# Network Security: Secret Key Cryptography

Henning Schulzrinne
Columbia University, New York
schulzrinne@cs.columbia.edu

Columbia University, Fall 2000



Last modified September 28, 2000

## Secret Key Cryptography

- fixed-size block, fixed-size key $\rightarrow$ block
- DES, IDEA
- message into blocks?

# Generic Block Encryption

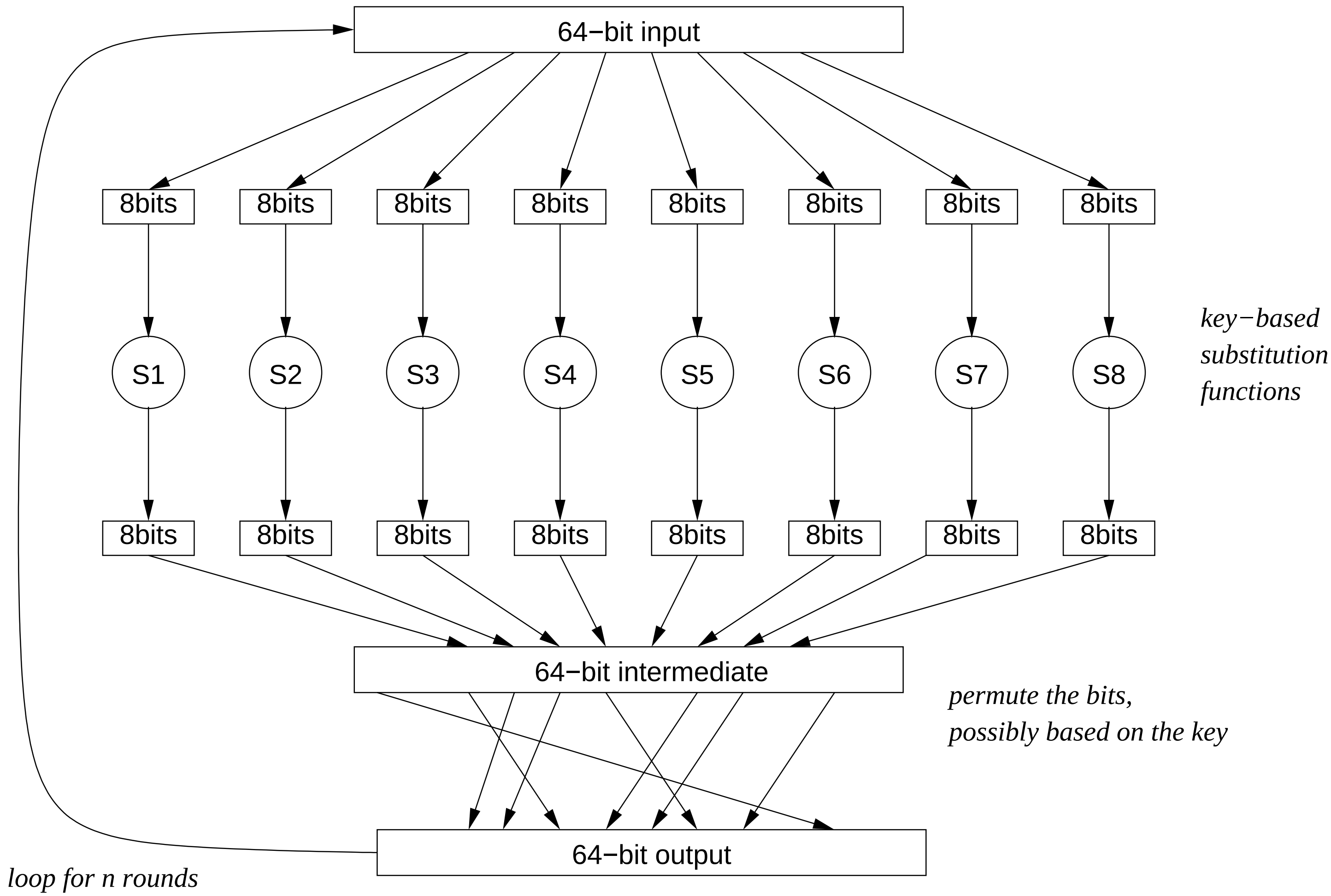
- convert block into another, *one-to-one*
- long enough to avoid known-plaintext attack
- 64 bit typical (nice for RISC!) ➠ $18 \cdot 10^{18}$ (peta)
- naive: $2^{64}$ input values, 64 bits each $\rightarrow 2^{70}$ bits
- output should look random
- plain, ciphertext: no correlation (half the same, half different)
- ➠ bit spreading

**substitution:** $2^k, k \ll 64$ values mapped ➠ $k \cdot 2^k$ bits

**permutation:** change bit position of each bit ➠ $k \log_2 k$ bits to specify

**round:** combination of substitution of chunks and permutation
do often enough so that a bit can affect every output bit – but no more

# Block Encryption



# Data Encryption Standard (DES)

- published in 1977 by National Bureau of Standards
- developed at IBM (“Lucifer”)
- 56-bit key, with parity bits
- 64-bit blocks
- easy in hardware, slow in software
- 50 MIPS: 300 kB/s
- 10.7 Mb/s on a 90 MHz Pentium in 32-bit protected mode
- grow 1 bit every 2 years

# Breaking DES

- brute force: check all keys ➠ 500,000 MIPS years
- easy if you have known plaintext
- have to know something about plaintext (ASCII, GIF, . . . )
- commercial DES chips not helpful: key loading time > decryption time
- easy to do with FPGA, without arousing suspicion
- easily defeated with repeated encryption

# DES Overview

- initial permutation
- 56-bit key $\rightarrow$ 16 48-bit per-round keys (different subset)
- 16 rounds: 64 bit input + 48-bit key $\rightarrow$ 64-bit output
- final permutation (inverse of initial)
- decryption: run backwards ➠ reverse key order

# Permutation

- just slow down software
- $i$th byte $\rightarrow$ $(9 - i)$th bits
- even-numbered bits into byte 1-4
- odd-numbered bits into byte 5-8
- no security value: if we can decrypt innards, we could decrypt DES

# DES: Generating Per-Round Keys

56-bit key $\rightarrow$ 16 48-bit keys $K_1, \ldots K_{16}$:

- bits 8, 16, ..., 64 are parity
- permutation
- split into 28-bit pieces $C_0, D_0$: $57, 49, \ldots$
- again, no security value
- rounds 1, 2, 9, 16: single-bit rotate left
- otherwise: two-bit rotate left
- permutation for left/right half of $K_i$
- discard a few bits ➠ 48-bit key in each round

# XOR Arithmetic

- $x \oplus x = 0$
- $x \oplus 0 = x$
- $x \oplus 1 = \bar{x}$

# DES Round

- mangler function can be non-reversible
  - $L_{n+1} = R_n$
  - $R_{n+1} = m(R_n, K_n) \oplus L_n$
- decryption
  - $R_n = L_{n+1}$
  - $L_n = m(R_n, K_n) \oplus R_{n+1}$

because ($\oplus L_n, R_{n+1}$): $R_{n+1} \oplus R_{n+1} \oplus L_n = m() \oplus L_n \oplus L_n \oplus R_{n+1}$

# DES Mangler Function

- $R(32), K(48) \oplus L_n \rightarrow R_{n+1}$
- expand from 32 to 48 bits: 4-bit chunks, borrow bits from neighbors
- 6-bit chunks: expanded $R \oplus K$
- 8 different S-boxes for each 6 bits of data
- **S box**: 6 bit (64 entries) into 4 bit (16) table: 4 each
- four separate 4x4 S-boxes, selected by outer 2 bits of 6-bit chunk
- afterwards, random permutation: P-box

# DES: Weak Keys

- 16 keys to avoid: $C_0, D_0$ 0. . . 0, 1. . . 1, 0101. . . , 1010. . .
- sequential key search ➠ avoid low-numbered keys
- 4 *weak keys* = $C_0, D_0 = 0 \ldots 0$ or $1 \ldots 1$ ➠ own inverses: $E_k(m) = D_k(m)$
- semi-weak keys: $E_{k_1}(m) = D_{k_2}(m)$

# IDEA

- International Data Encryption Algorithm
- ETH Zurich, 1991
- similar to DES: 64 bit blocks
- but 128-bit keys

# Primitive Operations

2 16-bit $\rightarrow$ 1 16-bit:

- $\oplus$
- $+ \bmod 2^{16}$
- $\otimes \bmod 2^{16}+1$:
  - reversible ➠ $\exists$ inverse $y$ of $x$, $\forall x \in [1, 2^{16}] a \otimes x \otimes y = a$
  - or $x \otimes y = 1$
  - example: $x = 2, y = 32769$ ➠ Euclid's algorithm
  - reason: $2^{16}+1$ is prime
  - treat 0 as encoding for $2^{16}$

# IDEA Key Expansion

- 128-bit key $\rightarrow$ 52 16-bit keys $K_1, \ldots, K_{52}$
- encryption, decryption: different keys
- key generation:
  - first chop off 16 bit chunks from 128 bit key ➠ eight 16-bit keys
  - start at bit 25, chop again ➠ eight 16-bit keys
  - shift 25 bits and repeat

# IDEA: One Round

- 17 rounds, even and odd
- 64 bit input $\rightarrow$ 4 16-bit inputs: $X_a, X_b, X_c, X_d$
- operations $\rightarrow$ output $X'_a, X'_b, X'_c, X'_d$
- odd rounds use $4K_i : K_a, K_b, K_c, K_d$
- even rounds use $2K_i : K_e, K_f$

# IDEA: Odd Round

- $X'_a = X_a \otimes K_a$
- $X'_d = X_d \otimes K_d$
- $X'_c = X_b + K_b$
- $X'_b = X_c + K_c$

reverse with inverses of $K_i$:
$X'_a \otimes K'_a = X_a \otimes K_a \otimes K'_a$

# IDEA: Even Round

mangler: $Y_{\text{out}}, Z_{\text{out}} = f(Y_{\text{in}}, Z_{\text{in}}, K_e, K_f)$

1.

$$\begin{aligned} Y_{\text{in}} &= X_a \oplus X_b \\ Z_{\text{in}} &= X_c \oplus X_d \end{aligned}$$

2.

$$\begin{aligned} Y_{\text{out}} &= ((K_e \otimes Y_{\text{in}} + Z_{\text{in}}) \otimes K_f \\ Z_{\text{out}} &= K_e \otimes Y_{\text{in}} + Y_{\text{out}} \end{aligned}$$

3.

$$\begin{aligned} X'_a &= X_a \oplus Y_{\text{out}} \\ X'_b &= X_b \oplus Y_{\text{out}} \\ X'_c &= X_c \oplus Z_{\text{out}} \\ X'_d &= X_d \oplus Z_{\text{out}} \end{aligned}$$

# IDEA Even Round: Inverse

$$X_a' = X_a \oplus Y_{\text{out}}$$

Feed $X_a'$ to input:

$$\begin{aligned} &= X_a' \oplus Y_{\text{out}} \\ &= (X_a \oplus Y_{\text{out}}) \oplus Y_{\text{out}} \\ &= X_a \end{aligned}$$

➠ round is its own inverse! ➠ same keys

# Encrypting a Large Message

- Electronic Code Book $\boxed{\text{(ECB)}}$
- Cipher Block Chaining $\boxed{\text{(CBC)}}$
- $k$-bit Cipher Feedback Mode (CFB)
- $k$-bit Output Feedback Mode (OFB)

# Electronic Code Book (ECB)

- break into 64-bit blocks
- encrypt each block independently
- some plaintext ➠ same ciphertext
- easy to change message by copying blocks
- bit errors do not propagate

➠ rarely used

# Cipher Block Chaining (CBC)

simple fix: $\oplus$ blocks with 64-bit random number

- must keep random number secret
- repeats in plaintext $\not\to$ = ciphertext
- can still remove selected blocks

# Cipher Block Chaining (CBC)

- random number $r_{i+1} = c_i$: previous block of ciphertext
- random (but public) *initialization vector* (IV): avoid equal initial text
- Trudy can't detect changes in plaintext
- can't feed chosen plaintext to encryption
- but: can twiddle some bits (while modifying others):
  modify $c_n$ to change desired $m_{n+1}$ (and $m_n$)
- ➠ combine with MICs

# Output Feedback Mode (OFB)

64-bit OFB:

- IV: $b_0 \xrightarrow{\text{encrypt}} b_1 \xrightarrow{\text{encrypt}} b_2 \ldots$
- $c_i = m_i \oplus b_i$, transmit with IV
- ciphertext damage ➠ limited plaintext damage
- can be transmitted byte-by-byte
- but: known plaintext ➠ modify plaintext into anything
- extra/missing characters garble whole rest

variation: $k$-bit OFB

# Cipher Feedback Mode (CFB)

- similar to OFB: generate $k$ bits, $\oplus$ with plaintext
- use $k$ bits of *ciphertext* instead of IV-generated
- ➠ can't generate ahead of time
- 8-bit $CFB$ will resynchronize after byte loss/insertion
- requires encryption for each $k$ bits

# Generating MICs

- only send last block of CBC ➠ *CBC residue*
- any modification in plaintext modifies CBC residue
- replicating last CBC block doesn't work
- P+I: use separate (but maybe related) secret keys for encryption and MIC ➠ two encryption passes
- CBC(message | hash)

# Multiple Encryption DES

- applicable to any encryption, important for DES
- encrypt-decrypt-encrypt (EDE): just reversible *functions*
- two keys $K_1$, $K_2$

$$\begin{array}{ccccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ m & \rightarrow & E & \rightarrow & D & \rightarrow & E & \rightarrow c & \end{array}$$

- decryption ➠ just reverse:

$$\begin{array}{ccccccccc} & & K_1 & & K_2 & & K_1 & & \\ & & \downarrow & & \downarrow & & \downarrow & & \\ c & \rightarrow & D & \rightarrow & E & \rightarrow & D & \rightarrow m & \end{array}$$

- standard CBC

# Triple DES: Why 3?

- security $\leftrightarrow$ efficiency
- $K_1 = K_2$: twice the work for encryption, cryptanalyst
- plaintext $m_i \xrightarrow{A:E(K_1)} r \xrightarrow{B:E(K_2)} c_i$ (ciphertext)
- *not* quite equivalent to 112 bit key:
  - assume given $(m_1, c_1), (m_2, c_2), (m_3, c_3)$
  - Table A: $2^{56}$ ($10^4$ TB) entries: $r = K\{m_1\} \forall K$, sort by $r$
  - Table B: $2^{56}$ entries: $r = c_1$ decrypted with $K$, sorted
  - find matching $r$ ➠ $K_A, K_B$
  - if multiple $K_A, K_B$ pairs, test against $m_2, c_2$, etc.
  - $2^{64}$ values, $2^{56}$ entries ➠ 1/256 chance to appear in table ➠ $2^{48}$ matches

# Triple DES: Why 3?

Table A:

| $r = E(m_1, K)$ (64 bits) | $K$ (56 bits) |
|---|---|
| ... | |
| 1234567890abcd00 | ab485095845922 |
| 1234567890abcd03 | 12834893573257 |
| 1234567890abcd04 | 43892ab8348a85 |
| 1234567890abcd08 | 185ab80184092c |
| ... | |

Table B:

| $r = D(c_1, K)$ (64 bits) | $K$ (56 bits) |
|---:|---:|
| $\ldots$ | |
| 1234567890abcd00 | 38acd043858ac0 |
| 1234567890abcd03 | 91870ab8a8d8a0 |
| 1234567890abcd07 | 058a0fa858abcd |
| 1234567890abcd09 | fd884a90407821 |
| $\ldots$ | |

computation: $2 \cdot 2^{56} + 2^{48}$

# Triple DES

- EDE: can run as single DES with $K_1 = K_2$
- can be used with any chaining method
- CBC on the outside ➠ no change in properties
- CBC on the inside ➠ avoid plaintext manipulation
- but want *self-synchronizing*: wrong bit $x$ in block $n-1$ ➠ $n-1$ garbled, $n_x$ changed, others unaffected
- CBC inside: parallelization