

Blowfish & RC5

Blowfish

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
 - fast implementation on 32-bit CPUs
 - compact in use of memory
 - simple structure eases analysis/implementation
 - variable security by varying key size
- has been implemented in various products

Blowfish Key Schedule

- uses a 32 to 448 bit key
- used to generate
 - 18 32-bit subkeys stored in K-array K_j
 - four 8x32 S-boxes stored in $S_{i,j}$
- key schedule consists of:
 - initialize P-array and then 4 S-boxes using pi
 - XOR P-array with key bits (reuse as needed)
 - loop repeatedly encrypting data using current P & S and replace successive pairs of P then S values
 - requires 521 encryptions, hence slow in rekeying

Blowfish Encryption

- uses two primitives: addition & XOR
- data is divided into two 32-bit halves L_0 & R_0

for $i = 1$ to 16 do

$$R_i = L_{i-1} \text{ XOR } P_i;$$

$$L_i = F[R_i] \text{ XOR } R_{i-1};$$

$$L_{17} = R_{16} \text{ XOR } P_{18};$$

$$R_{17} = L_{16} \text{ XOR } P_{17};$$

- where

$$F[a, b, c, d] = ((S_{1,a} + S_{2,b}) \text{ XOR } S_{3,c}) + S_{4,d}$$

Discussion

- key dependent S-boxes and subkeys, generated using cipher itself, makes analysis very difficult
- changing both halves in each round increases security
- provided key is large enough, brute-force key search is not practical, especially given the high key schedule cost

RC5

- a proprietary cipher owned by RSADSI
- designed by Ronald Rivest (of RSA fame)
- used in various RSADSI products
- can vary key size / data size / no rounds
- very clean and simple design
- easy implementation on various CPUs
- yet still regarded as secure

RC5 Ciphers

- RC5 is a family of ciphers RC5-w/r/b
 - w = word size in bits (16/32/64) nb data=2w
 - r = number of rounds (0..255)
 - b = number of bytes in key (0..255)
- nominal version is RC5-32/12/16
 - ie 32-bit words so encrypts 64-bit data blocks
 - using 12 rounds
 - with 16 bytes (128-bit) secret key

RC5 Key Expansion

- RC5 uses $2r+2$ subkey words (w -bits)
- subkeys are stored in array $S[i]$, $i=0..t-1$
- then the key schedule consists of
 - initializing S to a fixed pseudorandom value, based on constants e and ϕ
 - the byte key is copied (little-endian) into a c -word array L
 - a mixing operation then combines L and S to form the final S array

RC5 Encryption

- split input into two halves A & B

$$L_0 = A + S[0];$$

$$R_0 = B + S[1];$$

for $i = 1$ to r do

$$L_i = ((L_{i-1} \text{ XOR } R_{i-1}) \lll R_{i-1}) + S[2 \times i];$$

$$R_i = ((R_{i-1} \text{ XOR } L_i) \lll L_i) + S[2 \times i + 1];$$

- each round is like 2 DES rounds
- note rotation is main source of non-linearity
- need reasonable number of rounds (eg 12-16)

RC5 Modes

- RFC2040 defines 4 modes used by RC5
 - RC5 Block Cipher, is ECB mode
 - RC5-CBC, is CBC mode
 - RC5-CBC-PAD, is CBC with padding by bytes with value being the number of padding bytes
 - RC5-CTS, a variant of CBC which is the same size as the original message, uses ciphertext stealing to keep size same as original

Block Cipher Characteristics

- features seen in modern block ciphers are:
 - variable key length / block size / no rounds
 - mixed operators, data/key dependent rotation
 - key dependent S-boxes
 - more complex key scheduling
 - operation of full data in each round
 - varying non-linear functions

Stream Ciphers

- process the message bit by bit (as a stream)
- typically have a (pseudo) random **stream key**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message
 - $C_i = M_i \text{ XOR } \text{StreamKey}_i$
- what could be simpler!!!!
- but must never reuse stream key
 - otherwise can remove effect and recover messages

Stream Cipher Properties

- some design considerations are:
 - long period with no repetitions
 - statistically random
 - depends on large enough key
 - large linear complexity
 - correlation immunity
 - confusion
 - diffusion
 - use of highly non-linear boolean functions

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher
- given a key k of length l bytes

```
for i = 0 to 255 do
```

```
    S[i] = i
```

```
j = 0
```

```
for i = 0 to 255 do
```

```
    j = (j + S[i] + k[i mod l]) (mod 256)
```

```
    swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- tXOR with next byte of message to en/decrypt

$i = j = 0$

for each message byte M_i

$i = (i + 1) \pmod{256}$

$j = (j + S[i]) \pmod{256}$

swap($S[i]$, $S[j]$)

$t = (S[i] + S[j]) \pmod{256}$

$C_i = M_i \text{ XOR } S[t]$

RC4 Security

- claimed secure against known attacks
 - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- have a concern with WEP, but due to key handling rather than RC4 itself

Summary

- have considered:
 - some other modern symmetric block ciphers
 - Triple-DES
 - Blowfish
 - RC5
 - briefly introduced stream ciphers
 - RC4