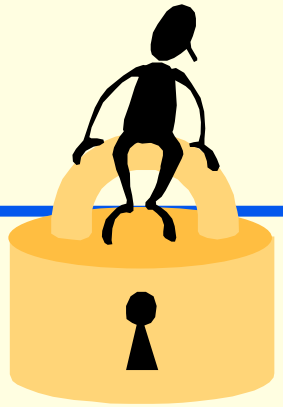


Symmetric Key Cryptography

Michael Huth

M.Huth@doc.ic.ac.uk

www.doc.ic.ac.uk/~mrh/430/

Introduction

- Also known as SECRET KEY, SINGLE KEY, PRIVATE KEY
- Assumption: Sender and Receiver share already a secret key
- Assumption requires solution to key-distribution problem
- Symmetric key algorithms also popular for file encryption, then

  Encrypter = Decrypter

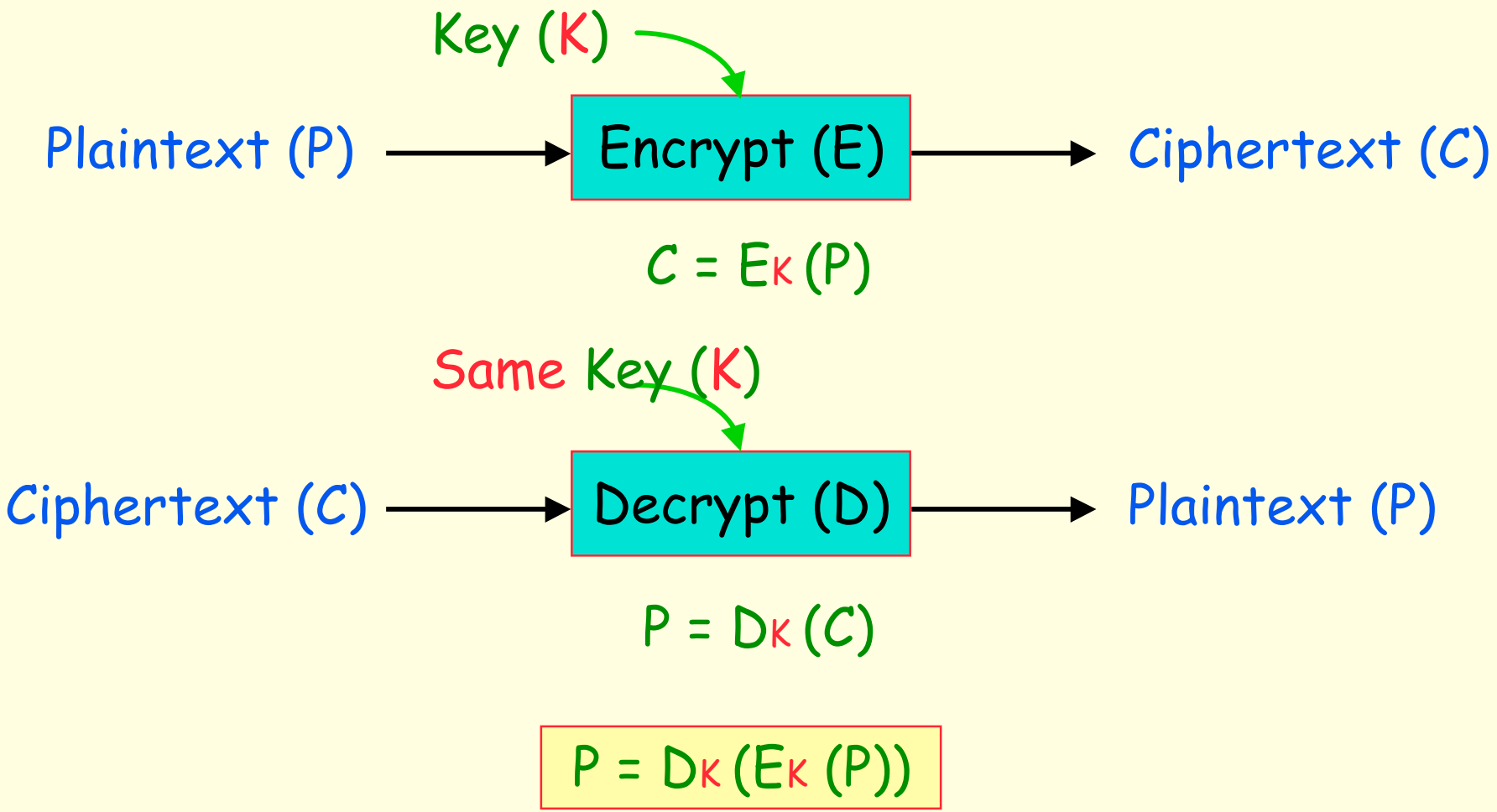
WEAK ALGORITHMS

- Classical substitution and transposition ciphers, as discussed last week

"STRONGER" ALGORITHMS

- DES - No longer considered safe
- Triple-DES
- AES (Rijndael)
- IDEA
- RC5, RC6
- Blowfish
- Many others

Encryption & Decryption



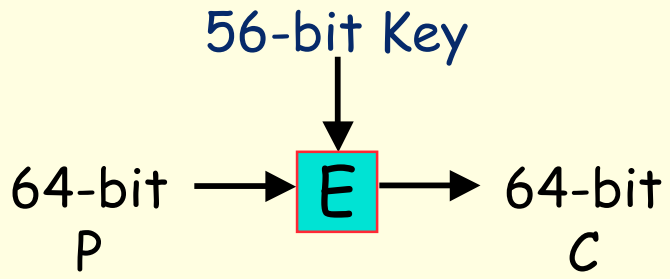
DES - Data Encryption Standard

- Intended usage:
  * Unclassified government business (USA)
  * Sensitive private sector business
- Was legally a munition in the US, like rocket launchers. DES could not be legally exported from the US as software (but could be published in a US book, or printed on a T-Shirt!)
- Re-certified every five years, i.e. 1983, 1988, 1993. US NSA ("National Security Agency" aka "No Such Agency") were reluctant for DES to be re-certified in 1988.

- 1973 - US NBS ("National Bureau of Standards", now called NIST) request for proposals.
  None judged worthy.
- 1974 - 2nd request for proposals.
  * US NSA urges IBM to submit its cipher Lucifer
  * US NSA modifies IBM's submission.
- 1975 - US NBS publishes proposal
  Much comment about US NSA modifications, e.g. fear of backdoors, shortening of key from 128 to 56 bits
- 1976 - DES Standard published.
  US NSA thought standard would be HW only, but NBS published enough details for software implementation.
- 1976 - 1998 DES widely used worldwide
- 1998 – DES brute force attackable

Basics

- Plaintext encrypted 64-bits at a time.
- 56 bits used for key.
- $2^{56} = 7.2 \times 10^{16}$ possible keys

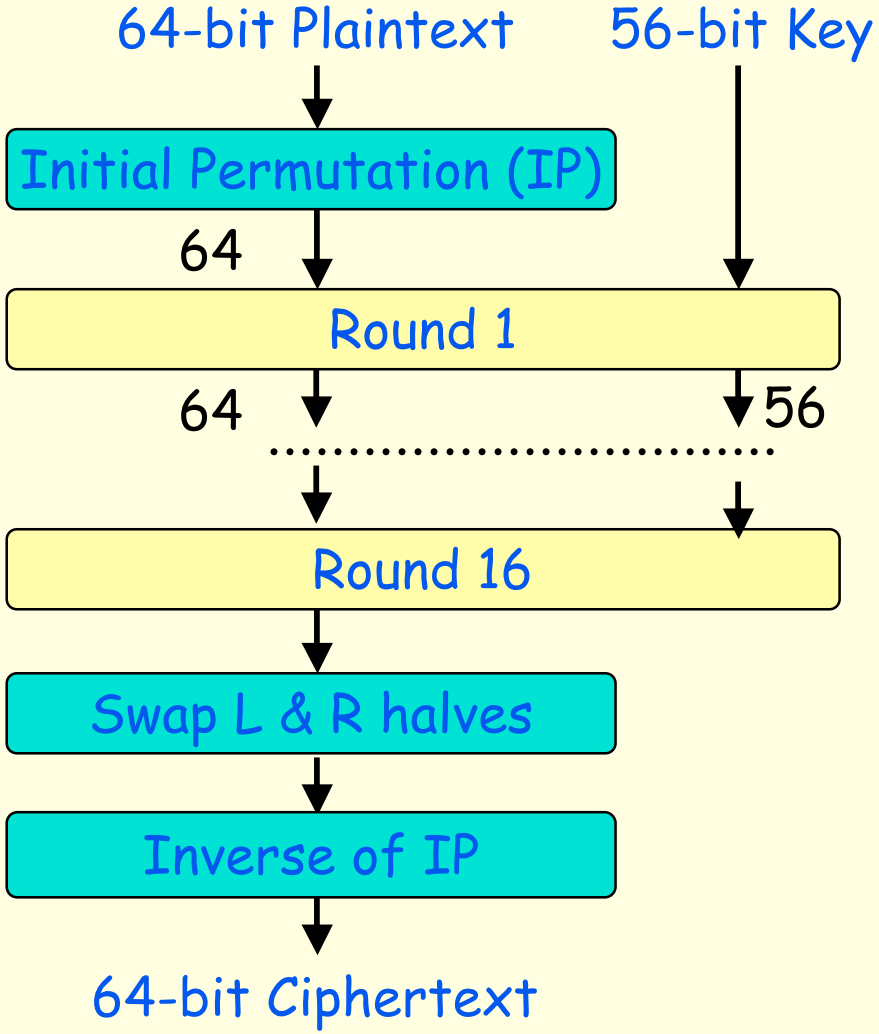


- DES is an example of a BLOCK CIPHER (but can also be operated as a STREAM CIPHER)

Desired Design Criteria:

- Ciphertext should depend on the plaintext and key in a complicated and involved way (CONFUSION)
- Each bit of ciphertext should depend on all bits of plaintext and all bits of the key (DIFFUSION)
- **AVALANCHE EFFECT**
  Small changes to input cause massive variation in output. In DES flipping 1 bit of the key or 1 bit of a 64-bit input block will flip 50% of the output block's bits

Structure of DES



ENCRYPTION

- Each block is subjected to 16 rounds of substitutions and permutations (transpositions).
- Permutations act to 'diffuse' data, substitutions act to 'confuse' data (SHANNON's terminology)
- Each round uses 48 bits from key called the subkey.
- Initial and final permutation appear to be redundant.

DECRYPTION

- Same process as encryption but with subkeys applied in reverse order

Feistel Cipher: a cipher design pattern

Encryption

- N rounds
- Plaintext = (L_0, R_0)
- For $1 <= i <= n$

  $L_i = R_{i-1}$

  $R_i = L_{i-1}$ **xor** $f(R_{i-1}, K_i)$
- Subkeys K_i derived from key K
- Ciphertext = (R_n, L_n)

Note: swapped halves

Decryption

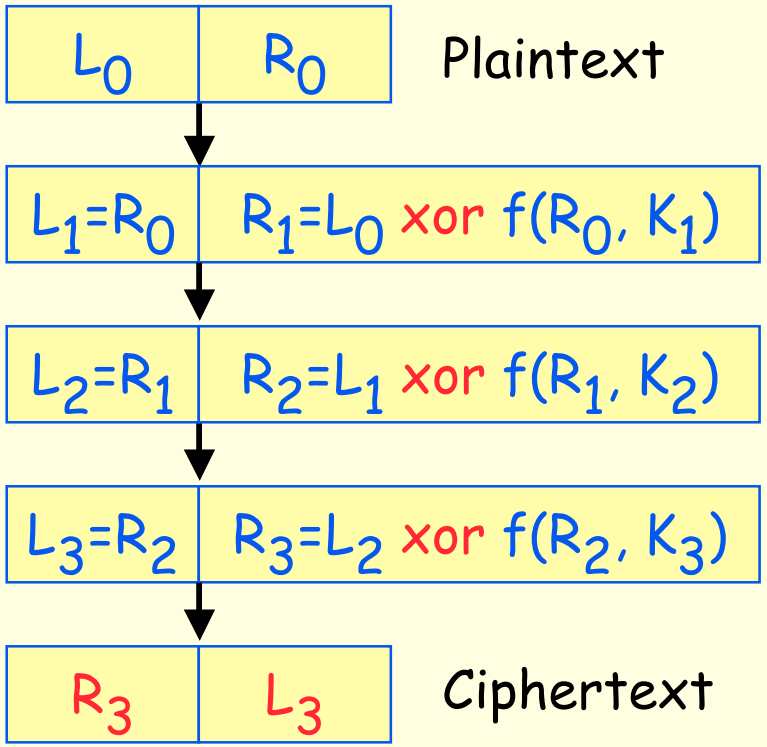
- As Encryption above, but subkeys applied in reverse order:

  N, N-1, N-2, ...

- Block size: Large block size better. 128-bit or 256-bits blocks best
- Key size: *These days* at least 128 bits, more better, e.g. 192 or 256 bits
- Number of rounds: Typically at least 16 rounds needed
- Round function f and subkey generation: Designed to make cryptanalysis difficult
- Round function f: typically built from transpositions, substitutions, modular arithmetic, etc.

Feistel Cipher

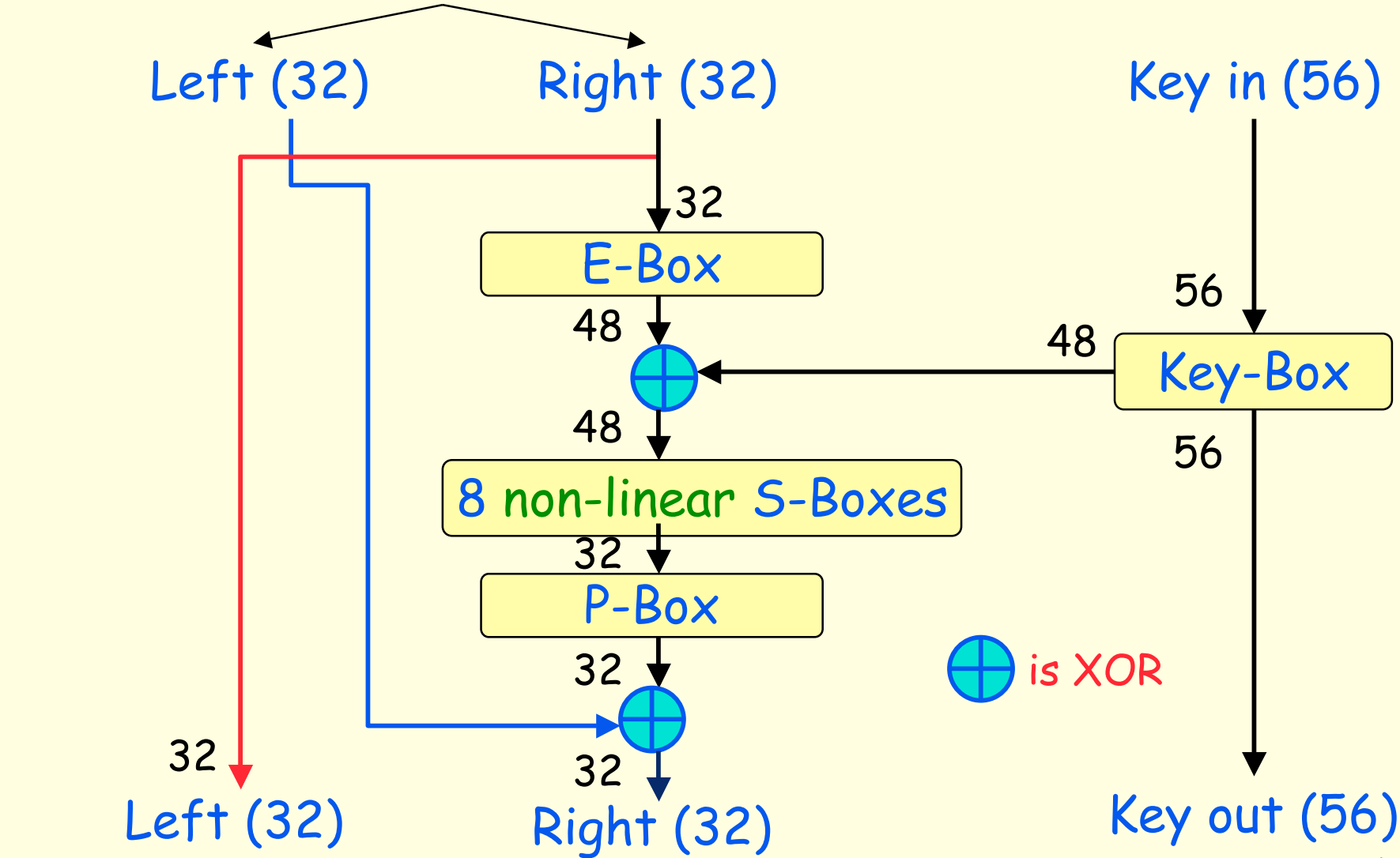
Feistel Cipher for 3 rounds



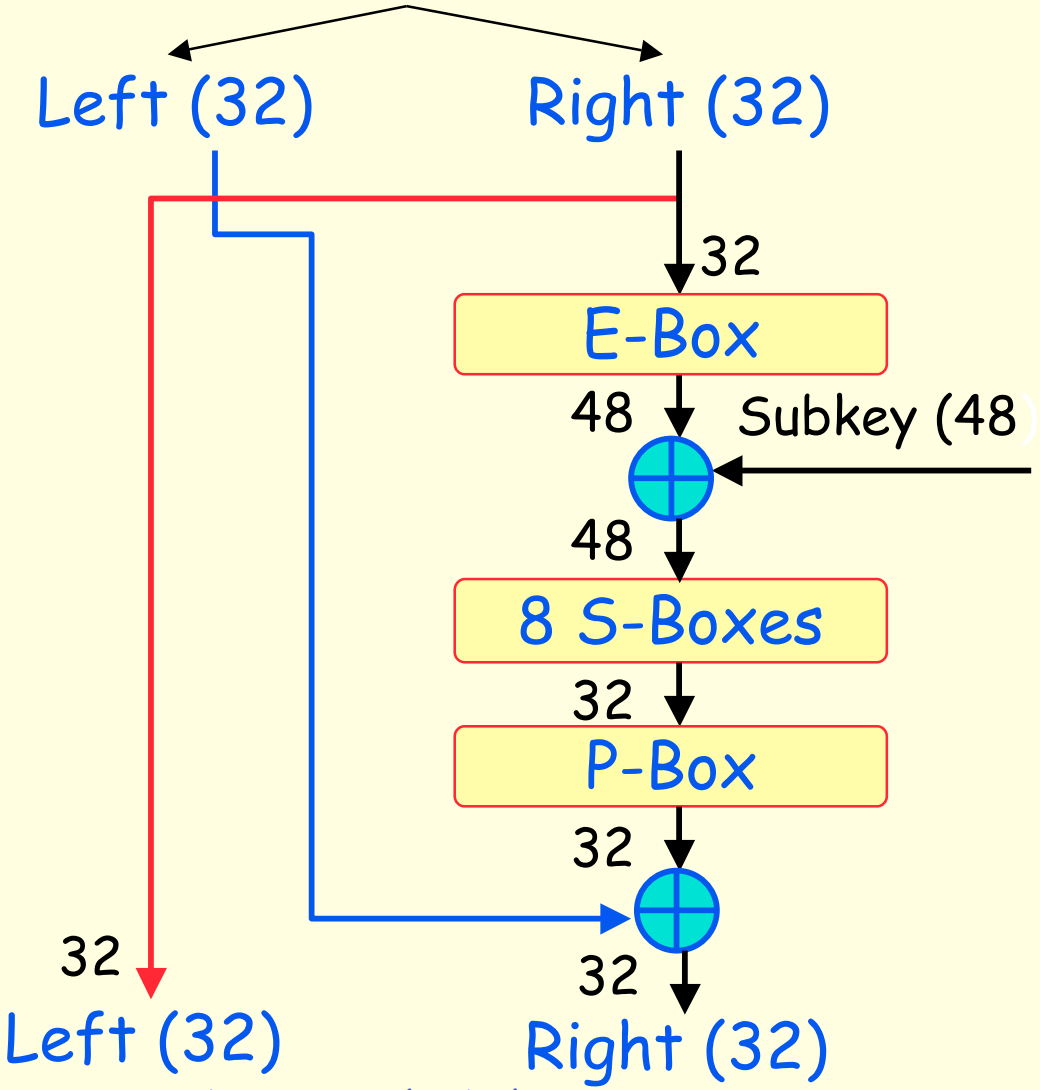
This example should also make clear why Decryption needs to supply

- key K3 in the first round,
- key K2 in the second round, and
- key K1 in the third round

A Round of DES



A Round of DES



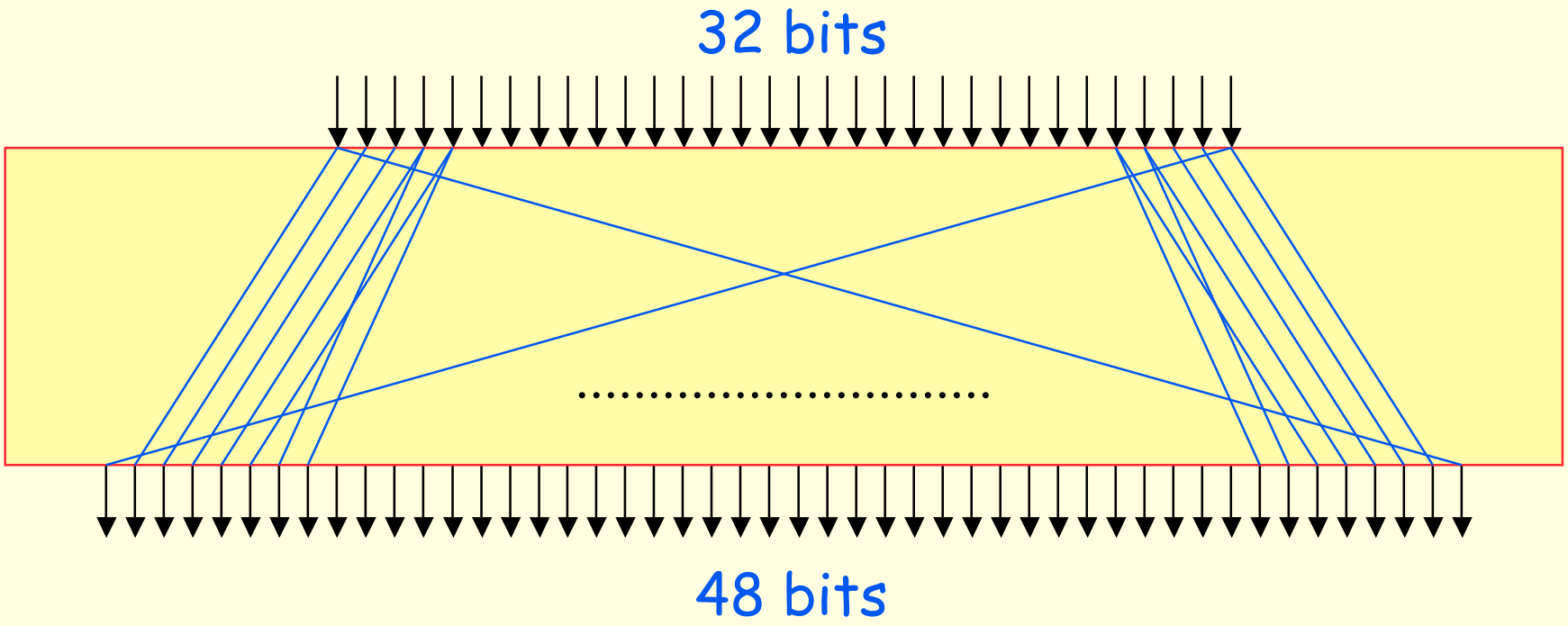
A Round

$Left_i = Right_{i-1}$

$Right_i = Left_{i-1}$ xor f_i

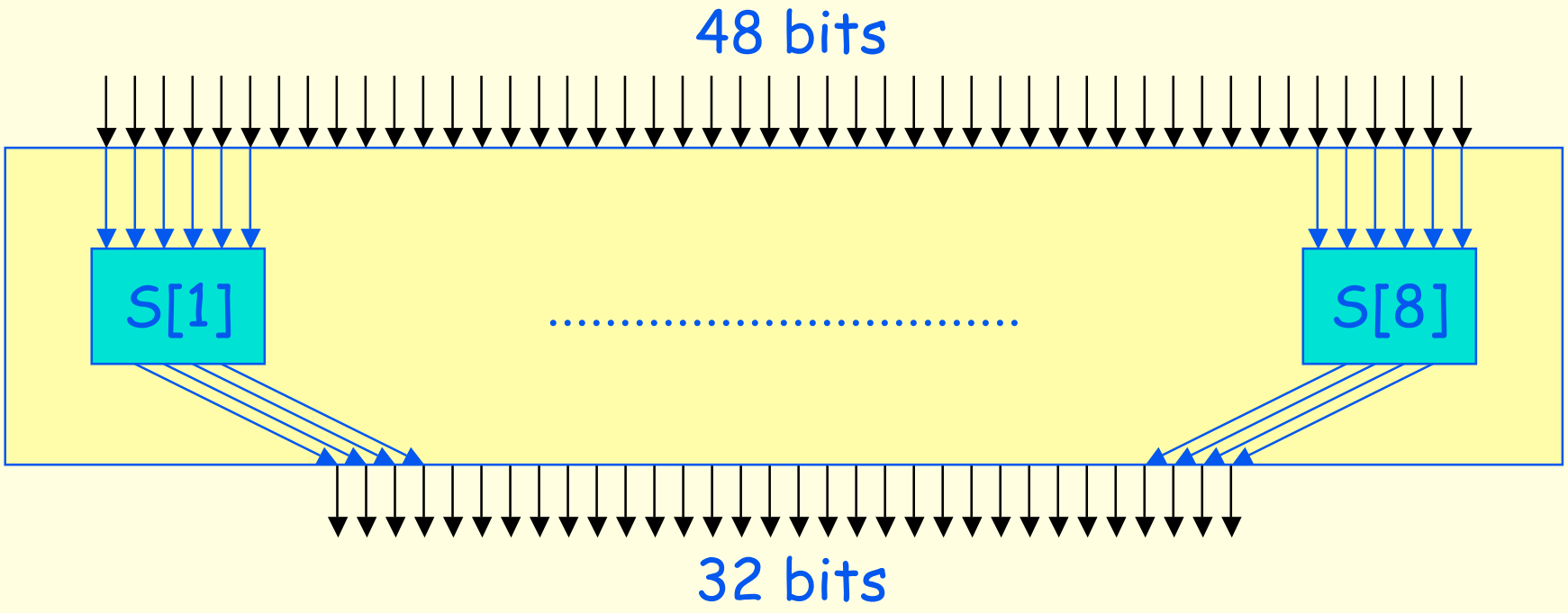
$f_i = P \circ S \circ (E(Right_{i-1})$ xor $Subkey_i)$

E-Box



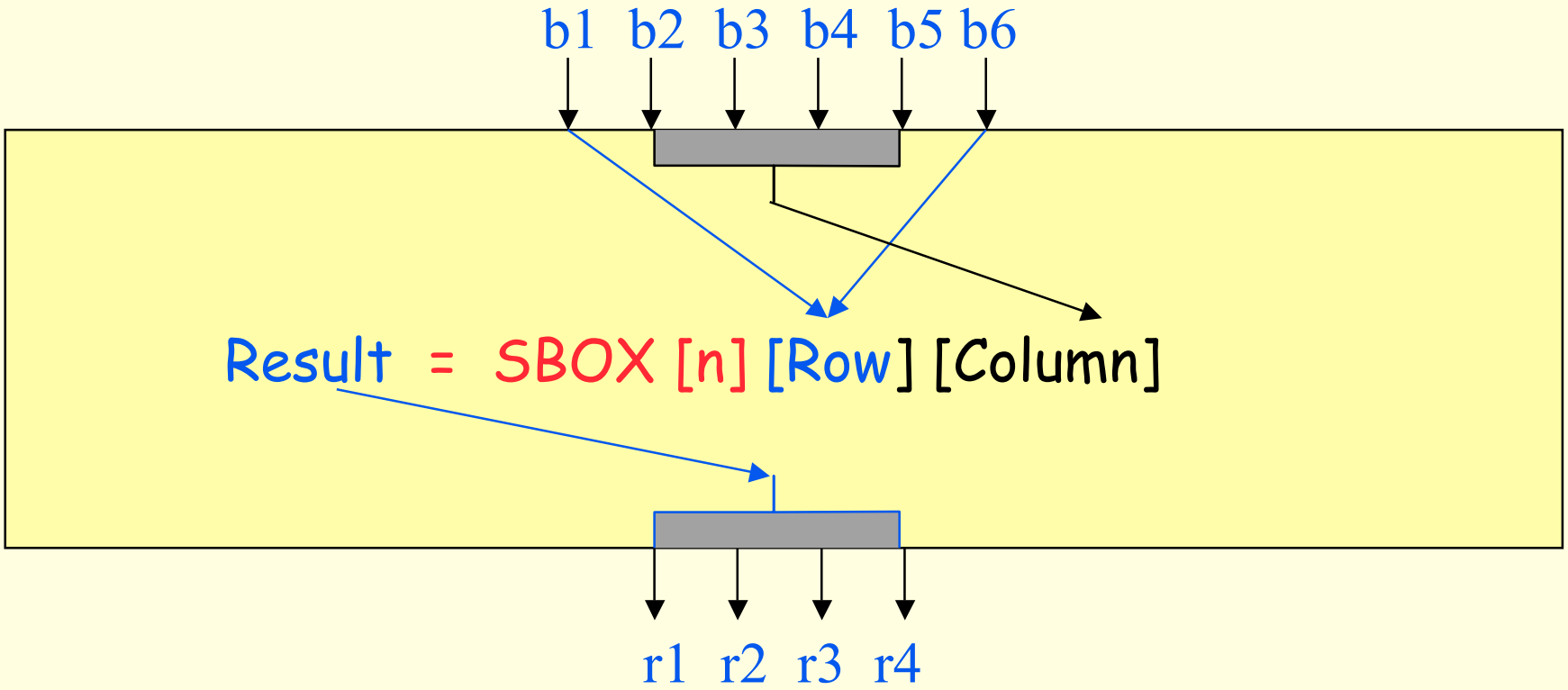
- E box expands & permutates (from 32-bits to 48 bits). Changes order as well as repeating certain bits (Helps with avalanche effect).

S-Boxes



- Each S-box takes 6-bits of input and produces 4-bits of output.
- S-Boxes give DES it's security. Other boxes are linear and easier to analyse. S-Boxes are non-linear and much harder to analyse.

S-Box [n]

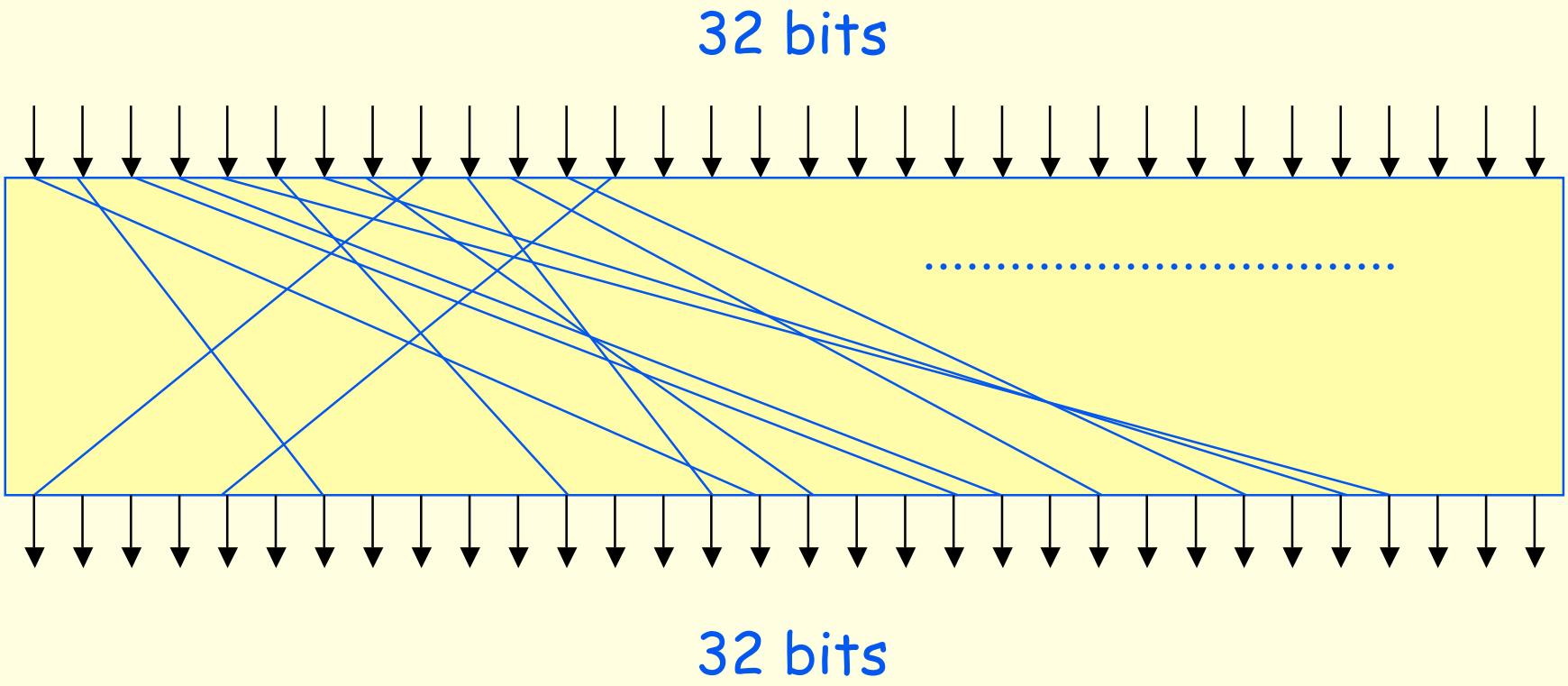


- Each S-box has its own substitution table. Outer 2 bits select row, middle 4 bits select column of substitution table. Entry gives new 4 bit value.

Substitution table for S-Box S5

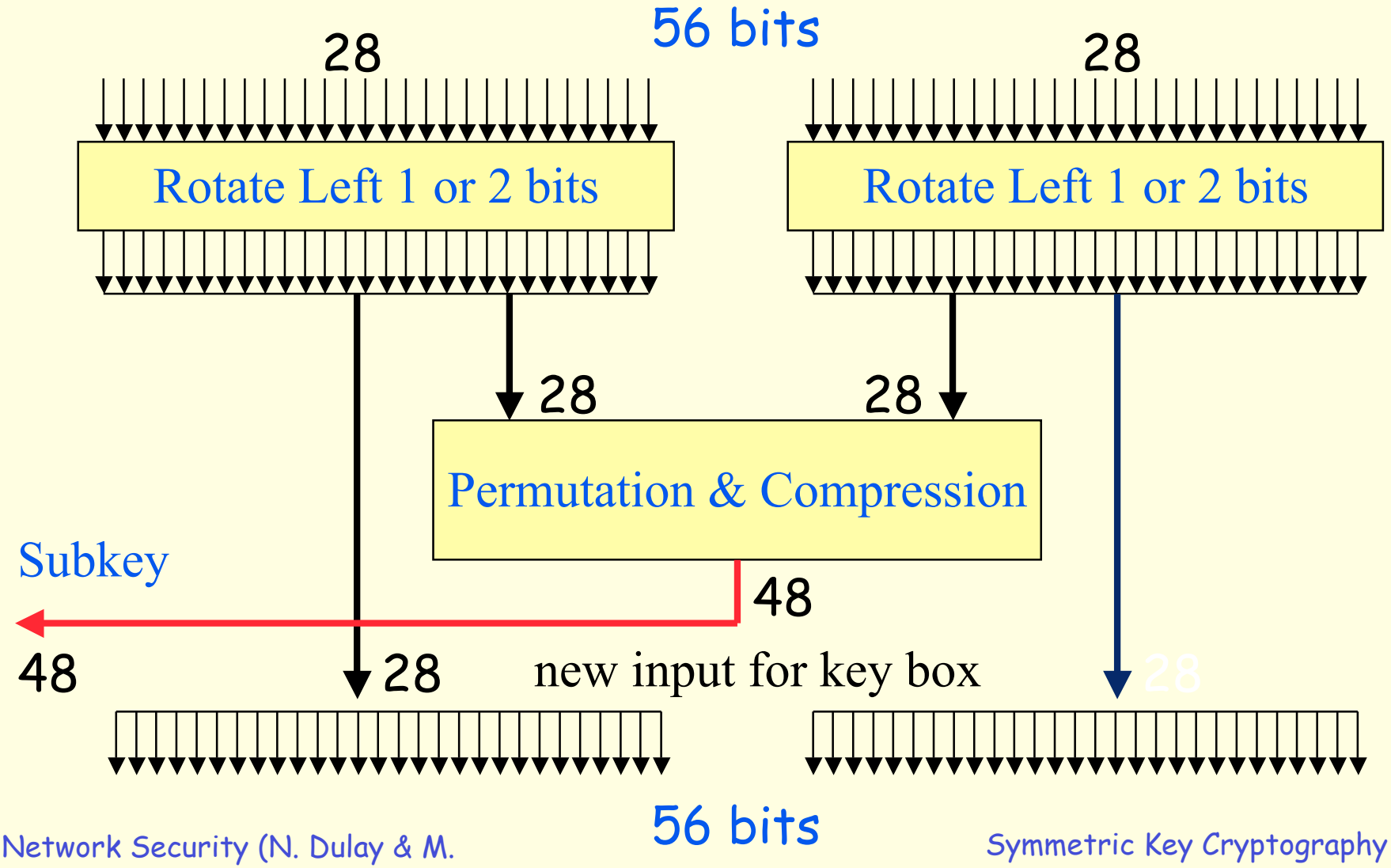
http://en.wikipedia.org/wiki/S-box

P-Box



- P-Box is just a mathematical permutation.

Key Box: determines subkeys



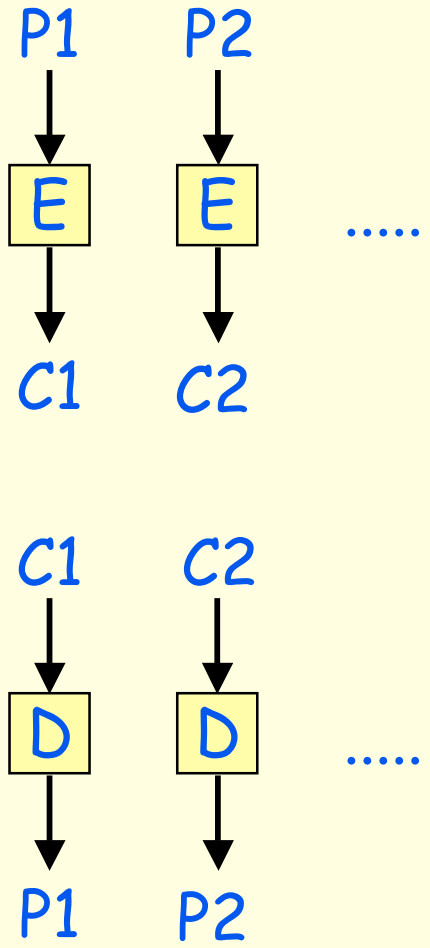
DES block cipher: modes of usage

- So far, we saw how DES encrypts one 64-bit block
- How to encrypt general plain-text messages?
- Cut up plain-text into 64-bit chunks.
- Practical question: What about plain-text that is not a multiple of 64 bits?
- Answer: add bits (but in a way proscribed by the DES standard!) to make plain-text fit.

- Encrypting each block in isolation may not be desirable.
- We now study different modes of using DES to encrypt sequences of 64-bit blocks.
- Practical aspect: if errors occur in encrypting one block, what other blocks will be affected by this error?

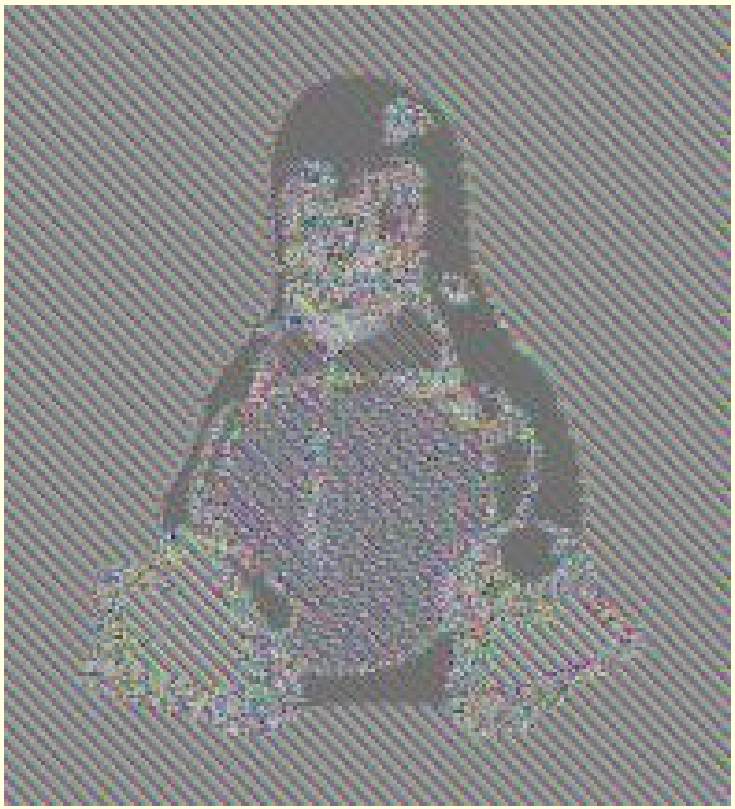
ECB - Electronic CodeBook

- $C_n = E(K, P_n)$
- Simplest operation mode of DES, no feedback between blocks
- Used for short values (e.g. keys) to prevent opponent building a code book.
- Identical blocks of plaintext -> identical ciphertext block
- ECB easily parallelizable.
  No processing before a block is seen, though.
- What if 1-bit of Ci is changed?
- What if 1-bit is inserted/deleted into Ci?



Wiki example of ECB Mode

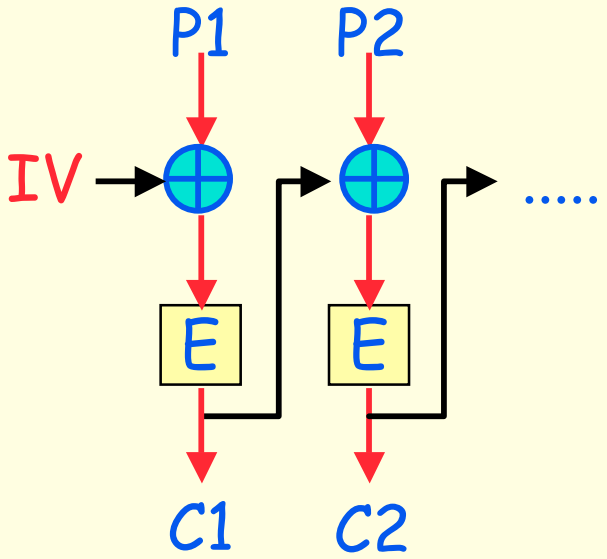
Plain-Text



Cipher-Text

CBC - Cipher Block Chaining

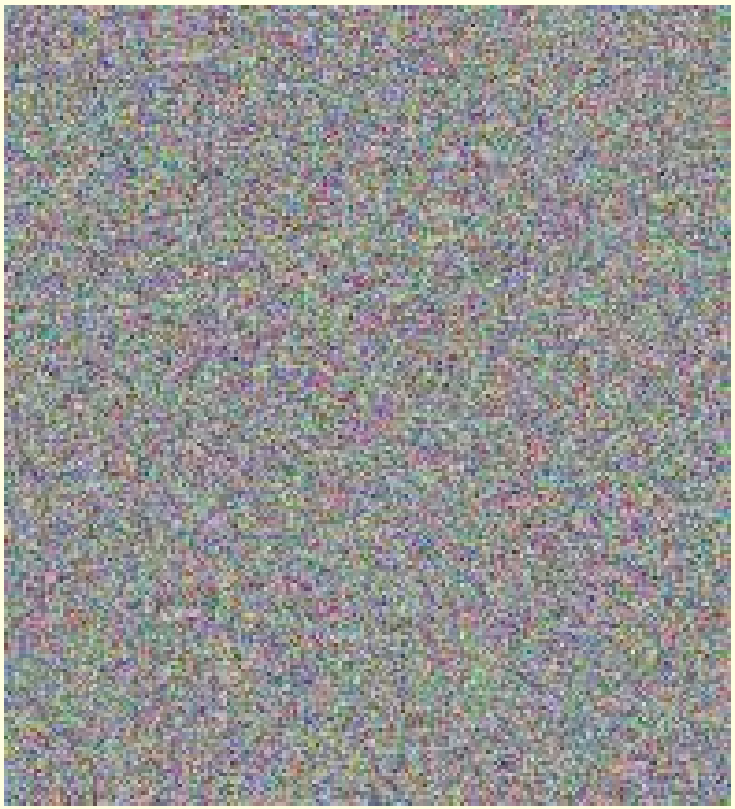
- $C_n = E(K, P_n \textbf{ xor } C_{n-1})$
  $C_0 = IV$ = random value
  called an *initialization vector*
- Adds feedback to encryption of next block. Most used mode. Conceals any repeated patterns in plaintext.
- Choose C_0=Initialisation Vector (IV) randomly. So Ciphertext has one extra block at start. Ensures P will generate different C at each encryption time.
  (Alternatively generate IV by encrypting nonce, include nonce in message.)
- Need to pad last P block if shorter than 64-bits. How to do this best?



⊕ is XOR

Wiki example of non-ECB mode

Plain-Text



Cipher-Text

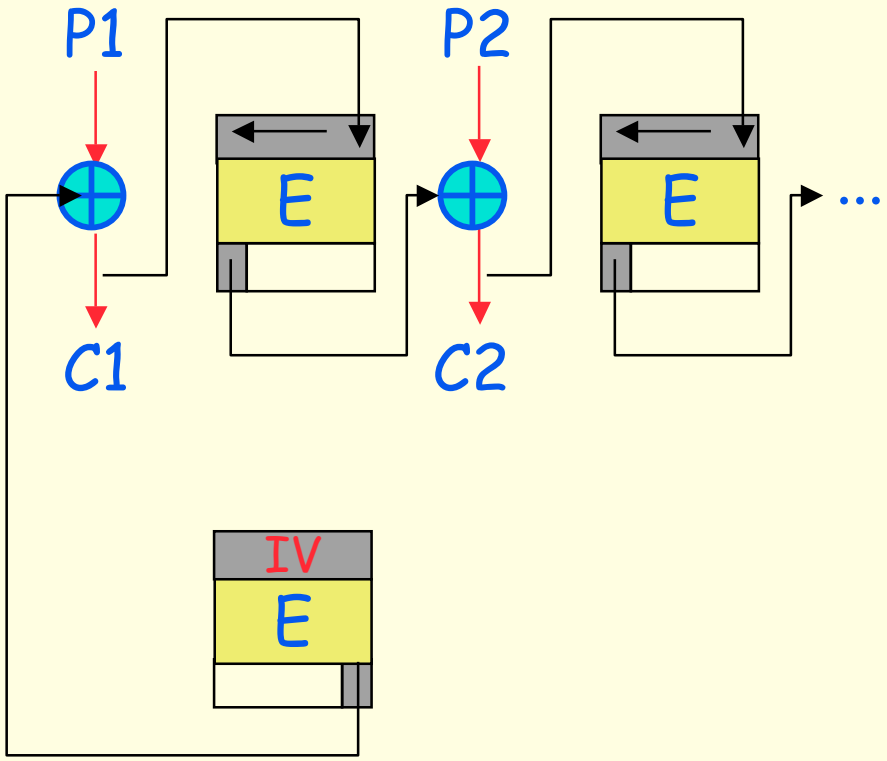
CFB - Cipher Feedback (Stream Cipher)

- $C_n = P_n$ **xor** $E(K, C_{n-1})$
- Self-Synchronising Stream Cipher.
- If P_n is less than 64-bits, e.g if 8 bits, use top 8 bits of C_n, and shift into bottom 8 bits of input to E (input is a 64-bit shift register). Only need to send 8-bit values in this case.

  1 bit → CFB1

  8-bits → CFB8

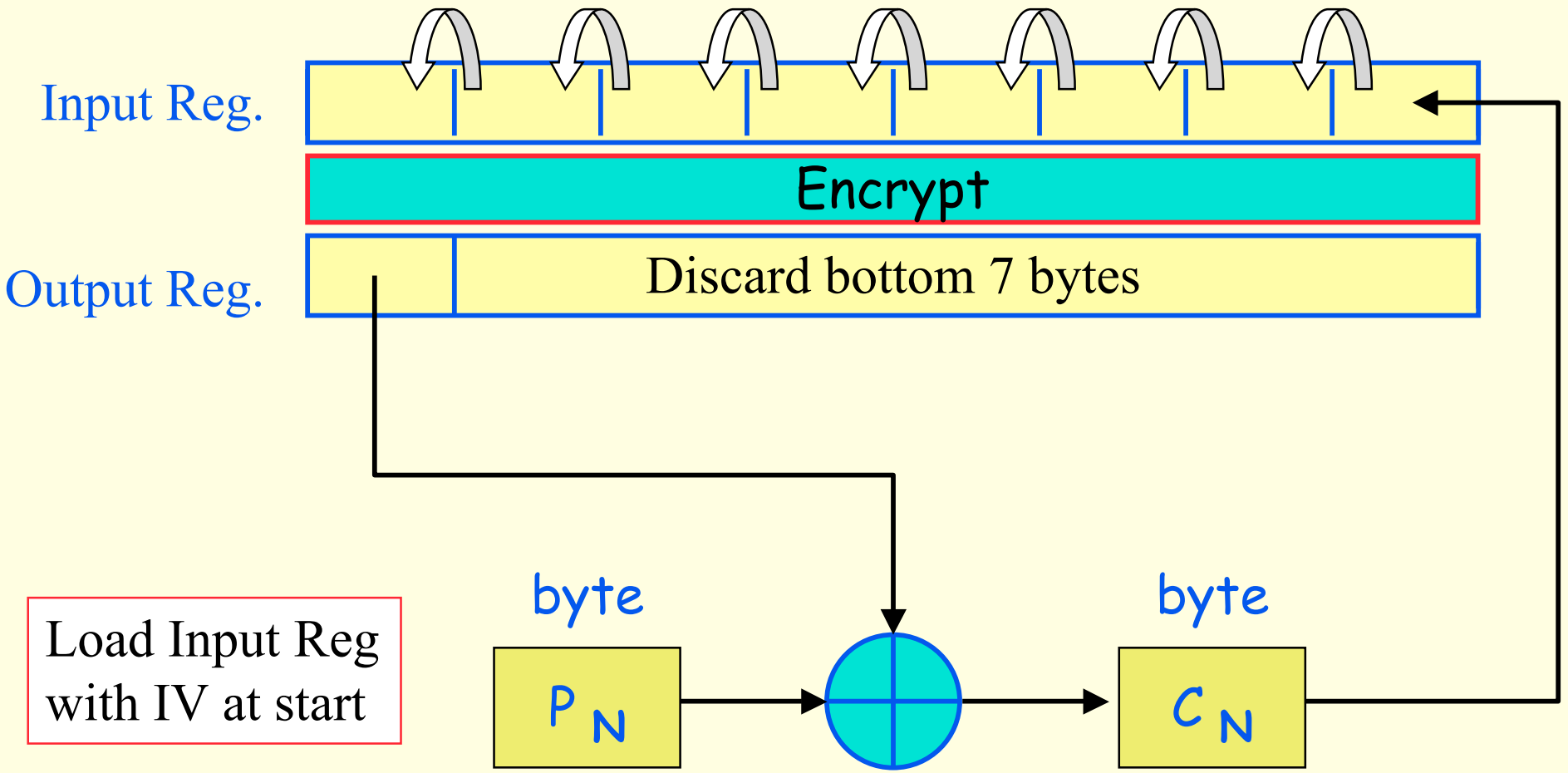
  64-bits → CFB64



IV: least significant bits

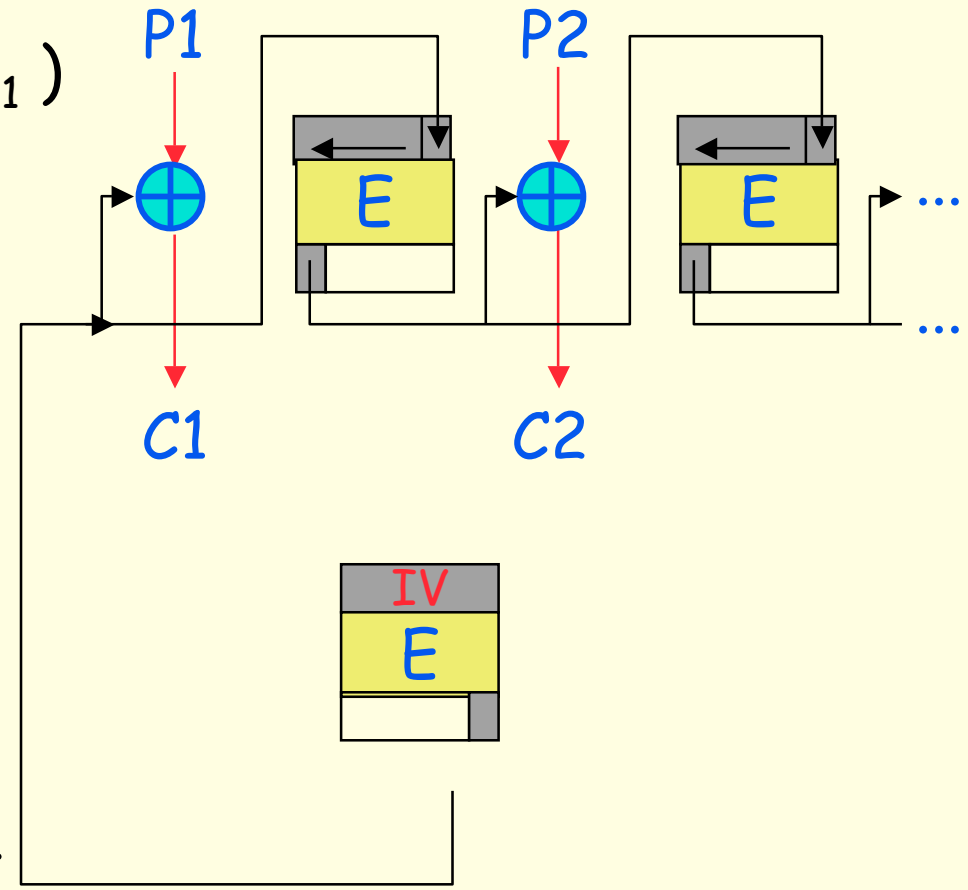
Output: most significant bits

CFB - Shift Register (Sending)

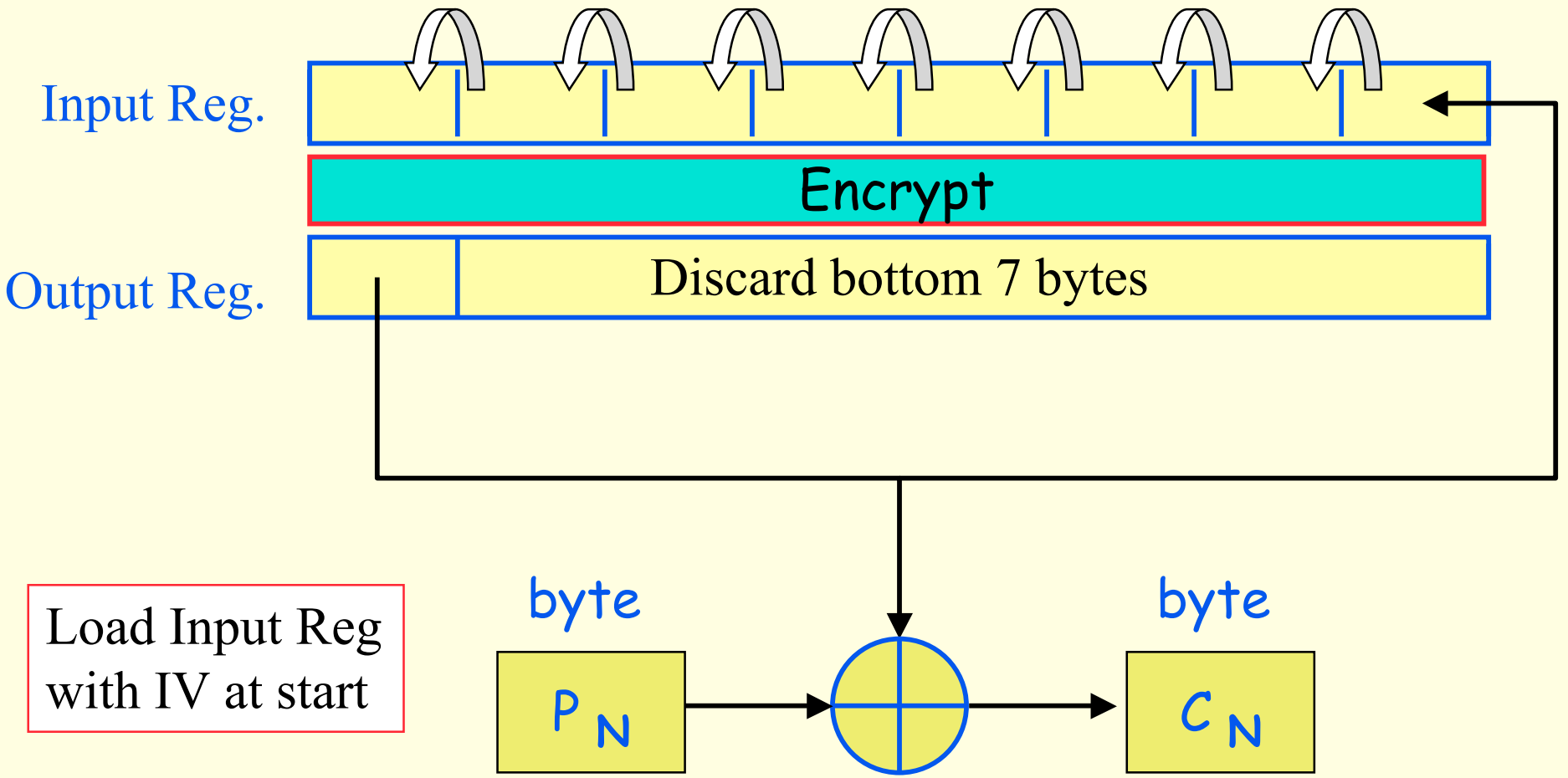


OFB - Output Feedback (Stream Cipher)

- $C_n = P_n$ **xor** X_n, $X_n = E(K, X_{n-1})$
- $X_0 = IV$ = randomvalue
- Synchronous Stream Cipher
- Like CFB, OFB is used for smaller bit-groups, e.g. bytes
- Good for nosier channels.
- Keystream can be pre-computed offline
- 1 bit → OFB1, 8 bits → OFB8, etc...

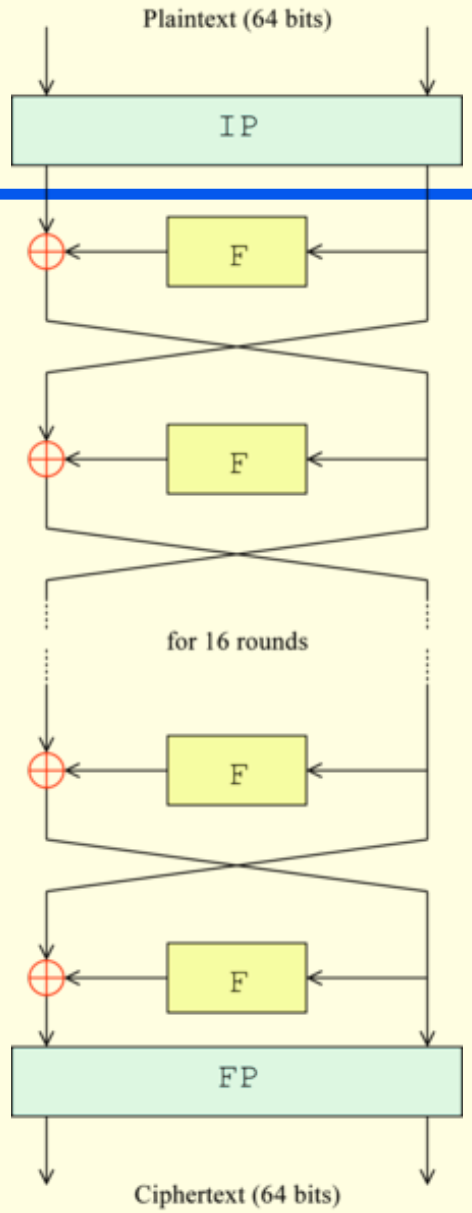


OFB - Shift Register (Sending)



Security of DES

- Design criteria (particularly of S-Boxes) not revealed until 1994
- No known trapdoors. No proof of non-existence either
- Oddity: If both plaintext and key are complemented so is resulting ciphertext.
- DES has 4 weak keys & 6 pairs of semi-weak keys which should not be used.



Security of DES

BRUTE FORCE ATTACK

- 2^{56} keys but brute force attacks are now becoming feasible
- In 1993 Michael Wiener showed that it was possible to cheaply build hardware that undertook a known-plaintext attack:
  in 3.5 hours for $1 million
  in 21 mins for $10 million
  in 35 hours for $100,000
- Intelligence agencies and those with the financial muscle most probably have such hardware.
- See link "How to break DES" on course home page: www.cryptography.com/des/

- **Differential Cryptanalysis**
  Exploits how small changes in plaintext affect ciphertext.

  For DES, requires 2^47 chosen plaintexts for 16 rounds ! Can break 8-round DES in seconds.

- **Linear Cryptanalysis**
  Approximate effect of encryption (notably S-boxes) with linear functions.

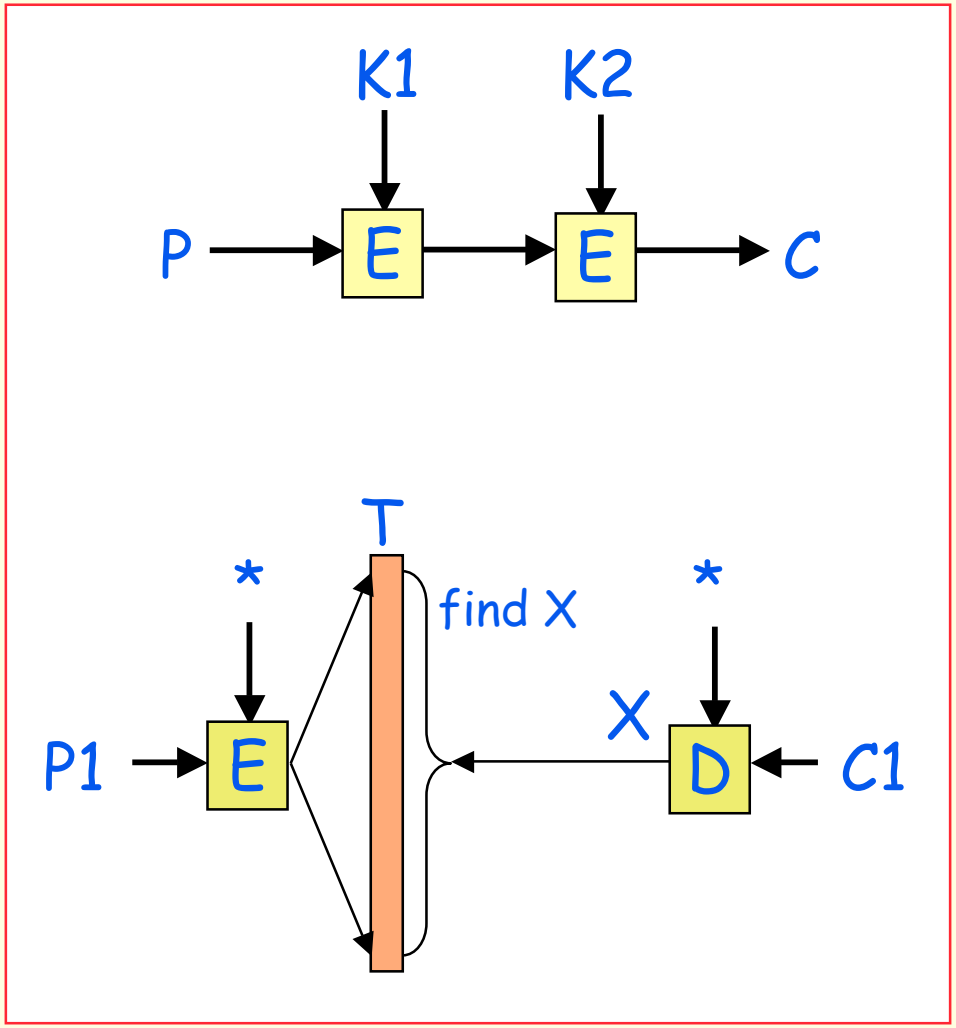
  For DES, requires 2^43 known plaintexts for 16 rounds !

Double DES (Multiple Encryption)

- Encrypt twice with two keys

MEET-IN-THE MIDDLE ATTACK

- Known plaintext attack (i.e. have crib P1 & C1)
- For all K1 encrypt P1: list all results in Table T
- For each K2 decrypt C1 -> X. If X in T, check K1 & K2 with new crib (P2, C2). If okay then keys found.
- Reduces 2^{112} to 2^{56} for Double DES, but T is huge!



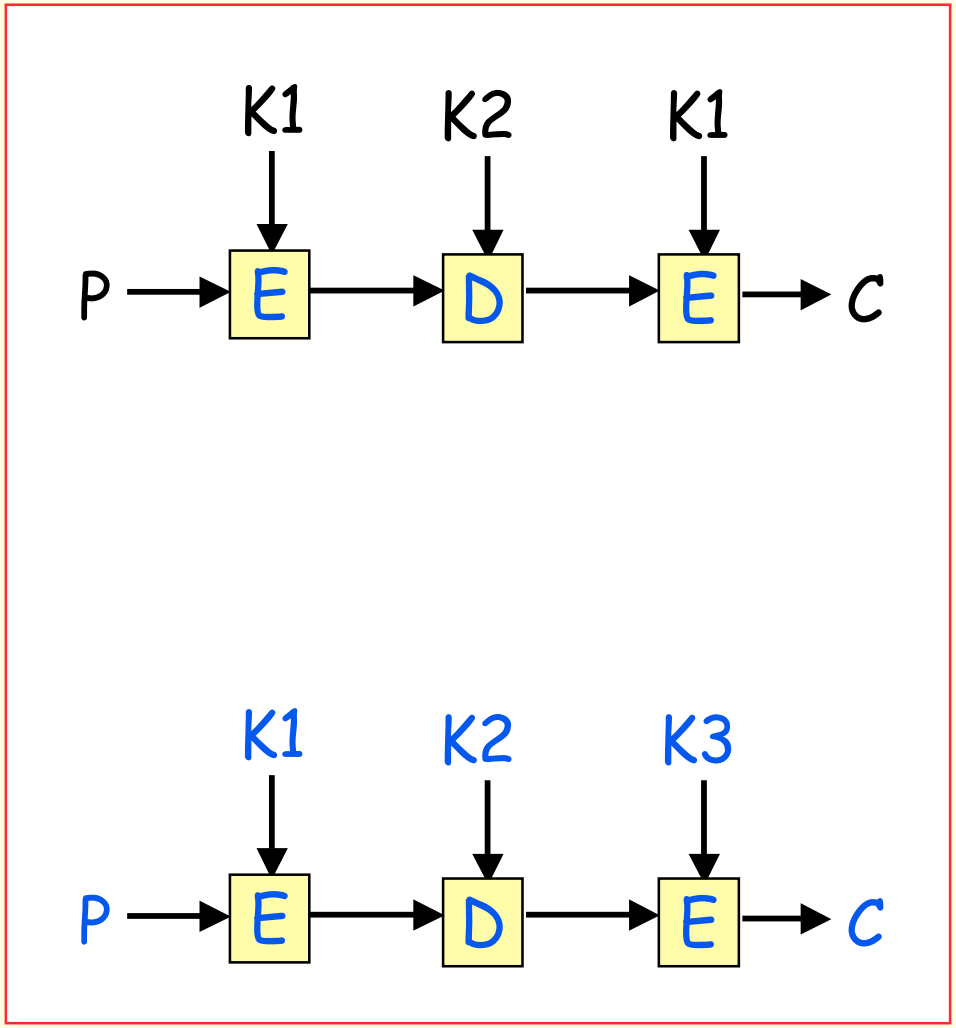
Triple DES (part of DES standard)

TRIPLE DES WITH 2 KEYS (EDE2)

- 3 keys considered unnecessary
- Cost of 2 key attack is thus 2^{112}
- 2nd Stage is decryption because if K2=K1 we gain backward compatibility with Single DES
- Available in PEM (Privacy Enhanced Mail), PGP, and others.

TRIPLE DES WITH 3 KEYS (EDE3)

- Preferred by some
- 168-bit key length

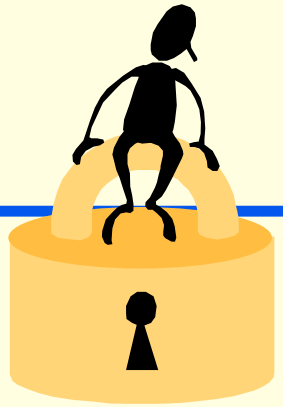


IDEA

- Lai and Massey, ETH Zurich, 1991
- International Data Encryption Algorithm
- Patented but blanket permission for non-commercial use
- 64-bit block cipher, 128-bit key
- Uses XOR, Modular + and * in each round (8 rounds)
- Considered strong, but 6-round attack requires 2^{64} known plaintexts and $2^{126.8}$ operations
- Used in PGP

RC5

- Designed by Ron Rivest (Ron's Code 5) of RSA fame in 1995
- Patented by RSA Inc
- Variable block size (32, 64, 128)
- Variable key size (0 to 2048)
- Variable no. of rounds (0 to 255)
- Uses XOR, modular + and circular left rotations.
- 12-round version subject to differential attack, needs 2^{44} plain-texts



The Advanced Encryption Standard

Michael Huth

M.Huth@doc.ic.ac.uk

www.doc.ic.ac.uk/~mrh/430/

Introduction

- In January 1997 US NIST solicited proposals for new Advanced Encryption Standard (AES) to replace DES as a federal standard. Approved for classified US governmental documents by US NSA
- Five algorithms shortlisted. Winner: Rijndael (by Joan **Rij**men & Vincent **Dae**men from Belgium). AES is minor variant of Rijndael
- Web Page: csrc.nist.gov/encryption/aes
- US FIPS PUB197, Nov 2001

- Resistant to Known Attacks, at least in "full" version
- Very fast. Parallel Design.
- Blocksize: 128 bits
  Keysizes (Rounds): 128 (10), 192 (12) & 256 (14) bits.
- Simple operations over bytes and 32-bit words.
- Bytes/words -> polynomials
- Implementations for wide range of processors incl. smartcards.
- Encryption # Decryption

Byte - $b_7b_6b_5b_4b_3b_2b_1b_0$

- Bytes represent **finite field elements in GF(2^8)**, GF means "Galois Field"
- Correspond to a 8 term polynomial, with 0 or 1 coefficients.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

Example:

$x^6 + x^5 + x^3 + x^2 + 1$	polynomial
{0110 1101}	binary
6D	hex

Byte Addition in GF(2^8)

- To add 2 finite fields elements in GF(2^8) we add coefficients of corresponding powers modulo 2
- In binary: **xor** ($\oplus$) the bytes

Example:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = (x^7 + x^6 + x^4 + x^2)$$

$\{0101\ 0111\} \oplus \{1000\ 0011\} = \{1101\ 0100\}$ binary

$57 \oplus 83 = D4$ hex

Byte Multiplication in GF(2^8)

- To multiply (denoted by •) 2 finite fields elements in GF(2^8) we multiply the polynomials modulo an **irreducible polynomial** of degree 8 (i.e. ensures result is less than degree 8).
- **Irreducible** if only divisors are 1 and itself. Can find multiplicative inverse using Extended Euclidean algorithm (with works for any "integral domains", certain kinds of rings).
- For AES we use ($x^8 + x^4 + x^3 + x + 1$) as the irreducible polynomial, i.e. multiplication is:

$$c(x) = a(x) \bullet b(x) \mod m(x)$$
$$\text{where } m(x) = (x^8 + x^4 + x^3 + x + 1)$$

Multiplication • is the basis for *non-linear behaviour* of AES: it's easy to understand over polynomials, but *hard to predict as operation on bytes*.

Byte Multiplication in GF(2^8) - Example

$$(x^7 + x^6 + 1) \bullet (x^2 + x) = (x^9 + x^8 + x^2) + (x^8 + x^7 + x)$$
$$= x^9 + x^7 + x^2 + x$$

$x^8 + x^4 + x^3 + x + 1$ | $x^9 + x^7 + x^2 + x$

$x^9 + x^5 + x^4 + x^2 + x$

$x^7 + x^5 + x^4$

Result = $x^7 + x^5 + x^4$

xtime - multiplication by x i.e. {02}

- If we multiply a byte by x we have

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

- If b_7=0, then the result is okay, otherwise we need to subtract m(x). This is known as the **xtime** operation in AES: **<<1** is shift left one

xtime (**byte** p) = (p << 1) ⊕ (**if** p < 80 **then** 00 **else** 1B)

- We can use **xtime** repeatedly to multiply by higher powers

AE • 02 = **xtime** AE = (AE << 1) ⊕ 1B
= {1010 1110} << 1 ⊕ {0001 1011} = {0101 1100} ⊕ {0001 1011}
= {0100 0111} = 47 i.e. $x^6 + x^2 + x + 1$

Word

- Word = 32-bits = 4 bytes.
- Words corresponds to 4 term polynomials, where **coefficients are finite field elements in GF(2^8)**, i.e. coefficients are bytes

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

Addition of two (word) polynomials corresponds to "adding" the coefficients (i.e. xor-ing the words)

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

Word Multiplication

- We multiply word-polynomials modulo a **polynomial** of degree 4 (i.e. to ensure result is less than degree 4).
- For AES we use $(x^4 + 1)$ as the polynomial. Note: this polynomial is not irreducible. However in AES we only ever multiply word-polynomials by the fixed word polynomial:
  $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ which does have an inverse
  $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$

Modular product $d(x) = a(x) \otimes b(x) = a(x) \bullet b(x) \bmod (x^4 + 1)$

$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0$ where

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$
$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$$
$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$$
$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$$

Encrypt Block (Cipher) // simplified

```
encrypt (plaintext, roundkey)
    state = plaintext       // note plaintext is 1-dim., state 2-dim.
    state = AddRoundKey (state, roundkey[0])
    for round = 1 to ROUNDS
          state = SubBytes (state)
          state = ShiftRows (state)
          if round < ROUNDS then state = MixColumns (state)
          state = AddRoundKey (state, roundkey[round])
    end
    return state     // convert to 1-dim. and return as ciphertext
```

State

- State is a 4 by 4 array of bytes, initialised (col-by-col) with the 16-byte plaintext block (see below)
- Final value of state is returned as ciphertext

State	0	1	2	3
0	in[0]	in[4]	in[8]	in[12]
1	in[1]	in[5]	in[9]	in[13]
2	in[2]	in[6]	in[10]	in[14]
3	in[3]	in[7]	in[11]	in[15]

- Bytes of State correspond to finite field elements in $GF(2^8)$
- Columns of State correspond to WORDS, i.e. 4-term polynomials with finite field elements in $GF(2^8)$, as coefficients.

SubBytes Transformation

- Change each byte of State with corresponding byte from SBOX matrix:

  State [Row, Col] = SBOX [X, Y]
  where X = State[Row, Col] div 16, Y = State [Row, Col] mod 16
  For example if State [3,6]= 4F we would lookup SBOX[4,F]

- SBOX is 16x16 byte array (indexed by hex digits 0..F, 0..F) defined as follows:

  SBOX [X, Y] = AffineTransformation ($\{XY\}^{-1}$)
  For example: if $\{95\}^{-1}$ = 8A then
  SBOX[9,5] = AffineTransformation (8A) = 2A

- AffineTransformation is a function that performs a matrix multiplication followed by a vector addition. See Stallings or Huth for specifics of matrix and vector used in AES.

ShiftRows Transformation

- Cyclically rotate LEFT last 3 ROWS of state matrix by 1, 2 and 3 bytes resp.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

Rotate left 1 Byte

Rotate left 2 Bytes

Rotate left 3 Bytes

$$\begin{pmatrix} a & b & c & d \\ f & g & h & e \\ k & l & i & j \\ p & m & n & o \end{pmatrix}$$

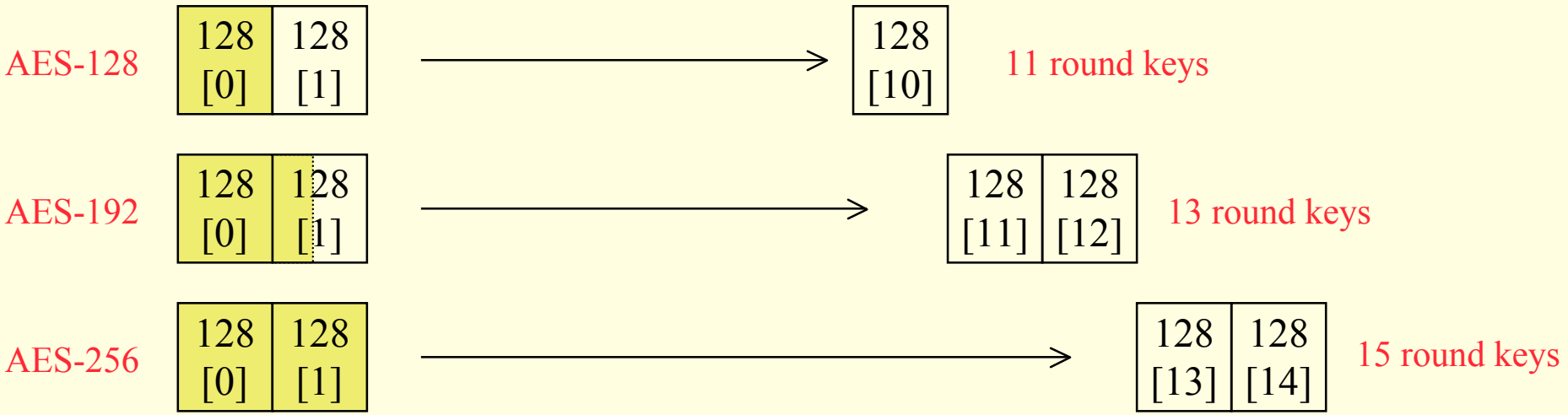
MixColumns Transformation

- Multiply each column by $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ **mod** $(x^4 + 1)$
  i.e. columns are word-polynomials
- This is equivalent to replacing the 4 bytes (m,n,p,q) in a column as follows:

$$\begin{pmatrix} m \\ n \\ p \\ q \end{pmatrix} \Rightarrow \begin{pmatrix} \{02\}\bullet m \oplus \{03\}\bullet n \oplus p \oplus q \\ m \oplus \{02\}\bullet n \oplus \{03\}\bullet p \oplus q \\ m \oplus n \oplus \{02\}\bullet p \oplus \{03\}\bullet q \\ \{03\}\bullet m \oplus n \oplus p \oplus \{02\}\bullet q \end{pmatrix}$$

AddRoundKey Transformation

- XOR round key with state.
- The cipher key (either 128/192/256 bits) is "expanded" into round keys (1 for each round, plus 1 for the initial AddRoundKey transformation). Note: each Round key is, say, 128-bit treated as a 2-dim. byte array. The cipher key words occupy the start of these round key words, the remaining ones are calculated from it.
- See Stallings or Huth for details of the key "expansion" function used.



Decrypt Block (Inverse Cipher) // simpl.

```
decrypt (ciphertext, roundkey)
    state = ciphertext       // note cipher is 1-dim., state 2-dim.
    state = AddRoundKey (state, roundkey[ROUNDS])
    for round = ROUNDS-1 to 0
          state = InvShiftRows (state) // ShiftRows inverse mode
          state = InvSubBytes (state)  // SubBytes inverse mode
          state = AddRoundKey (state, roundkey[round])
          if round > 0 then state = InvMixColumns (state)
    end
    return state     // convert to 1D and return as plaintext
```

Inverse Transformations

- **InvShiftRows** — Rotate Right last 3 rows of state
- **InvSubBytes** — Inverse SBOX uses inverse of AffineTranformation & then takes multiplicative inverse in $GF(2^8)$
- **InvMixColumns** — Multiply columns by inverse of a(x), i.e by $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$
- **AddKeyRound** — Is its own inverse!

- Encryption polynomial a(x) optimized for 8-bit processors
- Decrpytion polynomial $a^{-1}(x)$ not optimal for 8-bit processors
- Note: It is possible to write Decrypt (Inverse cipher) with the same sequence of transformations as Encrypt, with the transformations replaced by their Inverse ones. This uses the fact that (Inv)SubBytes and (Inv)ShiftRows commute (i.e. order can be swapped), and that (Inv)MixColumns is linear, i.e
  MixColumns(State xor RoundKey)=MixColumns (State) xor MixColumns(RoundKey)
- See Stallings or Huth for details

Implementation

8-bit Processors, e.g. Smartcards (typically 1Kbyte of code)

- ShiftRows and AddRoundKey -> Straightforward
- SubBytes requires a table of 256 bytes
- Above three transformations combined & executed serially for each byte
- MixColumns can be simplified to xor and xtime operations. InvMixColumns is much slower however due to large coefficents of $a^{-1}(x)$
- The Round keys can be expanded on-the-fly during each round.

32-bit Processors

- With straightforward algebraic manipulations, the four transformations in a round can be combined to produce a table-lookup implementation that requires four table lookups plus four xor's per column. Each table is 256 words.
- Most of the operations of the key expansion can be implemented by 32-bit xor's, plus use of the S-box and a cyclic byte rotation.

Wide Trail Strategy

- Resistance to Differential and Linear Cryptanalysis
- Each round has three distinct invertible layers of transformations
  - Linear Mixing layer - ShiftRows & MixColumns provide high diffusion
  - Non-Linear layer - Parallel S-Boxes provide optimal worst-case non-linear properties
  - Key addition layer - XOR of round key. Note: layers cannot be "peeled off" since key addition is always applied at beginning & end of cipher.

Security of AES

- Attacks aim to have less complexity than Brute Force
- Reduced round attacks:
  7 rounds for AES-128
  8 rounds for AES-192
  9 rounds for AES-256
- Algebraic Attacks
  AES can be expressed in equations (continued fractions) - huge number of terms however. Some claim to able to solve such equations with less complexity than brute force (e.g. XSL attack)

- Side Channel Attacks
  Most successful technique to date. Bernstein showed that delays in encryption-time due to cache-misses could be used to work out the AES key. Demonstrated against a remote server running OpenSSL's AES implementation. More recently Osvik et al. demonstrated memory timing attacks that can crack AES in milli-seconds! (.. given access to the encrypting host)

Problems with Symmetric Key Cryptography

SCALABILITY

- For full and separate communication between N people need N(N-1)/2 separate keys

KEY MANAGEMENT

- Key Distribution
- Key Storage & Backup
- Key Disposal
- Key Change

- READING
  Stallings - Chapters 3 and 5 (and 4)