CNS
Lecture 5

Digital Encryption Standard (DES)

Other Feistel ciphers (Lucifer, blowfish, CAST)

Non-Feistel (IDEA, RCx)

Blocking (ECB,CBC,OFB,CTR)

<u>assignments 4 and 5</u> and 6

In the news

- Over 1,000 laptops missing from Commerce Dept since '01
- Massive growth in organized crime targeting home PCs
- DNS attack in China takes 180,000 web sites offline
- Purdue Univ. notifying students of possible data breach
- Stolen laptop holds data on 50,000 GE employees

You are here …

Attacks & Defenses
- Risk assessment✓
- Viruses✓
- Unix security✓
- authentication✓
- Network security
  Firewalls,vpn,IPsec,IDS

Cryptography
- Random numbers✓
- Hash functions✓
  MD5, SHA,RIPEMD
- Classical + stego✓
- Number theory
- Symmetric key
  DES, AES, RC5
- Public key
  RSA, DSA, D-H,ECC

Applied crypto
- SSH
- PGP
- S/Mime
- SSL
- Kerberos
- IPsec

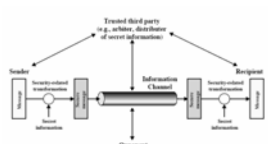
ciphers

- use substitution and permutation (SPN)
- assume algorithm known
- strength based on key
- resist cryptanalysis/statistical analysis
  - diffusion -- spread statistics of plaintext into many bits of ciphertext
    one plaintext bit affects many ciphertext bits --permute and replace
  - confusion -- use complex substitution to hide relation between key and ciphertext
  - bigger block (multiple characters) is better (playfair, hill)
  - resist chosen plaintext attacks
- efficient (speed/memory)

manual → machine/device → computers

computers easily break classic schemes

Symmetric ciphers



(shared) secret key
  Bob and Alice share a secret or key

- **block:** DES, IDEA, CAST, RC5, Blowfish, AES
  - ingredients: key, plaintext
  - pre-mix/expand key
  - break plaintext into block (e.g. 8 characters)
  - stir in some key bits and plaintext (block at a time)
  - stir in some more key bits, repeat N times for each block
  - BUT it's reversible!
- **stream:** RC4, hash, one-time pad, LFSR's
  - Encrypt a character at a time
  - XOR plaintext with keystream $c_i = p_i \oplus k_i$

DES roadmap

Digital Encryption Standard
- DES history
- DES internals, Feistel ciphers
- DES design
- DES attacks
- DES API, performance

Crypto Toolkit
- secret-key crypto
- public-key crypto
- big-number math
- random numbers✓
- prime numbers
- hash functions✓

DES history

- hodge-podge of incompatible crypto gear
- commercial interest in encryption
- NBS (NIST) 1972 call for proposals

Public encryption algorithm
- provide high security
- complete specification and easy to understand
- security NOT depend on algorithm secrecy
- available to all users
- adaptable for diverse applications
- economical for <u>hardware</u> implementation
- efficient
- able to be validated
- exportable

- based on IBM's Lucifer cipher (Feistel)
- assisted (?) by NSA
- authorized for use in '76 (unclassified)
- details published (NSA mistake)--software
- intent was hardware only

Feistel ciphers

- basis of many digital ciphers
- key is "expanded" into subkeys (K_1,K_2…)
- input block is split into halves
- operate on one half each round then swap
- multiple rounds
- each round has same structure using a mangler function F and a subkey and an XOR ⊕ -- <u>substitute and permute</u>
- output of one round is input to the next
- decryption uses same algorithm but with subkeys in reverse order
- mangler function F need not be invertible

Examples: DES, CAST, Blowfish, Lucifer

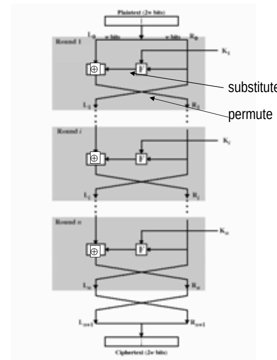


Figure 3.5 Classical Feistel Network

Feistel ciper

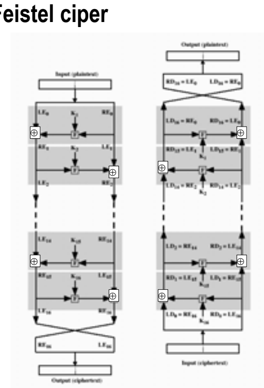


Figure 3.6 Feistel Encryption and Decryption

```
XOR properties
  (A ⊕ B) ⊕ C = A ⊕ (B ⊕ C)
  A ⊕ A = 0
  A ⊕ 0 = A
```

Mangler function F need not be reversible!

Decryption:

$LD_1 = RD_0 = LE_{16} = RE_{15}$

$RD_1 = LD_0 \oplus F(RD_0, K_{16})$

$= RE_{16} \oplus F(RE_{15}, K_{16})$

$= (LE_{15} \oplus F(RE_{15}, K_{16})) \oplus F(RE_{15}, K_{16})$

$= LE_{15}$

Strength vs speed

- bigger block size is stronger, but slower
- larger key is stronger, but slower
- more rounds are stronger, but slower
- complex mangler function is stronger
- subkey generation can effect strength, but it's only done once
- choose operations to be efficient in software/hardware and easy to analyze but hard to break

Baby DES

S-DES, teaching aid (appendix C)

- Feistel cipher
- specs
  - 10-bit key
  - 8-bit input (split into 2x4)
  - 2 rounds
  - 2 S-boxes (4 bits in, 2 bits out)
- like DES, has beginning and ending permutation
- key expansion at start

256-character alphabet (not just A-Z)

even if alphabetics in, lot of non-alphabetics out (e.g. Binary data)

S-DES steps

- subkey generation -- use various combinations of key bits to create subkeys for use in each round (key schedule)
- permute initial plain text ('cause DES does)
- iterate feistel rounds
- final permutation (inverse of first)

ciphertext = $IP^{-1}(f_{k2}(SW(f_{k1}(IP(plaintext)))))$

plaintext = $IP^{-1}(f_{k1}(SW(f_{k2}(IP(ciphertext)))))$

S-DES

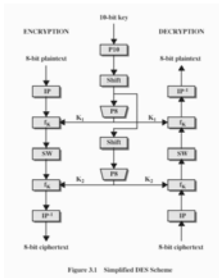


Figure 3.1 Simplified DES Scheme

Generating subkeys

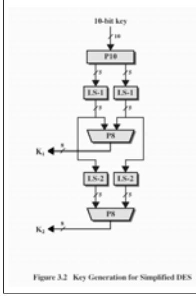


Figure 3.2 Key Generation for Simplified DES

1010000010

10000 01100

00001 11000

10100100 K_1

00100 00011

01000011 K_2

subkey applet

permutations

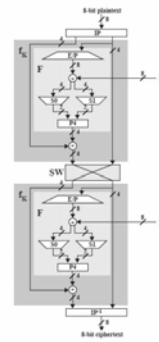
P10: 3 5 2 7 4 10 1 9 8 6 P8: 6 3 7 4 8 5 10 9

IP: 2 6 3 1 4 8 5 7 IP^{-1}: 4 1 3 5 7 2 8 6

S-DES round

- split 8-bit input in half (L,R)
- expand 4 bits to 8 bits
- in F, R is mixed with subkey then S-box substitution
- L is XOR'd with ouput of F
  $f_K(L,R) = (L \oplus F(R,SK),R)$
- output halves are swapped for next round
  animated applet

How would you do these in software:
- XOR
- Permutation and swapping
- substitution



E/P: 4 1 2 3 2 3 4 1

P4: 2 4 3 1

S boxes (Substitution)

- 2 S-boxes
- each with 16 2-bit values (pre-defined)
- 4-bit input selects 2-bit value (irreversible)
- provides strength, non-linearity (code book)

```
S0
        00  01  10  11
      +----------------
   00 | 01  00  11  10
   01 | 11  10  01  00
   10 | 00  10  01  11
   11 | 11  01  11  10

S1
        00  01  10  11
      +----------------
   00 | 00  01  10  11
   01 | 10  00  01  11
   10 | 11  00  01  00
   11 | 10  01  00  11
```

Spec: 4-bit input: bits 1 and 4 select row, bits 2 and 3 select column

Example: input = 0100 for S0, output 11

DES basics

- block Feistel cipher (64 bits at a time)
  - Mixing 8 plaintext bytes at a time (Playfair, Hill)
  - not stream cipher (Caesar, one-time pad)
- uses 56 bits of a 64-bit key
- bit-oriented (slow in software)
- 16 rounds (iterative)
- output from previous round used as input to next
- key expanded into 16 pieces
- permutations
- substitutions (S boxes), crypto strength
- can't reverse the S box without key (many-to-one)

"good for 5 years"

Used for financial transactions, software (ssh, ssl, ...), the algorithm of choice til AES

DES vs S-DES

S-DES instructional only
both Feistel
both useless initial/final permutation
64-bit block vs 8
56 bit key vs 10
16 rounds vs 2
key expanded into 16 pieces vs 2
8 6-to-4 S boxes, vs 2 4-to-2

DES structure

- key expansion
- initial permutation of 64-bit input (plaintext)
- 16 iterative rounds using different subkey each round
- final permutation of 64-bit output (ciphertext)
- initial/final permutations of no security value --slow down software ?

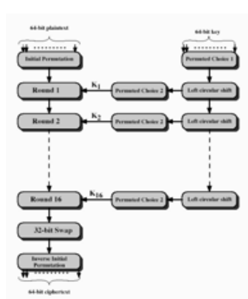
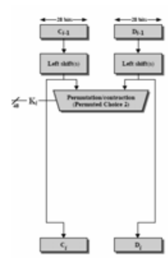


Figure 3.7 General Depiction of DES Encryption Algorithm

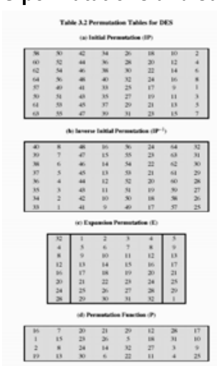
DES subkey generation

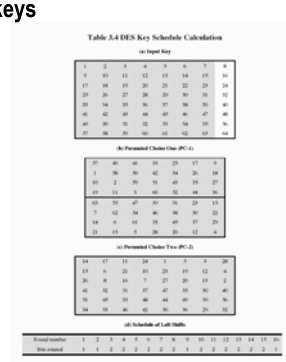
key schedule

- 64-bit key
- discards parity bits (8,16,...64)
- permutes remaining 56 bits into two 28-bit halves
- permutation is roughly a transpose (cols to rows)
  permutation of no security value
- generate 16 48-bit keys $k_1, k_2, \ldots k_{16}$
- iterative
- 1 or 2 bit left rotate, then compression permutation
- 1-bit rotate in rounds 1,2,9, 16
- different subset of key bits used in each subkey
- set_key() in software

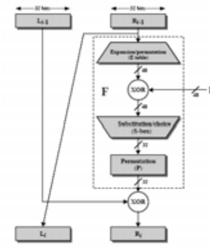


DES permutations and subkeys





DES round -- encryption



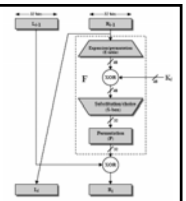
- 64-bit input split L_j and R_j
- mangler function $F(R_j,k_j)$
- mangler XOR'd with L_j to produce R_{j+1}
- R_j becomes L_{j+1}

  $L_i = R_{i-1}$

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

Decryption: start with cipher text and use subkeys in reverse order

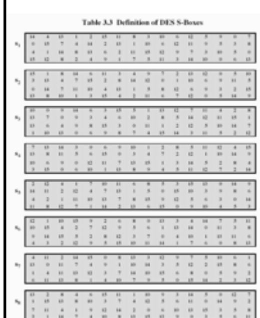
mangler function (F)



- substitution (strength)
- input is 32-bit R_j
- expansion permutation (E)*
  - 32 → 48 bits
- 48-bit subkey XOR'd with output of E
- S boxes (48 → 32 bits)
  - 8 6-bit chunks to 4-bit chunks
  - S boxes are predefined tables -- where did these numbers come from?
- 32-bits are permuted (P), permutation insures S-box output will affect multiple S-boxes in next round

*Note: E permutation is the one modified by UNIX passwd crypt()

DES S-box



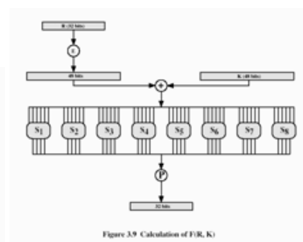


Figure 3.9 Calculation of F(R, K)

S-box (substitution)

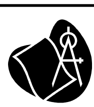
48 bits in → 32 bits out, not reversible!

eight 6x4 S boxes (S_1 … S_8)

6 bit input – outer two bits select row,

inner 4 bits select column → 4 bits out

Designing a cipher



- looks simple
- shuffle bits around, mix in some key bits ...
- for DES
  - permutations?
  - Why 16 rounds?
  - Why 56 bit keys?
  - Why subkey generation?
  - Why Feistel ?
  - Why S-boxes ?
  - Where did numbers in S boxes come from?

DES strengths

- expansion permutation allows one bit to effect two substitutions
- avalanche -- dependency of output bits on input bits spread faster
- quickly have every ciphertext bit depend on every input bit and key bit
- 8 rounds are sufficient to eliminate any one ciphertext bit dependence on subset of plaintext bits

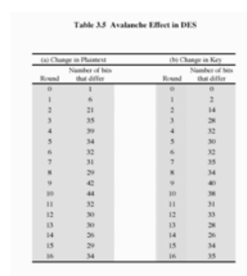


Table 3.5 Avalanche Effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

How many rounds

- after 5 rounds: every ciphertext bit is a function of every plaintext and key bit
- after 8: ciphertext is a random function of plaintext and key
- reduced-rounds DES have been broken
  - 4 rounds broken in '82
  - 6 rounds broken in '86
- differential cryptanalysis broke anything less than 16 rounds with known plaintext attack -- more efficient than brute force

subkeys

- Feistel needs subkey for each round
- subkey generation (key schedule)
  - expand (no additional strength?)
  - use different bits for each subkey
  - make it difficult to deduce key from subkey
  - OK to be slow
- should aid in avalanche

DES key expansion is weak (a circular shift then a permutation)

DES controversy

Did NSA leave a backdoor?

- design decisions kept secret
- several congressional reviews
- Coppersmith paper reveals S-box design, knew about differential cryptanalysis
- IBM says NSA didn't mess with algorithm
- but IBM had recommended 112-bit key !

S boxes

- all of the algorithms involved in DES are linear in binary arithmetic, if S-boxes were also linear then:

  $c = Ap \oplus Bk \oplus b$

  A, B,b are fixed and k is the 56-bit key. Knowing one (p,c) pair, then

  $k = B^{-1} (c - Ap - B)$ (you could solve for the key, k!)
- S-boxes need to be nonlinear
- 2x2 S boxes are linear
- some 3x3 boxes are linear
- studies have shown none of DES S-boxes (6x4) are affine (linear)
- **strict avalanche criterion** any output bit j should change with probability 1/2 when any input bit i is changed, for all i,j
- **bit indepedence criterion** ouput bits j and k should change independently when any single input bit i is changed, for i,j,k

S box design

Coppersmith DES paper

- 6x4 largest that would fit on '74 technology
- no output bit should be too close to linear function of input bits
- fix bits 1 and 6, vary middle bits, each possible output bit should be produced
- if two inputs differ by 1 bit, output must differ in at least 2 bits
- if two inputs differ in middle 2 bits, output must differ in at least 2 bits
- if two inputs differ only in first 2 bits, outputs must not be the same

spent months deriving S boxes and P permutation

Alternatives for selecting S box values

- random numbers -- may lead to S boxes with unwanted properties, maybe OK for large S boxes (8x32)
- random plus testing -- throw away bad ones
- man-made -- basically what DES did, not practical for larger S boxes
- math-made -- proven against linear/differential cryptanalysis (CAST, Bent functions, Rijndael analytical) plus testing
- key-based S boxes (Blowfish), like random, but different for each key!

DES weaknesses

- test of time
- lots of studies
- key weaknesses (size)
- 16 weak keys (self-inverse)
  - encrypt with one = decrypt with other
- complement reduces keyspace 2^{55}

  if y = $DES_k(x)$ then $y^c = DES_{k^c}(X^c)$
- alternatives to brute force key search
  - consider trying to find key, given (plaintext,ciphertext) for 1-round DES
  - differential cryptanalysis
  - linear cryptanalysis

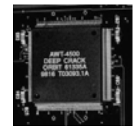
Weakness of key

- small key, brute force (see performance)
- brute force: 2^{56} keys, 2^{25} secs/yr → 2^{45} guesses/mip-yr

  1000 years for a 1 mip processor (or 1 Mguess/sec)

  1 year for a 1 GHz/mip processor

  1 day for 365 1 GHz processors ….
- special hardware for key guessing (pipelined/parallel)
- EFF DES cracker ($250K): 3 days
  - amortized cost over 3 yrs, 8 cents per key
  - If your secret is worth more than 8 cents, don't use DES
  - EFF + net: 22 hours
- NSA: 5 minutes ?
- dictionary attacks ("the human factor")
  - 56 bits is 8 7-bit ASCII
  - alphanumeric (8 x 5-bits/char = 40 bits)
  - drop low bit for parity, 32 bits



DES cryptanalysis

- differential ('90)
  - examine ciphertext pairs whose plaintext have particular differences
  - recover key, bit by bit, round by round
  - needs lots of chosen plaintext (2^{47} pairs)
  - fewer than 16 rounds, susceptible
  - greater than 18 rounds, more work than brute-force
  - S boxes optimized to thwart
  - 8 round attack: Lucifer 256 chosen plaintext, DES 2^{14}
- linear ('93)
  - linear approximation to action of block ciphers
  - XOR some plain and cipher text together, get a bit that is XOR of some of the key bits
  - needs lots of known plaintext (2^{47})
  - recovered a key in 50 days with 12 HP9735's

These attacks are effective against any Feistel cipher and have a work-factor smaller than brute force, BUT you need lots of plaintext/ciphertext for the desired key!

Countermeasure: change key "often"

DES in software

- messy in software (bit-based)
- use table lookups for S boxes
- source available on the net
- OpenSSL
  - File encrypt/decrypt
  - API for encrypt/decrypt

```
data structures: S boxes

encrypt(key,plain,cipher)
{
   expand_key(key) //subkeys
  // rotate/permute
   in = permute1(plain)
   for i=1, rounds
   {
         out = rnd_fcn(in,subkey[i])
         in = out
    }
   cipher = permute2(out)
}

rnd_fcn(in, sk)
{
         L = left(in)
         R = right(in)
         X = L ^ F(R,sk)
         L = R
         R = X
         return (LR)
}
```


UNIX encryption

- crypt(3) password hash function (modified DES)
  - Uses 25 iterations of modified DES (EP is altered to thwart DES hardware cracker)
  - Encrypts 0 using salt/password
  - Encode 64-bit output to 6-bit alphanumerics
- crypt command
  - 1 rotor enigma, polyalphabetic (256)
  - key guessing
  - crypt breakers workbench (cbw)
  - compression helps
- setkey() encrypt() -- DES functions, key and message expressed as binary ASCII -- slow, 1 bit per byte
- Strong encryption: PGP, ssh, OpenSSL, kerberos, cfs
- many packages use simple encryption
  - guides to cracking on web (WORD, wordperfect, PKZIP)

Strengthening DES

- take advantage of existing DES hardware
- take advantage of test-of-time of DES
- superencrypt
- double DES

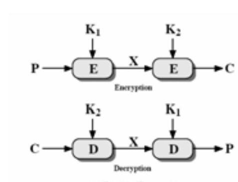
  encryption: $E_{k2}(E_{k1}(m))$

  decryption: $D_{k1}(D_{k2}(m))$

Is there a k3 such that

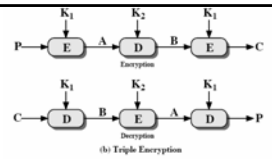
$E_{k2}(E_{k1}(m)) = E_{k3}(m)$?

- DES defines $2^{56} < 10^{17}$ mappings of 64-bits to 64-bits out of possible $(2^{64})! = (10^{10})^{20}$
- there is no k3 (DES not a group, '92)
- double encryption with two keys (112 bits), not a big improvement -- meet-in-the-middle attack requires only twice the effort of single DES



(a) Double Encryption

Triple DES (3DES)



triple DES: $E_{k1}(D_{k2}(E_{k1}(m)))$

- 112-bits
- decrypt: $D_{k1}(E_{k2}(D_{k1}(c)))$
- compatible with DES if k1=k2 thanks to EDE
  $E_{k1}(D_{k1}(E_{k1}(m))) = E_{k1}(m)$

- 3 key: $E_{k3}(D_{k2}(E_{k1}(m)))$
  - 168 key bits
  - compatible with DES if k1=k2 or k2 = k3
  - used by PGP and S/MIME

3 DES performance

speed.c in libdes (also see openssl speed command)

```
  on cetus engine
set_key             per sec =    118258.95 (  8.5uS)
DES raw ecb bytes   per sec =   2089940.80 (  3.8uS)
DES cbc bytes       per sec =   1959656.91 (  4.1uS)
DES ede cbc bytes   per sec =    739647.04 ( 10.8uS)
crypt               per sec =      8297.39 (120.5uS)
```

- hardware can pipeline so 3DES is not that much slower than DES
- improved resistance to brute force and linear/diff. cryptanalysis
- Since '98 banks require 3DES rather than DES

DES-X

- DES extension (Rivest, '84)
- $DESX_{k,k1,k2}(m) = k2 \oplus DES_k(k1 \oplus m)$
- 184 bit "key" (56 + 64 + 64)
- "whitening" keys k1 and k2
- mask plaintext, then ciphertext
- just as fast as DES, uses existing DES (e.g., hardware)
- brute force "impossible" 2^{184}
- more plaintext/ciphertext required for linear and differential cryptanalysis (2^{60})
- using + instead of $\oplus$ is even stronger against linear/diff.

DES – executive summary

- block cipher (64 bit)
- symmetric, secret key
- 56-bit key
- product cipher (combo of simple operations)
- substitution: 6x4 4-bit (S box) (strength)
- transposition: swap and permute
- 16 rounds
- awkward in software -- bit manipulation: permutation, shifts

why no nice mathematical representation?

lucifer

- Feistel, '70 IBM
- DES predecessor
- 128-bit blocks/key
- 16 rounds (key-dependent nibble swap, 64-bit permute)
- weak key schedule (72-bit sub-key/round)
- weak, 4x4 S boxes
- weak against differential attacks
  8 round attack: Lucifer 256 chosen plaintext, DES 2^{14}
- longer key is not sufficient

Sub-keys

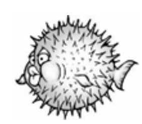
$Round_1$: repeat first byte, append next 7

$Round_n$: rotate previous left by 7 bytes

S boxes

Nibble	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S0	c	f	7	a	e	d	b	0	2	6	3	1	9	4	5	8
S1	7	2	e	9	3	b	0	4	c	d	1	a	6	f	8	5

blowfish (Schneier)



- like DES (Feistel) (both halves)
- iterative (16 rounds)
- block cipher (64-bits)
- fast, 32-bit (worry about byte order)
- compact (5K)
- simple: add, XOR, lookups
- variably secure: key length up to 448 bits
- four 8x32-bit S-boxes, 256 entries each
- key expansion builds 18 32-bit subkeys and four S-boxes (521 executions of blowfish)
- slow subkey generation makes it bad for rapid key switching, but makes brute force expensive
- initial value of S-boxes and subkeys Pare digits of pi
- twice as fast as DES
- in OpenSSL
- AES candidate (twofish)

blowfish

- Setup: generate sub-keys and 4 S-boxes
- Encryption/decryption uses XOR and addition (mod 2^{32})
- Both L and R modified in a round

subkey/S-box generation

Initialize 4 S-boxes and 18 subkeys P with pi π

then XOR with key (repeating key as needed)

$P_1,P_2 = E_{P,S}[0]$

$P_3,P_4 = E_{P,S}[P_1 || P_2]$

…

$P_{17},P_{18} = E_{P,S}[P_{15}||P_{16}]$

$S_{1,0},S_{1,1} = E_{P,S}[P_{17}||P_{18}]$

…

$S_{4,254},S_{4,255} = E_{P,S}[S_{4,252}||S_{4,253}]$

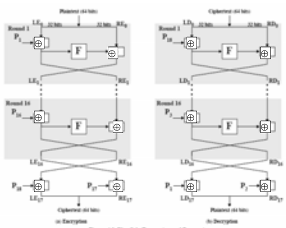


Figure 6.3 Blowfish Encryption and Decryption

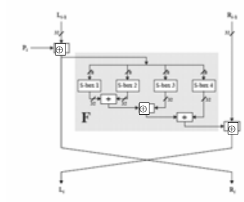


Figure 6.4 Detail of Single Blowfish Round

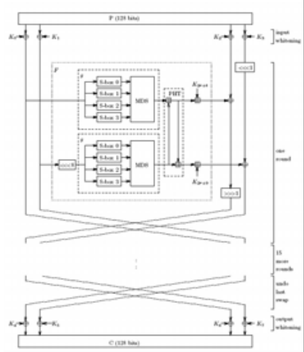
Blowfish vs DES

- S-boxes are <u>key dependent</u> produced by repeated applications of a "changing" blowfish
- Subkey generation is very strong (but slow)
- Both halves of data mangled in each round
- With key of 448 bits, invulnerable to brute force attack, plus takes 522 executions of blowfish to test a single key!
- Computationally efficient round (fast)
- No round-dependent F functions
- Function F has perfect avalanche effect
  - Every subkey bit is affected by every key bit
  - Every bit of L_{i-1} affects every bit of R_{i-1}

Test of time?

twofish

- Son of blowfish, AES candidate
- 128-bit blocks, key up to 256 bits
- Key schedule like blowfish
- Pre/post whitening with key material
- Four key-dependent 8x8 bit S-boxes
- Round function
  - maximum distance separable (MDS) matrix (8x32 bit table)
  - pseudo-Hadamard transform (PHT) a' = a +b b' = a +2b
  - Addition, rotation

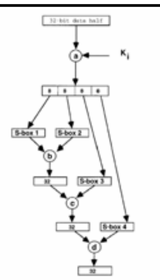


CAST-128

- Feistel, 64-bit block, 128-bit key
- 16 rounds
- add/subtract modulo 2^{32}, XOR, rotates
- pre-defined 8x32 S boxes based on **Bent functions** (highly nonlinear)
- strong subkey generation using S boxes
- key-dependent rotates
- masking function depends on round
- used in Entrust, OpenSSL

CAST round

- Designed in early 90's to resist known attacks
- Design parameters: key/block size, number of rounds
- Round functions selected to resist diff./lin. cryptanalysis
  - -key-dependent rotate
  - -round-dependent round functions
- S-boxes designed to provide high avalanche and bit independence (select and test from bent functions)
- Key schedule insures no weak or semi-weak keys



Type 1: $I = ((K_{mi} + D) <<< K_{ri})$
$f = ((S1[I_a] \hat{} S2[I_b]) - S3[I_c]) + S4[I_d]$

Type 2: $I = ((K_{mi} \hat{} D) <<< K_{ri})$
$f = ((S1[I_a] - S2[I_b]) + S3[I_c]) \hat{} S4[I_d]$

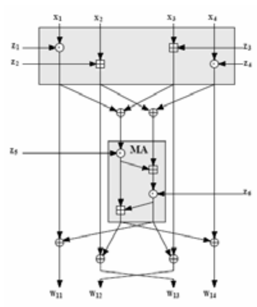
Type 3: $I = ((K_{mi} - D) <<< K_{ri})$
$f = ((S1[I_a] + S2[I_b]) \hat{} S3[I_c]) - S4[I_d]$

Rounds 1, 4, 7, 10, 13, and 16 use f function Type 1.
Rounds 2, 5, 8, 11, and 14 use f function Type 2.
Rounds 3, 6, 9, 12, and 15 use f function Type 3.

IDEA

- Swiss cipher ('91) in original PGP, patent problems
- 64-bit block, 128-bit key, 16-bit words non Feistel
- Three operations: addition 2^{16}, XOR, multiplication mod 2^{16} + 1
- Decryption needs multiplicative inverse and subtraction, and modified subkeys
- 8 rounds + 1 output transformation
- 52 16-bit subkeys (key schedule: 25 bit rotate)
- Large classes of weak keys ☹
  - Could be fixed with better key schedule

IDEA single round



X_i 16 bit Z_i 16 bit subkey

RC2

Ron's Code

- non-Feistel (Rivest, '97)
- used in S/MIME
- optimized for 16-bit arithmetic, addition/subtraction, XOR, AND, complement, rotate
- 64-bit block
- 18 rounds (mixing/mashing)
- 8 to 1024 bit key
- subkey generation uses XOR and digits of pi
- subkey selection is data dependent in each round
- decryption: rounds and subkeys in reverse order, subtract for add
- 40-bit key for export

RC5

- Rivest's Cipher (RSA) RC2, RC4, (RC6/AES)

- parameterized block cipher, not Feistel
- select key size, rounds, block size
- RC5-32/16/10 (16 rounds, 80bit key), 32-bit word worry about byte-order
- uses XOR, rotate, add/subtract **mod** wordsize
- no substitutions
- security: data-dependent rotates
- key expansion uses addition/rotates, seeded with constants e and ϕ
- decryption: reverse rounds, rotation, and use subtraction
- twice as fast as DES
- licensed, in lots of RSA products

RC5

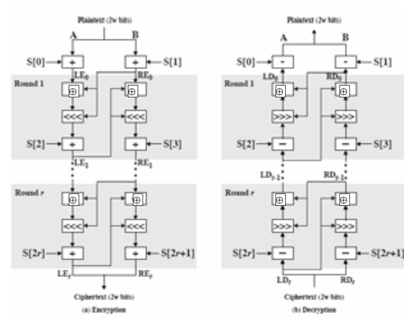


Figure 6.6 RC5 Encryption and Decryption

- Uses addition, XOR, left circular shift
- RC5 round == two DES rounds

Guessing an RC5-w/r/b key

- Brute force key guessing (RSA/ECC challenge)
- www.distributed.net/rc5 (like SETI@home)
- RC5-32/12/7 (56 bit key)
  - Found 10/19/97, 250 days
- RC5-32/12/8 (64-bit key)
  - Found 8/12/02, 1757 days
  - 247 x10 ^{6} keys/sec (max rate) 00.12% keyspace/day
  - 331,252 participants tested 15,769,938,166,961,326,592 keys
- RC5-32/12/9 (72 bit key) … work in progress

NOTE: this only finds one key, doesn't really "break" the cipher

RC6

- AES candidate
- Key schedule same as RC5
- Changes from RC5
  - 2 streams of RC5, AB and CD
  - Mix AB CD streams in swap
  - 5-bit rotation ($\log_2 32$)
  - Quadratic function (B (2B + 1))
  - Pre and post whitening steps

Encryption with RC6-$w/r/b$

Input: Plaintext stored in four w-bit input registers A, B, C, D
Number r of rounds
w-bit round keys $S[0, \ldots, 2r+3]$

Output: Ciphertext stored in A, B, C, D

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
{
$t = (B \times (2B+1)) \lll \lg w$
$u = (D \times (2D+1)) \lll \lg w$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i+1]$
$(A, B, C, D) = (B, C, D, A)$
}
$A = A + S[2r+2]$
$C = C + S[2r+3]$

Block cipher design

substitution and transposition

- confusion and diffusion
- easy if you have memory for 48x32 S-boxes
- easy if you iterate for 128 rounds
- easy if you use a 512-bit key
- trick is design one with
  - –smallest possible key
  - –smallest memory requirement
  - –low power consumption
  - –fastest running time

Feistel ciphers: Lucifer, DES, Blowfish, CAST

not: IDEA, RC5, RC2, AES/Rijndael

Block cipher design -- summary

Table 6.1 Speed Comparisons of Block Ciphers on a Pentium

Algorithm	Clock cycles per round	# of rounds	# of clock cycles per byte encrypted
Blowfish	9	16	18
RC5	12	16	23
DES	18	16	45
IDEA	50	8	50
Triple-DES	18	48	108

substitution and permutation

- performance (time/space) vs strength
- large keys
- strong subkey generation
- large blocks
- simple operations, complex, non-linear functions (S-box, rotate)
- iterative, more rounds
- resist known attacks (diff./lin.)
- ciphertext should have uniform distribution (look random)

analyze, analyze, analyze

Blocking for block ciphers

Handling messages bigger than 64 bits….

- do 64-bits (block) at a time
- applies to all block ciphers (FIPS 81)
- rule for padding last block and/or encoding length? PKCS5
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

Chaining/feedback methods require an initialization vector (IV) to start
-change IV for each message so same message encrypts differently
-keep IV secret?

padding

- Needed for ECB and CBC modes
- Last block must be filled out to block size (DES 8 bytes, AES 16)
- Recall: hashes did last block padding too (10000…length)
- Output of encryption will be multiple of block size
- Could provide "actual" length out of band or encode in pad
  - –If encode in pad, then will need extra block if exact multiple
  - –Sender and receiver must agree on length/padding encoding
- Lots of ways to pad
  - –Pad with 0's, last byte = # of padding bytes
  - –Pad with 0,s or spaces or random (need out of band length)
  - –Pad with bytes all = # of padding bytes (PKCS5)
- DES encrypt "hello" 0x68656c6c6f030303
- OpenSSL API will pad with null (0's)

ECB -- electronic code book

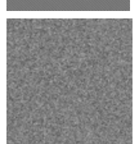
original

- just encrypt each block
- worry about padding last block
- blocks can be done in parallel ☺
- simple, stupid ☹
  - –identical plain will encrypt to identical cipher
  - –replace ciphertext blocks with other blocks
  - –reorder blocks
  - –lot of structure remains
- Loss of a cipher block? Change a cipher bit?

PLEASE use only for one-block messages!

ECB

CBC



CBC

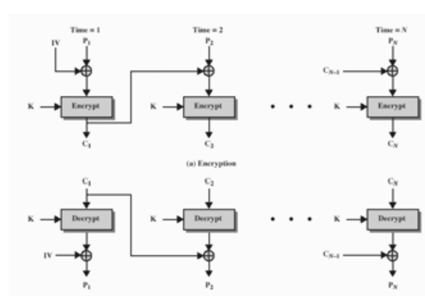


Figure 3.12 Cipher Block Chaining (CBC) Mode

CBC – cyclic block chaining

CBC chaining in software

```
previous = IV
while(moreplaintext){
    plain = next_block()
  encrypt(result,key,xor(previous,plain))
    output result
    previous = result
}
```

- need an IV (initialization vector) first iteration
- iterative, $E_k(p_i \oplus c_{i-1}) = c_i$
- decryption: $D_k(c_i) \oplus c_{i-1} = p_i$
- worry about padding last block
- use for long messages, files
- communicate IV (in the clear?)
- differing IV assures same plaintext produces different ciphertext
- garbled cipherbits affect two plaintext blocks
- cipher blocks lost or added, rest of decryption trashed
- doesn't assure integrity
  - can change a bit of ciphertext and have it alter bit of plaintext on decryption!
  - rearrange ciphertext blocks

CBC-MAC

also can use final output of CBC encryption as MAC/hash (slow) or better, CMAC, but need export license ☹

CFB

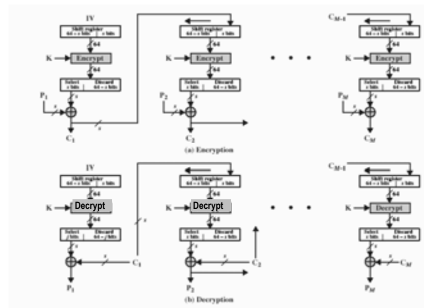


Figure 3.13 s-bit Cipher Feedback (CFB) Mode

CFB – cipher feedback

- need an IV
- cipherbyte fed into next step $\quad c_i = E_k(c_{i-1}) \oplus p_i$
- can do a byte (8-bits) at a time
- if cipherbytes are lost, CFB will re-synch, but every byte of input requires an encryption operation
- Flip a cipher bit results in bit flip of plaintext bit (and half next block)
- useful for short message encryption (e.g., telnet)
- used by stel, ssh v1, deslogin (e.g. interactive streams)

Keep IV secret?

-there are some attacks based on knowledge of IV

-best practice is to derive IV from message key and session nonces

OFB

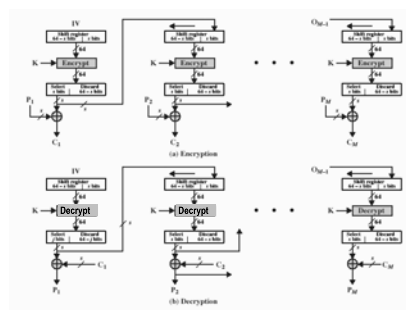


Figure 3.14 s-bit Output Feedback (OFB) Mode

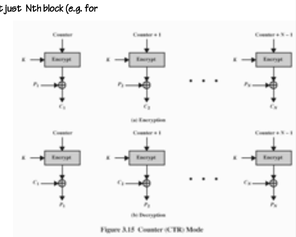
OFB – output feedback

- need an IV
- best used in 64-bit mode
- can pre-generate stream (o_i) (a PRNG)
- XOR plaintext with OFB stream $\quad c_i = E_k(o_{i-1}) \oplus p_i$
- like one-time pad (stream cipher)
- garbled cipherbits only affect corresponding plain
- attacker can flip bits in ciphertext and corresponding bits in recovered plaintext will be flipped
- loss/add of cipher block, trashes decryption
- if attacker knows a cipherblock and plainblock, he can substitute his own plainblock m (a problem with all stream ciphers)

$p \oplus r = c$ if know p and c, then replace c with $c' = c \oplus p \oplus m$
when c' is decrypted ($\oplus$ r), you get m

Counter mode (CTR)

- Chaining (CBC) makes parallel speedups hard and hard to decrypt just Nth block (e.g. for disk)
- ECB can be done in parallel (weak)
- OFB pre-generate key stream, parallel
  but still have to generate N-1 sequences to
  decode N[th] block



counter mode

$E_k(\text{counter++}) \oplus p_i = c_i$

- counter should be BIG integer
- counter initialized to some value (IV)
- Can be done in parallell
- Can pre-generate sequence (stream cipher)
- permits direct decoding of block N
  (disk encryption)
- Errors
  - Garbled cipher block affect only one block
  - flip a cipher bit causes plaintext bit to flip
  - Add or lose a cipher block ??
- As secure as CFB, OFB, CBC
- also use as PRNG

Block-mode cipher summary

- ECB for single blocks (careful)
- CBC for multiple block (ECB and CBC need padding)
- stream/character based: OFB/CFB/CTR
- understand error properties
  - blocks re-ordered
  - error or modified cipher block (cipher bit flipped)
  - missing or duplicated/added block
- special requirements: parallel, disk encryption

encryption does not guarantee message integrity!

OpenSSL file encryption commands

- Takes care of password to key conversion, padding and salt
  - Output file larger and "gibberish" (binary data)
- Prepends 8 byte random salt – same file encrypted with same password will be "different"

```
openssl des-cbc –in letter.txt –out letter.des –k secret

openssl des-cbc –d –in letter.des –out tmp –k secret
```

- DES variants: des des-cbc des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb des-ede-ofb des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb des-ofb des3 desx
- Others: blowfish, AES, rc2, rc4, cast
- Key options: command line, file, or prompt -pass pass:secret
- Benchmark with speed command (or visit Crypto++ website)

OpenSSL encryption API DES

- Encrypted data is "gibberish", don't use str*()
- Encrypt output will be rounded up (0-padded) to multiple of 8 bytes
- May need to do IV chaining

```
#include <openssl/des.h>

        DES_cblock key, iv;
        DES_key_schedule sched;
        int r;
        char out[4096],in[4096], *str="123456789abcdefghij";

        DES_random_key(&key);
        r = DES_set_key_checked(&key,&sched);
        printf("r %d\n",r);
        strncpy(in,str,strlen(str)+1);
        //  DES_ncbc_ ... actually updates the iv, _cbc_ does not
        DES_cbc_encrypt(in,out, sizeof(out),&sched,&iv,DES_ENCRYPT);
        DES_cbc_encrypt(out,in, sizeof(out),&sched,&iv,DES_DECRYPT);
        printf("%s\n",in);
```


OpenSSL encryption – crypto agile (EVP)

```
#include <openssl/evp.h>
 EVP_CIPHER_CTX ctx;
 char key[EVP_MAX_KEY_LENGTH];
 char iv[EVP_MAX_IV_LENGTH];

 set up key and iv
EVP_EncryptInit (&ctx, EVP_des_cbc (), key, iv);
EVP_EncryptUpdate (ctx, out, &outlth, in, inlth);
EVP_EncryptFinal (ctx, out, &outlth);

EVP_DecryptInit (&ctx, EVP_des_cbc (), key, iv);
EVP_DecryptUpdate (ctx, out, &outlth, in, inlth);
EVP_DecryptFinal (&ctx, out, &outlth);
```

For more coding examples see ~dunigan/cns06, see OpenSSL tar file, see OpenSSL book website

Next time …

The next generation : AES

Stream ciphers

Key management

Lectures

1. Risk, viruses
2. UNIX vulnerabilities
3. Authentication & hashing
4. Random #s classical crypto
5. Block ciphers DES, RC5
6. AES, stream ciphers RC4, LFSR
7. **MIDTERM** ⊗
8. Public key crypto RSA, D-H
9. ECC, PKCS, ssh/pgp
10. PKI, SSL
11. Network vulnerabilities
12. Network defenses, IDS, firewalls
13. IPsec, VPN, Kerberos, secure OS
14. Secure coding, crypto APIs
15. review