attacker could build codebook

⌘ CBC: Cipher Block Chaining mode:

- ☑ Encryption: $C_i = E_K(P_i \oplus C_{i-1})$
- ☑ Decryption: $P_i = C_{i-1} \oplus D_K(C_i)$

⌘ CFB, OFB: allow byte-wise encryption

- ☑ Cipher FeedBack, Output FeedBack

PeDEStrian attacks

- ⌘ Obvious attack: guess the key. 2^{56} keys
- ⌘ Complementation Property: 2^{55} keys
- ⌘ 1 million per second: 1100 years
- ⌘ Store $E_K(P_1)$ for all K : 512 petabytes
- ⌘ Time/Memory Tradeoff (Hellman, 1980):
 - ⌘ 1 terabyte
 - ⌘ 5 days

DEStroying Security

- ⌘ Differential Cryptanalysis (1990):
- ⌘ Say you know plaintext, ciphertext pairs
- ⌘ Difference $d_P = P_1 \oplus P_2$, $d_C = C_1 \oplus C_2$
- ⌘ Distribution of d_C 's given d_P may reveal key
- ⌘ Need lots of pairs to get lots of good d_P 's
- ⌘ Look at pairs, build up key in pieces
- ⌘ Could find some bits, brute-force for rest

DEServing of Praise

⌘ Against 8-round DES, attack requires:

☑ $2^{14} = 16,384$ chosen plaintexts, or

☑ 2^{38} known plaintext-ciphertext pairs

⌘ Against 16-round DES, attack requires:

☑ 2^{47} chosen plaintexts, or

☑ Roughly $2^{55.1}$ known plaintext-ciphertext pairs

⌘ Differential cryptanalysis not effective

⌘ Designers knew about it

DESperate measures

⌘ Linear cryptanalysis:

- ☑ Look at algorithm structure: find places where, if you XOR plaintext and ciphertext bits together, you get key bits

- ☑ S-boxes not linear, but can approximate

⌘ Need 2^{43} known pairs; best known attack

⌘ DES apparently not optimized against this

⌘ Still, not an easy-to-mount attack

DESuetude



- ⌘ “Weakest link” is size of key
- ⌘ Attacks take advantage of encryption speed
- ⌘ 1993: Weiner: \$1M machine, 3.5 hours
- ⌘ 1998: EFF’s Deep Crack: \$250,000
 - ☑ 92 billion keys per second; 4 days on average
- ⌘ 1999: distributed.net: 23 hours
- ⌘ OK for some things (e.g., short time horizon)
- ⌘ DES slides into wiDESspread DESuetude

Triple-DES

⌘ Run DES three times:

☑ ECB mode: $C_i = E_{K_3} \left(D_{K_2} \left(E_{K_1} (P_i) \right) \right)$

⌘ If $K_2 = K_3$, this is DES

☑ Backwards compatibility

⌘ Known not to be just DES with K_4 (1992)

⌘ Has 112 bits of security, not $3 \times 56 = 168$

⌘ Why? What's the attack?

⌘ What's wrong with Double-DES?

DESpair



- ⌘ Double-DES: $C_i = E_B(E_A(P_i))$
- ⌘ Given P_1, C_1 : Note that $D_B(C_1) = E_A(P_1)$
- ⌘ Make a list of every $E_K(P_1)$.
- ⌘ Try each L : if $D_L(C_1) = E_K(P_1)$, then maybe $K = A, L = B$. (2^{48} L 's might work.)
- ⌘ Test with P_2, C_2 : if it checks, it was probably right.
- ⌘ Time roughly 2^{56} . Memory very large.

Advanced Encryption Standard

- ⌘ DES cracked, Triple-DES slow: what next?
- ⌘ 1997: AES announced, call for algorithms
- ⌘ August 1998: 15 candidate algorithms
- ⌘ August 1999: 5 finalists
- ⌘ October 2000: Rijndael selected
 - ☑ Two Belgians: Joan Daemen, Vincent Rijmen
- ⌘ May 2001: Comment period ended
- ⌘ Summer 2001: Finalized, certified until '06

AESthetics



- ⌘ Similar to DES: block cipher (with different modes), but 128-bit blocks
- ⌘ 128-bit, 192-bit, or 256-bit key
- ⌘ Mix of permutations, “S-boxes”
- ⌘ S-boxes based on modular arithmetic with polynomials:
 - ☑ Non-linear
 - ☑ Easy to analyze, prove attacks fail

AES: State array

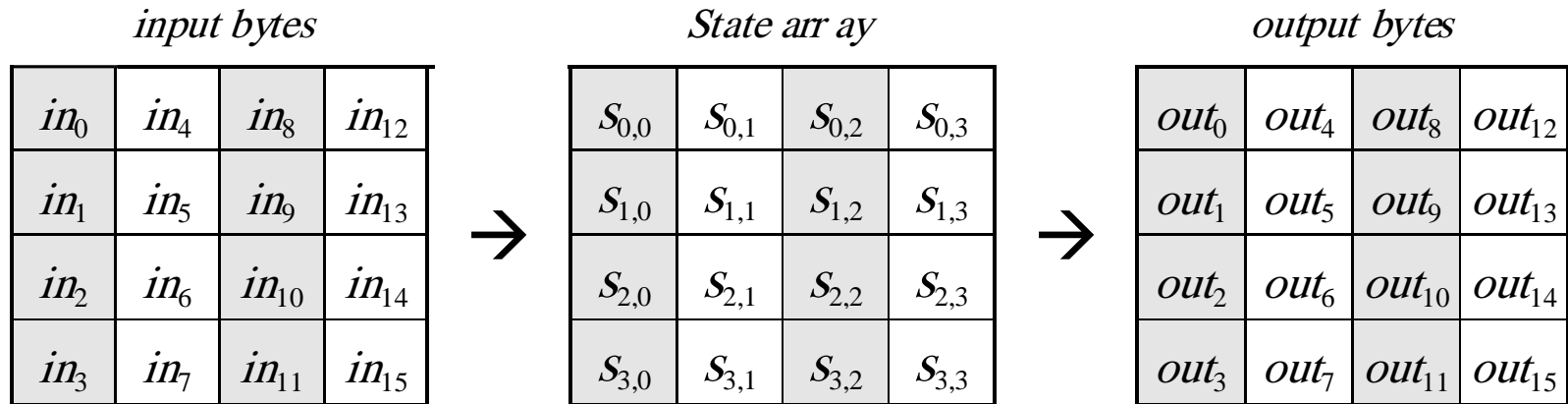


Figure 3. State array input and output.

“State” of machine given by 4x4 array of bytes

AES: Pseudocode

```
Cipher(byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr + 1)])  
begin  
    byte state[4, Nb]  
  
    state = in  
  
    AddRoundKey(state, w) // See Sec. 5.1.4  
  
    for round = 1 step 1 to Nr - 1  
        SubBytes(state) // See Sec. 5.1.1  
        ShiftRows(state) // See Sec. 5.1.2  
        MixColumns(state) // See Sec. 5.1.3  
        AddRoundKey(state, w + round * Nb)  
    end for  
  
    SubBytes(state)  
    ShiftRows(state)  
    AddRoundKey(state, w + Nr * Nb)  
  
    out = state
```

AES: SubBytes() (S-Box)



Figure 7. `SubBytes()` applies the S-box to each byte of the State.

Non-linear, based on polynomial arithmetic

AES: ShiftRows()

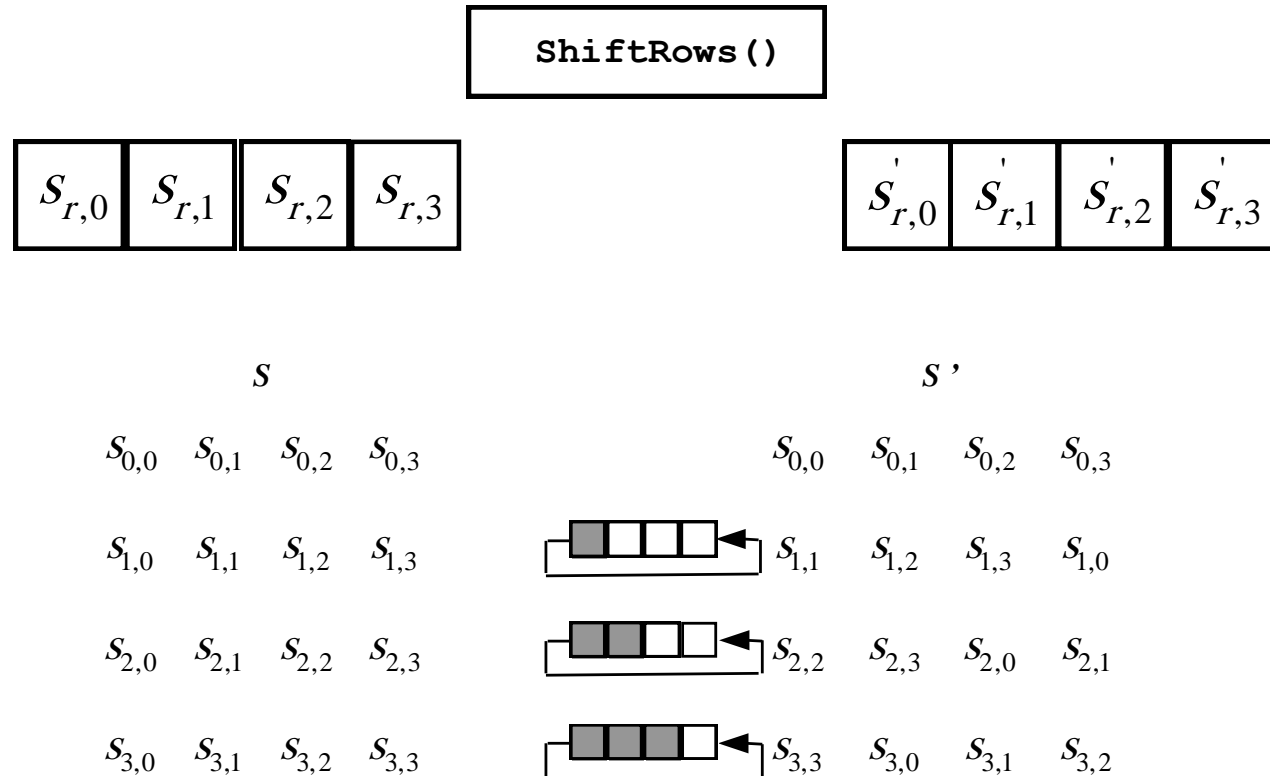


Figure 9. `ShiftRows ()` cyclically shifts the last three rows in the State

AES: MixColumns()

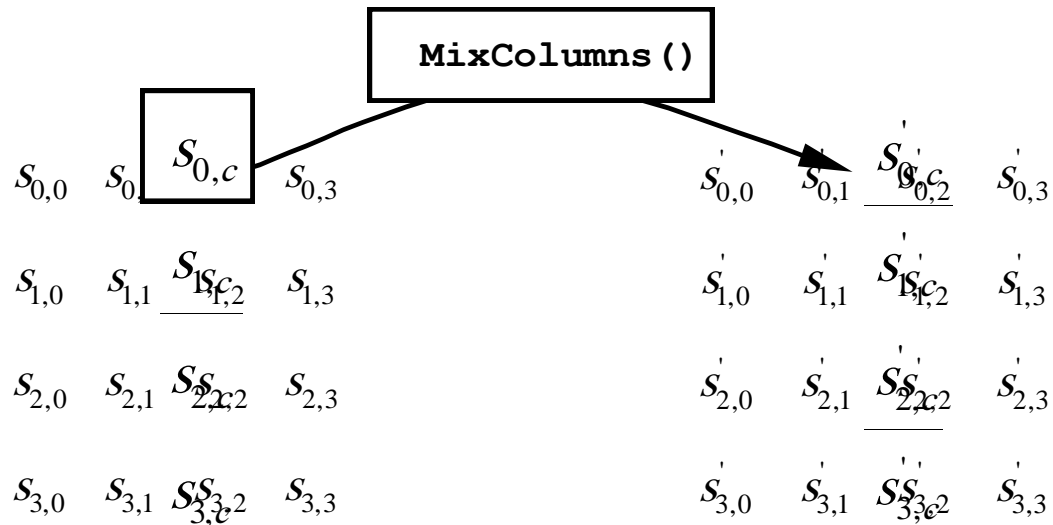


Figure 10. **MixColumns()** operates on the State column-by-column

AES: AddRoundKey()

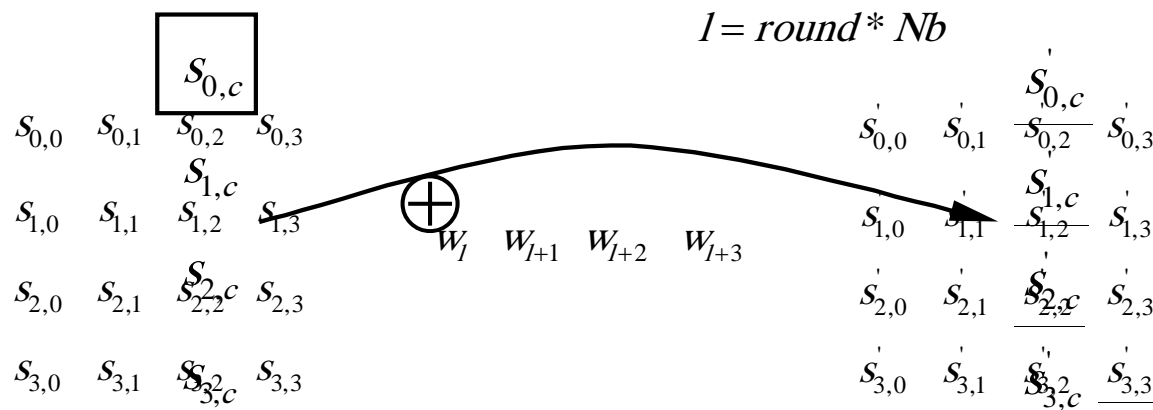


Figure 11. **AddRoundKey()** XORs each column of the State with a word from the key schedule.

Key schedule: expand N_b -word key to 4 words per round for $(6 + N_b)$ rounds (N_b could be 4, 6, or 8)

Not just a CAESar Shift

- ⌘ A byte $B = b_7b_6b_5b_4b_3b_2b_1b_0$ is a polynomial $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$
- ⌘ Can add, subtract, multiply polynomials
- ⌘ Coefficients are manipulated mod 2
- ⌘ Do polynomial division, get remainders
- ⌘ Can work “mod” a particular polynomial
- ⌘ AES uses a particular “prime” polynomial

KafkAESque Complexity

⌘ S-box: input is a byte B

☑ First take $B^{-1} \pmod{p}$

☑ Next, do a linear transformation on the bits

☑ Finally, XOR with a fixed byte

⌘ MixColumns() also uses polynomials

⌘ S-box can be done with a lookup table

⌘ Easier to analyze than “random” S-boxes used in DES

Suggested Reading



- ⌘ Chapter references are to Stallings
- ⌘ Modular Arithmetic: Sections 7.1-7.3, 7.5
- ⌘ Big-Oh Notation: Appendix 6A
- ⌘ DES: Chapter 3
- ⌘ Double-DES, Triple-DES: Section 4.1
- ⌘ AES: The AES home page:
<http://csrc.nist.gov/encryption/aes/>