CNS Lecture 5

Digital Encryption Standard (DES)
Other Feistel ciphers (Lucifer, blowfish, CAST)
Non-Feistel (IDEA, RCx)
Blocking (ECB, CBC, OFB, CTR)

Assignments 4 and 5 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

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 $\rightarrow$ machine/device $\rightarrow$ 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 APL performance

Cryptoolkit
- secret key crypto
- public key crypto
- big number math
- prime numbers
- hash functions

Attacks & Defenses

- Risk assessment
- Virus
- Unix security
- Authentication
- Network security
  - Firewalls, VPN, PAs, 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

- 3DES
- 3DES
- RC4
- S/MIME
- SSL
- Kerberos
- IPSec

You are here ...

- Attacks & Defenses
  - Risk assessment
  - Virus
  - Unix security
  - Authentication
  - Network security
    - Firewalls, VPN, PAs, 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
  - 3DES
  - 3DES
  - RC4
  - S/MIME
  - SSL
  - Kerberos
  - IPSec

Computers easily break classic schemes

Computers easily break classic schemes
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 hardware 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 ($K_1, K_2, \ldots$)
- 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
  $\oplus$ -- substitute and permute
- 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

Feistel cipher

Mangler function $F$ need not be reversible!

Decryption:
$$LD_0 = RD_0 = LE_{16} = RE_{16}$$
$$RD_1 = LD_0 \oplus F(RD_0, K_{16}) = RE_{16} \oplus F(RE_{15}, K_{16})$$
$$= (LE_{15} \oplus F(RE_{15}, K_{16})) \oplus F(RE_{15}, K_{16})$$
$$= LE_{15}$$

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 affect 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 b-halves (4 bits in, 2 bits out)
- key expansion at start

256-character alphabet (not just A-Z)
- even if alphabetic in, lot of non-alphabetic 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 = $IP^{-1}\left(f_{12}(SW(f_{12}(IP(plaintext))))\right)$

Plaintext = $IP^{-1}\left(f_{12}(SW(f_{12}(IP(ciphertext))))\right)$
S-DES

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

1010000010  10000 01100  00001 11000  10100100 K1
00100 00011  01000011 K2

Generating subkeys

10100100 K1

fK(L,R) = (L ⊕ 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

DES basics

block Feistel cipher (64 bits at a time)
— Using 8 plaintext bytes at a time (Playfair, Hill)
— word stream cipher (Caesar, one-time pad)
— use 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 (5 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 subkey generation

key schedule
— 64-bit key
— discard parity bits (8,16,...,64)
— permutation 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 k1, k2,...k16
— 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

Table 2.1: Initial Permutations (IP):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>08</td>
<td>18</td>
<td>01</td>
<td>06</td>
<td>09</td>
<td>0b</td>
</tr>
<tr>
<td>B</td>
<td>1e</td>
<td>0b</td>
<td>06</td>
<td>1d</td>
<td>13</td>
<td>09</td>
<td>06</td>
</tr>
<tr>
<td>C</td>
<td>05</td>
<td>14</td>
<td>03</td>
<td>02</td>
<td>13</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>D</td>
<td>1f</td>
<td>0e</td>
<td>07</td>
<td>15</td>
<td>08</td>
<td>1b</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>09</td>
<td>05</td>
<td>12</td>
<td>0c</td>
<td>11</td>
<td>00</td>
<td>0b</td>
</tr>
<tr>
<td>F</td>
<td>0a</td>
<td>0b</td>
<td>1a</td>
<td>0d</td>
<td>04</td>
<td>1c</td>
<td>1b</td>
</tr>
<tr>
<td>G</td>
<td>12</td>
<td>04</td>
<td>1c</td>
<td>0a</td>
<td>1b</td>
<td>0d</td>
<td>09</td>
</tr>
</tbody>
</table>

Table 2.2: Inverse Initial Permutations (IP^{-1}):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>08</td>
<td>18</td>
<td>01</td>
<td>06</td>
<td>09</td>
<td>0b</td>
</tr>
<tr>
<td>B</td>
<td>1e</td>
<td>0b</td>
<td>06</td>
<td>1d</td>
<td>13</td>
<td>09</td>
<td>06</td>
</tr>
<tr>
<td>C</td>
<td>05</td>
<td>14</td>
<td>03</td>
<td>02</td>
<td>13</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>D</td>
<td>1f</td>
<td>0e</td>
<td>07</td>
<td>15</td>
<td>08</td>
<td>1b</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>09</td>
<td>05</td>
<td>12</td>
<td>0c</td>
<td>11</td>
<td>00</td>
<td>0b</td>
</tr>
<tr>
<td>F</td>
<td>0a</td>
<td>0b</td>
<td>1a</td>
<td>0d</td>
<td>04</td>
<td>1c</td>
<td>1b</td>
</tr>
<tr>
<td>G</td>
<td>12</td>
<td>04</td>
<td>1c</td>
<td>0a</td>
<td>1b</td>
<td>0d</td>
<td>09</td>
</tr>
</tbody>
</table>

DES round -- encryption

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

\[ L_j = R_j \]
\[ R_j = L_j \oplus f(R_j, k_j) \]

Encryption: 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 \rightarrow 48 bits
* 48-bit subkey XOR’d with output of E
* S boxes (48 \rightarrow 32 bits)
  - 8 6-bit chunks to 4-bit chunks
  - S boxes are predefined tables -- how 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

Table 2.3: Definition of S-box values:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>02</td>
<td>0e</td>
<td>0b</td>
</tr>
<tr>
<td>B</td>
<td>0d</td>
<td>09</td>
<td>04</td>
<td>07</td>
</tr>
<tr>
<td>C</td>
<td>06</td>
<td>0a</td>
<td>0f</td>
<td>03</td>
</tr>
<tr>
<td>D</td>
<td>01</td>
<td>05</td>
<td>0c</td>
<td>08</td>
</tr>
</tbody>
</table>

DES S-box

S-box (substitution)
48 bits in \rightarrow 32 bits out, not reversible!
8 \times 64 S boxes (S_1, ..., S_8)
6 bit input -- outer two bits select row,
inner 4 bits select column \rightarrow 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 8-S boxes?
  - Where did numbers in S boxes come from?

DES strengths

* expansion permutation allows one bit to affect two substitutions
* avalanche -- dependency of output bit on input bit 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 bit

Table 2.4: Avalanche Criteria for DES:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>02</td>
<td>0e</td>
<td>0b</td>
</tr>
<tr>
<td>B</td>
<td>0d</td>
<td>09</td>
<td>04</td>
<td>07</td>
</tr>
<tr>
<td>C</td>
<td>06</td>
<td>0a</td>
<td>0f</td>
<td>03</td>
</tr>
<tr>
<td>D</td>
<td>01</td>
<td>05</td>
<td>0c</td>
<td>08</td>
</tr>
</tbody>
</table>
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)
- \(* \text{strict avalanche criterion} \) any output bit should change with probability 1/2 when any input bit is changed, for all \( i,j \)
- \(* \text{bit independence criterion} \) output 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)
- complement reduces key space 2^n

If y = DES(k)(x) then y = DES(*)(x)
- 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
- 2^10 guesses/yr
- 1000 years for a 1 Mip-yr processor
- 1 year for 1 Gips
- 1 day for 365 1 Gips processors
- EFF DES cracker (1998): 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
  - fewer than 16 rounds, sucplanch
  - greater than 16 rounds, more work than brute-force
  - 8 round attack: Lucifer 256 chosen plaintext, DES 2^n

- linear ('93)
  - linear approximation to action of block ciphers
  - XOR some plain and ciphertext together, get bit that is XOR of some key
  - recover 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: 6 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)
  - hash 25 iterations of modified DES is altered to thwart DES hardware cracker
  - Encode 64-bit output to 6-bit alphanumerics
- crypt command
  - 1 rotor enigma, polyalphabetic (256)
  - key guessing
  - crypt breaker workbench (cw)
  - compression helps

- setkey() encrypt() -- DES functions, key and message expressed as binary ASCII
- slow, 1 bit per byte
- strong encryption: PGP, ssh, OpenSSL, kerberos, cfe
- 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 * 10^10 mappings of 64-bit to 64-bits out of possible (2^56) = 10^{1030}
  - 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
Triple DES (3DES)

$E_k_1(D_k_2(E_k_1(m)))$

- 112-bit key
- compatible with DES if $k_1 = k_2$ thanks to EDE

$E_k_1(D_k_1(E_k_1(m))) = E_k_1(m)$

- 3 key: $E_k_3(D_k_2(E_k_1(m)))$
  - 168 key bits
  - compatible with DES if $k_1 = k_2$ or $k_2 = k_3$
  - 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)
- $D_{X,k}(m) = k_2 \oplus E_{k_1}(k_1 \oplus m)$
- 184-bit "key" ($56 + 64 + 64$)
- "whitening" keys $k_1$ and $k_2$
- 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 (5 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 8-boxes
- weak against differential attacks
- 8 round attack: Lucifer 256 chosen plaintext, DES 2¹⁴
- longer key is not sufficient

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
- variable secure key length up to 448 bits
- four 8x32-bit 5-boxes, 256 entries each
- key expansion builds 16 32-bit subkeys and four 6-boxes (521 executions of Blowfish)
- slow subkey generation makes it hard for rapid key switching, but makes brute force expensive
- initial value of 5-boxes and subkeys are 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

**Blowfish vs DES**

- S-boxes are key dependent, produced by repeated applications of a "changing" blowfish
- Subkey generation is very strong (but slow)
- Both halves of data mixed in each round
- Key of 448 bits, infeasible 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 affects every bit of R_i

Test of time?

**Twofish**

- Son of blowfish, AES candidate
- 128-bit block, 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
    - \( a' = a + b \)
    - \( b' = a + 2b \)
  - 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 function
- 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

**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
- 6 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

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
- 16 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

RC2

- non-Feistel (Rivest, '97)
- used in S/MIME
- optimized for 16-bit arithmetic, addition/subtraction, XOR, AND, complement, rotate
- 64-bit block
- 16 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

- parameterized block cipher, not Feistel
- select key size, rounds, block size
- RC5-32/16/10 (16-round, 64-bit key), 32-bit word worry about byte-order
- use XOR, rotate, add/subtract mod wordsize
- no substitutions
- security: data-dependent rotation
- key expansion uses addition/rotation, seeded with constants e and ϕ
- decryption: reverse rounds, rotation, and use subtraction
- twice as fast as DES
- licensed, in lots of RSA products

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 (log2 32)
  - Quadratic function (B (2B + 1))
  - Pre and post whitening steps

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 10/21/97, 250 days
- RC5-32/12/9 (64-bit key)
  - Found 8/12/02, 1757 days
  - 247 x10^6 keys/sec (max rate) 0.12% keystake/day
  - 351,252 participants tested 15,769,935,166,981,326,592 keys
- RC5-32/12/9 (72-bit key) ... work in progress

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

substitution and transposition

• confusion and diffusion
• easy if you have memory for 4x32 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

substitution and permutation

• performance (time/space) vs strength
• larger keys
• strong subkey generation
• large blocks
• simple operations, complex, non-linear functions (S-box, rotate)
• sensitive, more rounds
• resists 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-bit (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

• 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!

CBC

Figure 3.12 Cipher Block Chaining (CBC) mode
CBC – cyclic block chaining

- need IV (initialization vector) first iteration
- iterative, $E_k(c_{i-1}) = c_i$
- worry about padding last block
- use for long messages, files
- communicate IV (in the clear?)
- differing IV assures same plaintext produces different ciphertext
- 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 – cipher feedback

- need an IV
- cipherbyte fed into next step
- can do a byte (8-bit) at a time
- if cipherbytes are lost, CFB will re-sync, 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, sshv1, dealogin (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 – output feedback

- need an IV
- best used in 64-bit mode
- can pre-generate stream (a PRNG)
- XOR plaintext with OFB stream
- one-time pad (stream cipher)
- garbled cipherbyte only affects 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 plaintext, he can substitute his own plaintext block (a problem with all stream ciphers)

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

Counter mode (CTR)

- Chaining (CBC) makes parallel speedup hard and hard to decrypt just Nth block (e.g. disk)
- ECB can be done in parallel (weak)
- OFB pre-generate key stream, parallel but still have to generate N-1 sequences to decode Nth block
- counter mode
  - $c = E_k(counter++) \oplus p$
  - counter should be 32 integer
  - counter initialized to some value (IV)
  - can be done in parallel
  - pre-generate sequence (stream cipher)
  - permute absolute desizing of block N (data encryption)
  - Errors
    - garbled cipher block affect only one block
      - flip a cipher bit causees plaintext bit to flip
      - can be done in parallel
  - 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

- Take care of password to key conversion, padding and salt
  - Output file larger and "jibberish" (ASCII data)
- Prepare 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

  
  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 "jibberish", don’t use str[*]
- Encrypted output will be rounded up (0-padded) to multiple of 8 bytes
- May need to do IV chaining

```c
#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_cbc does not actually updates the iv, cfb 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)

```c
#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, renovation
2. UNIX vulnerabilities
3. Authentication & hashing
4. Random:
   1. classical crypto
   2. Block ciphers DES, AES
5. Stream ciphers RC4, LFSR
6. MIDCUM
7. Public key crypto RSA, D-H
8. ECC, PASS, signature
9. PKI, SSL
10. Network vulnerabilities
11. Network defenses, IDS, firewalls
12. IPv6, VPN, Kerberos, secure OS
13. Secure coding, crypto APIs
14. Review