CS-4920: Lecture 7
Secret key cryptography

- Reading
  - Chapter 3 (pp. 59-75, 92-93)
- Today's Outcomes
  - Discuss block and key length issues related to secret key cryptography
  - Define several terms related to secret key cryptography
  - Describe and evaluate DES, focusing on both design and implementation issues
  - Explain some uses of one-time pads with RC4 as a representative example

1

Block encryption

- Block algorithms
  - Take a fixed-length message block... size?
    - Short block problem (*e.g.*, monoalphabetic cipher)
      - Too easy to make <plaintext, ciphertext> table
    - 64-bit blocks are commonly used
    - Longer is merely inconvenient, once security is established
  - Take a fixed-length key
    - 56 bits for DES, 128 for IDEA, typically ≥64 bits
  - Generate a ciphertext block equal in length to the input block

2

Determining the plaintext→ciphertext mapping

- Want a random-looking mapping
  - So a few <pt, ct> pairs cannot be used to infer the key without exhaustive search
    - Consider Caesar cipher vs. monoalphabetic
- For 64 bits, there are 2^{64}! 1-to-1 random mappings
  - Nearly 2^{70} bits needed – this key is too long
- Random-looking vs. random, various tests...
  - Single bit input change results in an apparently unrelated output
    - Roughly half of the bits are different
- We can get random-looking output with much **shorter**, **practical** keys...

3

Evading cryptanalysis by spreading input effects

- Substitution
  - A separate output for each input
  - Random tables reasonable for 8 bits (2^8 entries)
    - But not 64 bits (2^{64} entries)
  - Takes (nearly) $k \cdot 2^k$ bits
    - Bits per entry × entries
- Permutation
  - Spreads the influence of bits throughout the block
    - Let LSBs affect any bits, not just other LSBs
  - Random, reversible, reordering
  - Takes (nearly) $k \cdot \log_2 k$ bits
    - Entries × cost to encode a position

4

A round: substitution followed by permutation

- Choose a reasonable number of rounds
  - Too few: can see patterns and attack
  - Too many: inefficient
- Reversible
  - Everything done can be undone
    - If we know the permutations and substitutions

5

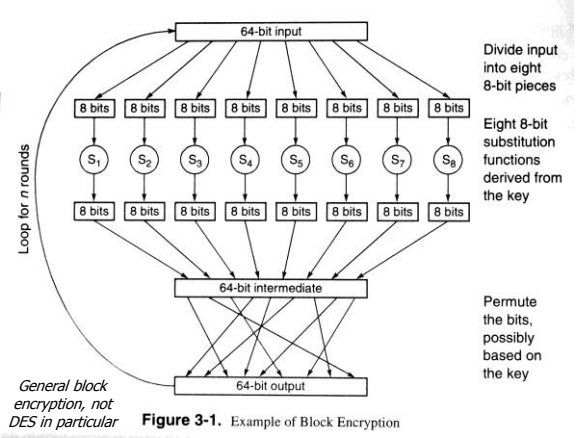

Figure 3-1. Example of Block Encryption

DES (Data Encryption Standard)

- 1977 – National Bureau of Standards (now NIST)
- 56-bit key, 64-bit I/O blocks
- Key appears as 64 bits
  - But LSB of each byte is an odd-parity bit
    - Consensus is no practical value to this use of parity
- Reasonable encryption speeds on standard CPUs
  - Roughly 60 kB/s per 1000 MIP, varying with architecture and implementation
- Much more efficient with custom hardware
  - Some features of DES do not add to security, but make software implementations inefficient

7

Breaking DES

- 1977
  - ~$20M hardware to find a key in 12 hours
- 1998
  - <$250k to find a key in 4.5 days
- Triple DES (Chapter 4)
  - E → EDE, D → DED
  - "Keying option 2," K1=K3
    - Believed to be 2^{56} times harder to break
    - Prevents "meet-in-the-middle" attack
    - Secure for foreseeable future
- 2010
  - 112 → 80-bit chosen plaintext attack per NIST

8

DES Overview

- 64-bit input through fixed permutation
- 56-bit key generates 16, 48-bit keys for 16 rounds
- Round processing × 16
  - 64-bit input and output, 48-bit key
- Swap halves
- Final permutation (inverse of initial)
- Decryption: just reverse the steps

9

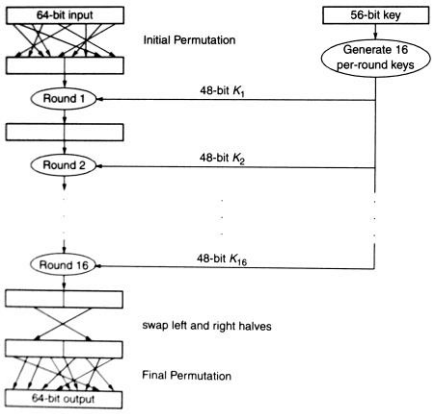

Figure 3-2. Basic Structure of DES

10

Initial and final permutations

- 64-bit to 64-bit
- See book, pages 66-67
- Do not add security value
  - Since they are fixed and occur at the very beginning and end
  - Just to make software implementation on a general CPU less efficient?

11

Generating the per-round keys

- 56-bit key is passed through a fixed permutation
  - Again, no security value
  - The two 28-bit halves are called C_0 and D_0
- Rounds 1-16
  - Left rotate C and D each by 1 (round 1, 2, 9, 16) or 2 bits
    - The rotations go full circle (1·4 + 2(16-4) = 28)
  - C and D are permuted (with 4 bits discarded) to generate the 2 halves of the 48-bit round key
    - These permutations are believed to have security value
      - Less locality in the round keys

12

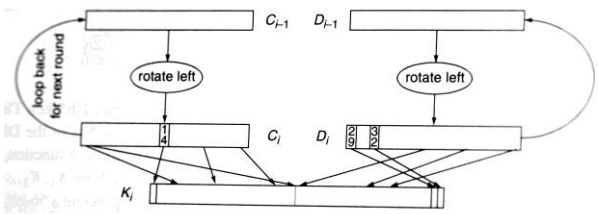

Figure 3-5. Round *i* for generating K_i

13

DES round (see Figure)

- Block divided into 2, 32-bit halves
  - L_n and R_n
- $L_{n+1} = R_n$
- $R_{n+1} = L_n \oplus \text{mangler}(R_n, K_n)$
- Decryption: reverse process
  - Need L_n. Reverse mangler? No (elegant design)…
    - Use, $A \oplus B \oplus B = A$
      - $L_n = R_{n+1} \oplus \text{mangler}(R_n, K_n)$
    - R_n and K_n are known

14

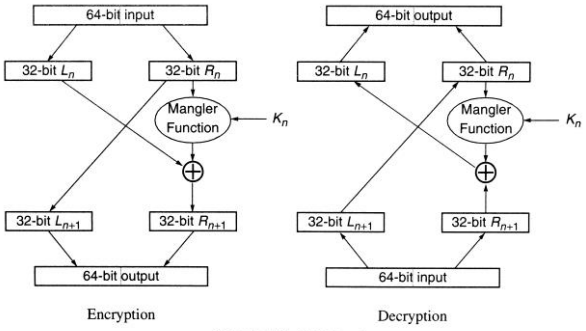

Figure 3-6. DES Round

15

The mangler

- Purpose: scramble the data based on the round key
- Also called the Feistel function
- Inputs: R_n (32-bit data block), K_n (48-bit round key)
- Output: Mangled 32-bit block
- Processing
  - Expand R_n to 48 bits
    - Take 8, 6-bit chunks that overlap by 2 bits
      - bits -1 through 4, 3 through 8, ...
  - (Expanded R_n) $\oplus$ K_n
  - Map each 6-bit chunk to a 4-bit chunk
  - Separate table for each of the 8 chunks (S-boxes)
    - Center 4 bits are based on the data
    - 2 edge bits select 1 of 4 subtables for each chunk
  - Permute the final 32-bit quantity
    - Security value: bit spreading for next round, good randomness property

16

Weak and semi-weak keys

- 4 of the 2^{56} keys are weak
  - All 0s or 1s in C_0 and D_0
  - They are their own inverses
    - Encrypting with a key is the same as decrypting with its inverse
- 12 more are semi-weak
  - Alternating 1010 or 0101 in C_0 and/or D_0
  - The inverse of each is in the set of 12

17

DES Issues

- Design process was secret
  - Were the S-boxes chosen to be weak?
- 1991: Swapping box 3 with 7 makes DES about an order of magnitude less secure
  - Admittedly specific, unlikely attack
- To address this "secret weakness" concern
  - Many cryptographic algorithms choose their "random" numbers based on demonstrable methods
  - *E.g.*, digits of π

18

RC4

- One-time pad – a long (pseudo)random string used to encrypt a message
  - Use one time – theoretically very secure
  - But, how do we generate?
- Pseudorandom
  - Passes many/all tests for randomness
    - Distribution, correlation with previous samples (overall or of particular bits), …
  - Generated by algorithm – predictable if you know algorithm/key
- Stream cipher – apply one-time pad to a stream of plaintext
- RC4 (page 93)
  - Simple algorithm generating a 1-time pad based on a key of 1 to 256 bytes
  - 258 bytes of state information
  - Also useful for generating pseudorandom numbers

19