Diyala University
College of Engineering
Computer & Software
Engineering Department

Fourth Year 2012/2013

Cryptography Algorithms

Chapter 2/Part4

PRESENTED BY

DR. ALI J. ABBOUD

Replacement of DES

- Examine in this lecture some of the most important symmetric ciphers currently in use
  - Triple DES
  - Block cipher operation
  - RC5
  - RC4
- DES is vulnerable nowadays to a brute-force attack
- It is replaced as a standard by AES
- There has been interest to provide another algorithm during the transition to AES
- Also interest to preserve the existing investment in software and hardware, increasing the security
- **Solution:** *iterated version of DES*

Double DES

Obvious solution: double DES

- Use two keys K1, K2 and encrypt the text using the two keys

  $$C=E_{K2}(E_{K1}(P))$$

- To decrypt simply use DES decryption twice

  $$P=D_{K1}(D_{K2}(C))$$

- The scheme involves now a key of 112 bits which should make it much more secure than DES, at least in principle

Double DES

Multiple encryption

- Source: W.Stallings – "Cryptography and network security" (5th edition), Fig 6.1

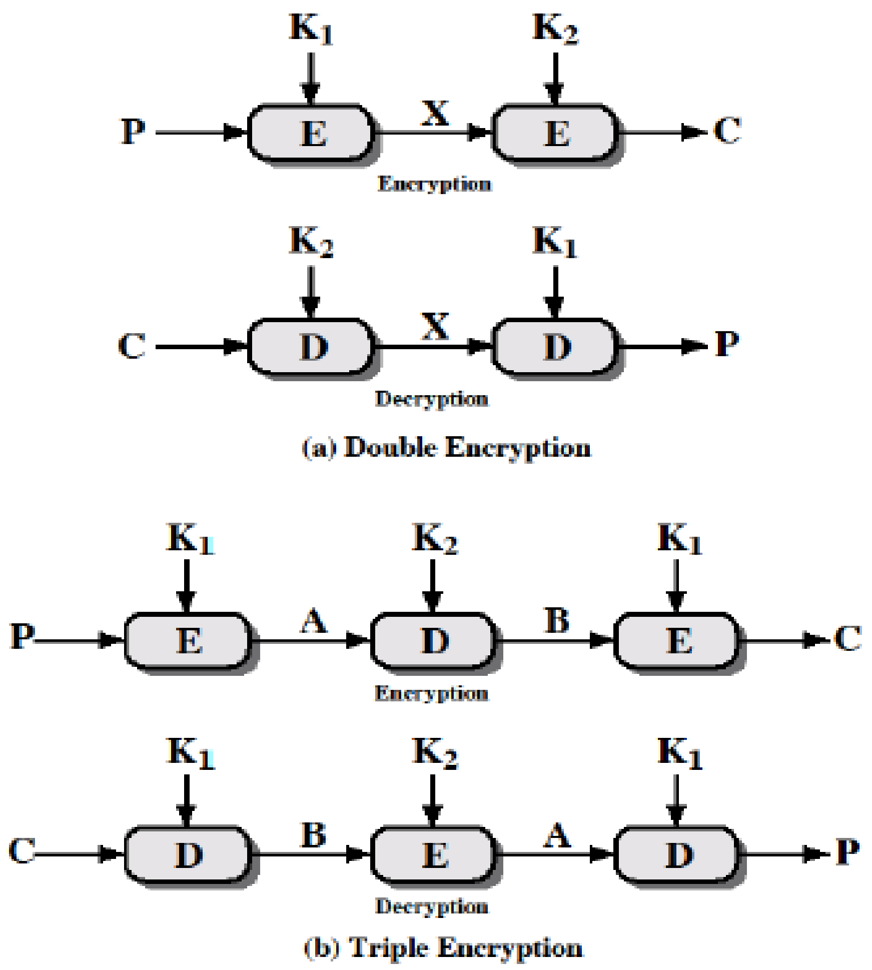


Figure 6.1 Multiple Encryption

Security of Double DES

- **Question 1**: Is double DES more than just DES?
  - In other words, could there be a key K_3 such that $E_{K2}(E_{K1}(P)) = E_{K3}(P)$?
- **Answer:** Double DES cannot be reduced to single DES (intuitively clear, difficult to prove though)
- **Question 2**: Is double DES more secure than DES?
  - Key is twice as long
  - Does this imply necessarily that all attacks must take significantly more time than a DES attack?
- **Answer**: Double DES can be broken with only twice the effort of breaking DES with a brute-force attack
  - *Meet-in-the-middle attack*

Man in the Middle Attacks on the Double DES

- Due to Diffie, Hellman (1977)
- Main observation: if $C=E_{K2}(E_{K1}(P))$, then

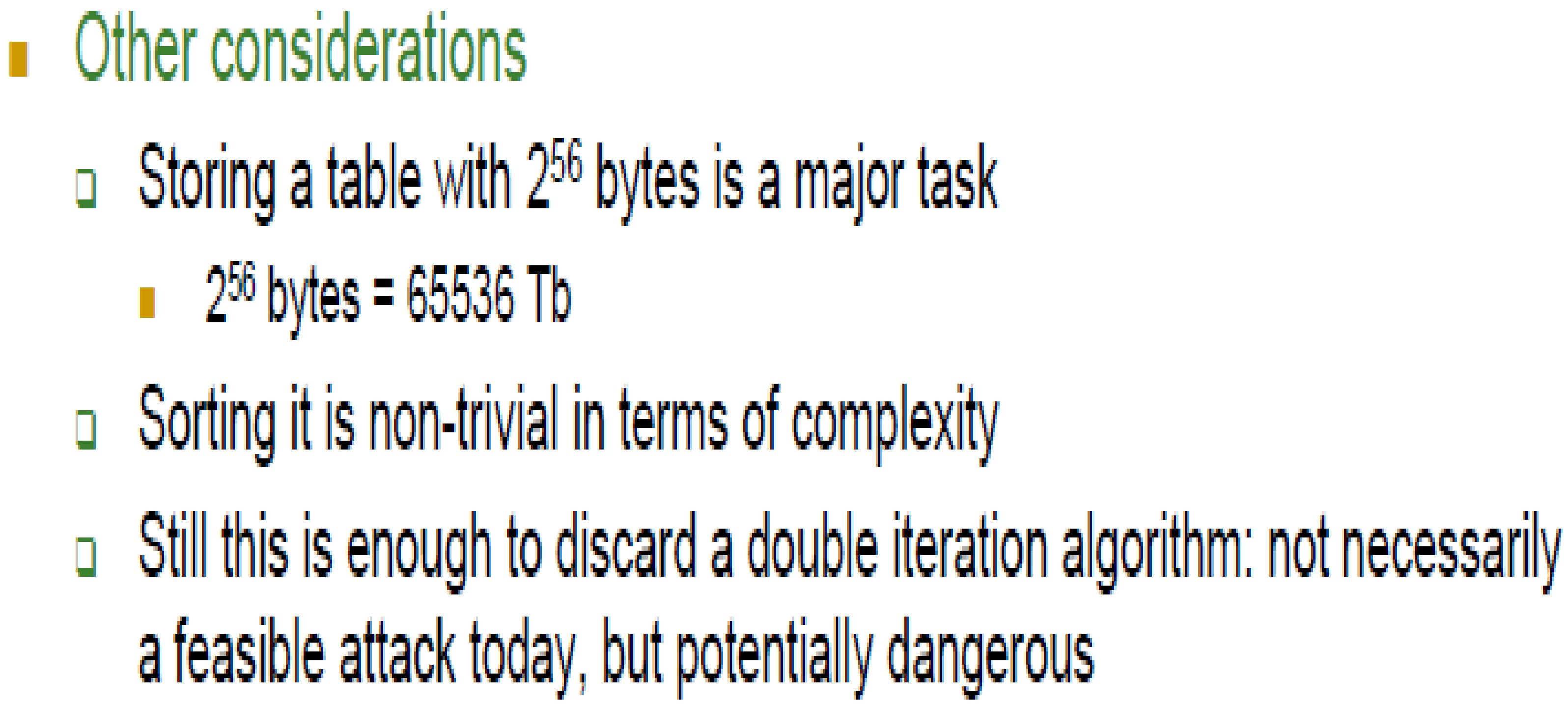
$$E_{K1}(P)=D_{K2}(C)=X$$

- Assume we have two pairs of plaintext-ciphertext (of only one byte each!)
- Given a known pair (P,C) attack as follows:
  - Encrypt P for all 2^{56} possible keys K_1
  - Store the results in a table and sort the table by the values of X
  - Decrypt C using all possible 2^{56} possible keys K_2
    - For each decryption check the result in the table
    - In case of match, either we have the answer, or a false positive
    - Test the two keys with the second pair of plaintext-ciphertext: if they match, the correct keys were found

Man in the Middle Attacks on the Double DES

- Short analysis:
  - For a given plaintext P, there are 2^{64} possible ciphertext values that could be produced by 2DES
  - 2DES uses 112-bit key so that there 2^{112} possible keys
  - Thus, on average the number of different keys that give the same C for a given P is $2^{112}/2^{64}=2^{48}$: thus, the attacker will have 2^{48} "false alarms"
  - Having the second pair of plaintext-ciphertext the false alarm rate is reduced to $2^{48-64}=2^{-16}$
- **Conclusion:** 2DES is broken in 2^{56} steps with probability larger than $1-2^{-16}$
  - **The effort is not much bigger than the 2^{55} required to break DES**

Man in the Middle Attacks on the Double DES



- Other considerations
  - Storing a table with 2^{56} bytes is a major task
    - 2^{56} bytes = 65536 Tb
  - Sorting it is non-trivial in terms of complexity
  - Still this is enough to discard a double iteration algorithm: not necessarily a feasible attack today, but potentially dangerous

Tripple DES

- Counter to the meet-in-the-middle attack: use three stages of encryption with three different keys

$$C=E_{K3}(E_{K2}(E_{K1}(P)))$$

  - *Drawback*: keys is now 168 bits which makes it slower
  - *Alternative*: 3DES with 2 keys: K3=K1
- Another alternative: instead of three encryptions, use 2 encryptions and one decryption

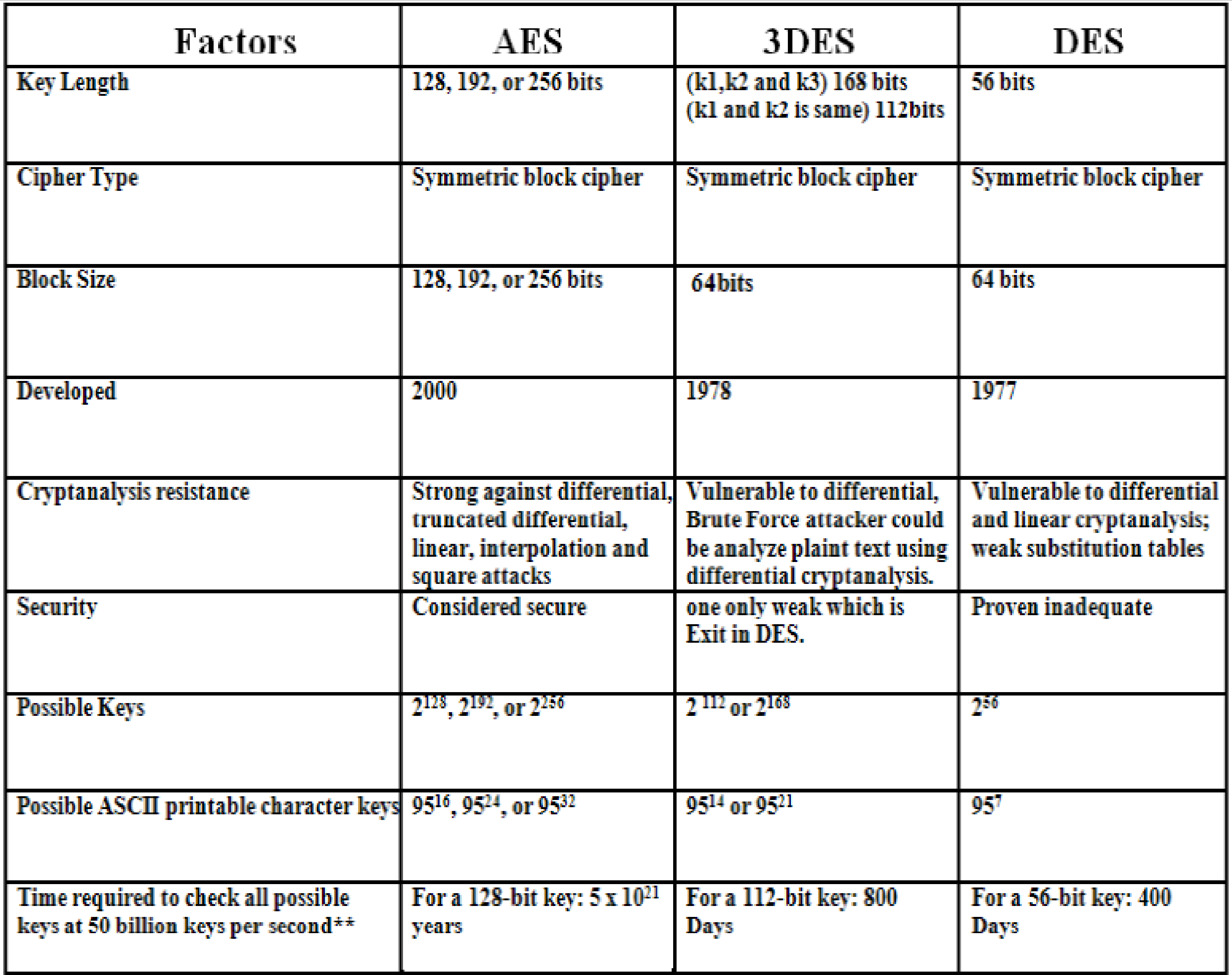
$$C=E_{K1}(D_{K2}(E_{K3}(P)))$$

  - Using decryption rater than encryption in the second stage yields no weakness
  - This is only a solution to remain compatible with the users of simple DES. Such users should use K1=K2:

$$C=E_{K1}(D_{K1}(E_{K3}(P)))=E_{K3}(P)$$

- There are no practical cryptoanalytical attacks on 3DES
  - Brute-force attack is on the scale of 2^{112}, i.e., more than 5×10^{33}
  - Differential cryptanalysis exceeds an effort of 10^{52}
  - Best attack (Lucks, 1998): 2^{32} known plaintexts, 2^{113} steps, 2^{90} single DES encryptions, 2^{88} bytes memory $\rightarrow$ not practical
- ***Suggestion***: given the current state of technology, 3DES should be used with 3 keys

Comparison



Factors	AES	3DES	DES
Key Length	128, 192, or 256 bits	(k1,k2 and k3) 168 bits (k1 and k2 is same) 112bits	56 bits
Cipher Type	Symmetric block cipher	Symmetric block cipher	Symmetric block cipher
Block Size	128, 192, or 256 bits	64bits	64 bits
Developed	2000	1978	1977
Cryptanalysis resistance	Strong against differential, truncated differential, linear, interpolation and square attacks	Vulnerable to differential, Brute Force attacker could be analyze plaint text using differential cryptanalysis.	Vulnerable to differential and linear cryptanalysis; weak substitution tables
Security	Considered secure	one only weak which is Exit in DES.	Proven inadequate
Possible Keys	2^{128}, 2^{192}, or 2^{256}	2^{112} or 2^{168}	2^{56}
Possible ASCII printable character keys	95^{16}, 95^{24}, or 95^{32}	95^{14} or 95^{21}	95^{7}
Time required to check all possible keys at 50 billion keys per second**	For a 128-bit key: 5 x 10^{21} years	For a 112-bit key: 800 Days	For a 56-bit key: 400 Days

RC 4 and RC5 Algorithms

RC5, RC4

- Before going into AES: RC5, RC4
  - RC5: another symmetric cipher proposed while DES was under replacement
  - RC4: a special-purpose stream cipher

- *Variable key length / block size / number of rounds* – RC5
- *Mixed operators*: use of more than one arithmetic and/or Boolean operator is advisable, especially if these operator are not distributive and associative – RC5
  - Modular addition/subtraction, XOR
- *Data-dependent (and key-dependent) rotation* – an alternative to S-boxes – RC5
- *Key-dependent S-boxes* – Blowfish
- *Lengthy key scheduling* – generation of subkeys takes much longer than encryption and decryption (brute-force attacks become much more difficult) – Blowfish
- Operation on full data (rather than one half as in Feistel) in each round – Blowfish, RC5, AES
- *Varying non-linear functions* – the encryption function varies from round to round

RC5 Algorithm

- Symmetric encryption algorithm developed by Rivest 1994; incorporated into BSAFE, JSAFE, S/MAIL (of RSA Data Security Inc.)
- Characteristics of RC5 (see http://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf)
  - Suitable for hardware and software: uses only common operations found on microprocessors
  - Fast: simple and word oriented
  - Adaptable to processors of different word lengths: the number of bits in the word is the 1st parameter of RC5
  - Variable number of rounds: number of rounds is the 2nd parameter
  - Variable-length key: key length is the 3rd parameter of RC5
  - Simple: easy to implement and analyze
  - Low memory requirement: suitable for smart cards or other devices with limited memory
  - High security
  - Data-dependent rotations

RC5 Algorithm

- Parameters
  - w is the word size in bits – RC5 encrypts blocks of 2 words. Allowed values: 16, 32, 64
  - r is the number of rounds. Allowed values: 0,1,…,255
  - b is the number of 8-bit bytes in the secret key K. Allowed values: 0,1,…,255
- A specific version of RC5 is denoted RC5-w/r/b
  - The author advises to use RC5-32/12/16 as the “nominal” version
  - That means: 64-bit plaintext/ciphertext blocks, 12 rounds, 128-bit key
- Algorithm
  - Key expansion
  - Input manipulation
- Details are on the following slides:
  - where addition and subtraction (+ and -) are modulo 2^w
  - bitwise XOR is $\oplus$
  - x<<<y is the circular left-shift of x by y bits
  - x>>>y is the circular right-shift of word x by y bits

RC5 Algorithm

RC5 encryption/decryption scheme

RC5-w/r/b – **encryption**

$LE_0=A+S[0]$
$RE_0=B+S[1]$
For i=1 to r do
    $LE_i=((LE_{i-1}\oplus RE_{i-1})<<<RE_{i-1})+S[2i]$
    $RE_i=((RE_{i-1}\oplus LE_i)<<<LE_i)+S[2i+1]$

RC5 – decryption

For i=r downto 1 do
    $RD_{i-1}=((RD_i-S[2i+1])>>>LD_i)\oplus LD_i$
    $LD_{i-1}=((LD_i-S[2i])>>>Rd_{i-1})\oplus Rd_{i-1}$
$B=RD_0-S[1]$
$A=LD_0-S[0]$

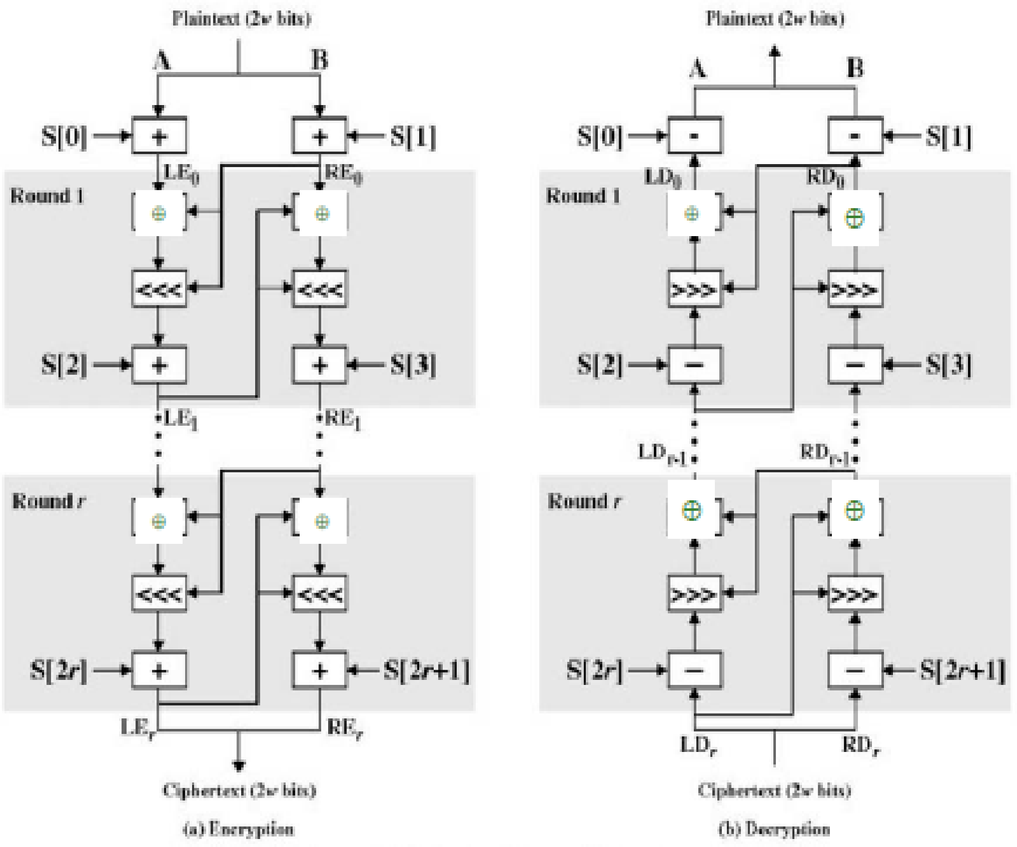


Figure 6.6 RC5 Encryption and Decryption

RC5 Algorithm: Key Expansion

- Two subkeys are used in each round, two subkeys are used in an additional operation
  - We need 2r+2 subkeys; let t=2r+2
- Each subkey is one word (w bits) in length
- The subkeys are generated in the array S
  - S is initialized with a certain pattern (details on the next slide)
  - The b-byte key K[0…b-1] is converted into an array of words K'[0…c-1]
    - Content of K is simply copied to K'; if b is not a multiple of w, fill in with 0
    - Nominal version: key has 128 bits, words have 32 bits: K' will have c=4 elements
    - Recall: they key K has b*8 bits, a word has w bits
    - c is the smallest integer larger than b*8/w
  - Mix K' with S to produce the final values for the array S (details on the next slide)

RC5 Algorithm: Key Expansion

- Initialize S as follows:

```
S[0]=$P_w$
For i=1 to t-1 do S[i]=S[i-1]+$Q_w$
```

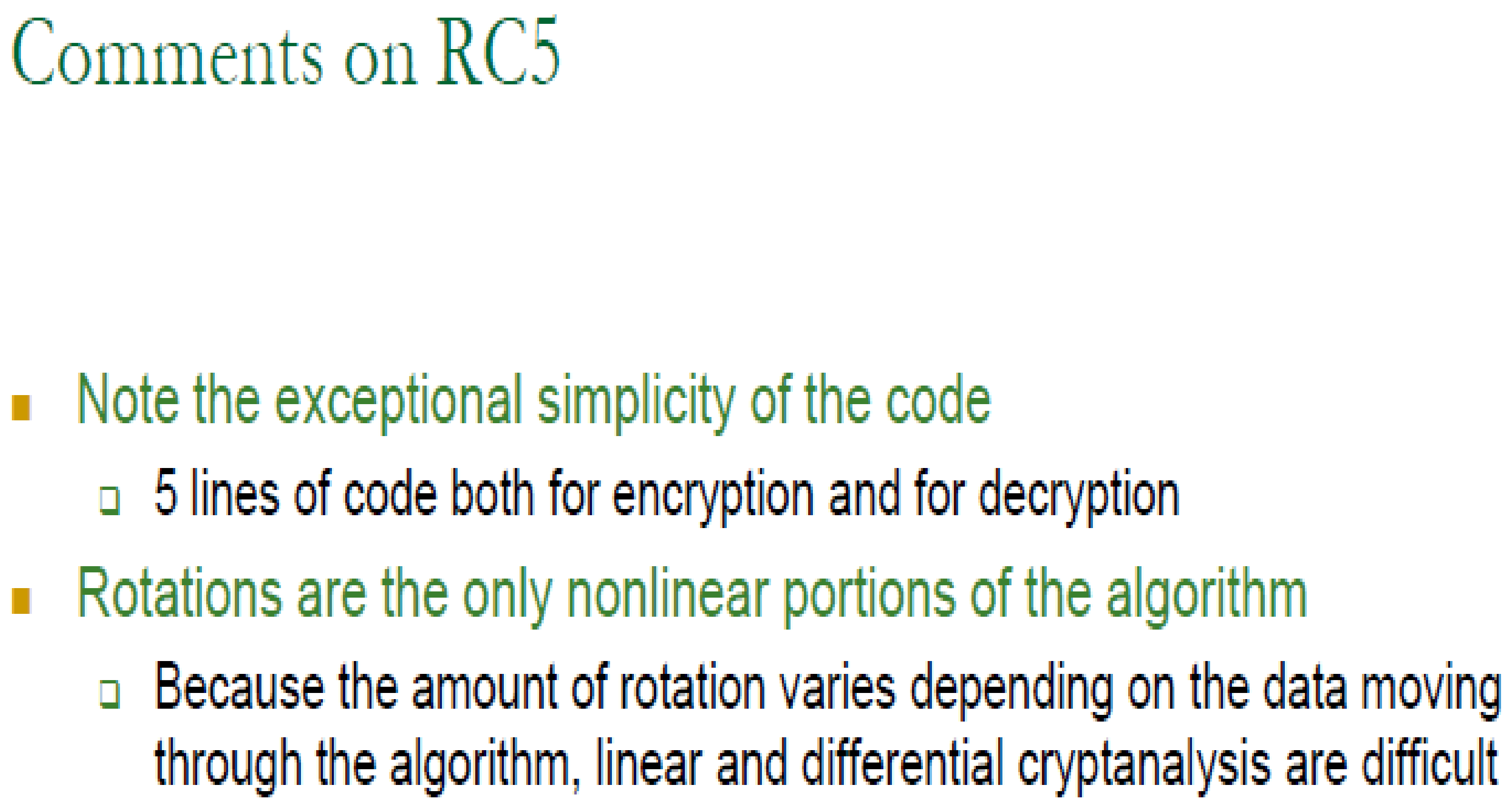
  - Constants P_w and Q_w (on w bits) are defined as P_w=**Odd**[(e-2)2^w], Q_w=**Odd**[(ϕ-1)2^w], where **Odd** gives the closest odd integer to its output, **e** is the base of the natural logarithm and ϕ is the golden ration (**ϕ=(1+√5)/2**. Values are given in the table bellow (in hexadecimal)

w	16	32	64
P_w	B7E1	B7E15163	B7E151628AED2A6B
Q_w	9E37	9E3779B9	9E3779B97F4A7C15

- Mix array S with the key array K' to produce the final values for S:

```
i=j=X=Y=0;          t=2r+2
Do 3 x max(t,c) times:
    X=S[i]=(S[i]+X+Y)<<<3;
    Y=K'[j]=(K'[j]+X+Y)<<<(X+Y);
    i=(i+1) mod t;
    j=(j+1) mod c;
```

RC5 Algorithm: Key Expansion



Comments on RC5

- Note the exceptional simplicity of the code
  - 5 lines of code both for encryption and for decryption
- Rotations are the only nonlinear portions of the algorithm
  - Because the amount of rotation varies depending on the data moving through the algorithm, linear and differential cryptanalysis are difficult

RC4 Stream Cipher

- This is the most popular symmetric stream cipher
- Designed by Rivest for RSA Security
- Used in SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards for secure communication between Web browsers and servers
- Used in WEP, part of the IEEE 802.11 wireless LAN standard
- RC4 was kept as a trade secret by RSA Inc but got anonymously posted on the Internet in 1994

RC4 Stream Cipher Structure

- Process the message byte by byte (as a stream)
- Typically have a (pseudo) random **stream key** that is XORed with plaintext bit by bit
- Randomness of **stream key** completely destroys any statistically properties in the message
  - C_i = P_i XOR $StreamKey_i$
- The simplest encryption/decryption algorithm possible!
- A stream cipher is similar to the one-time pad discussed a few lectures back
  - The difference is that a one-time pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream generated based on a secret key
- One must never reuse stream key
  - Otherwise can remove effect and recover messages
  - XOR two ciphertexts obtained with the same key stream to obtain the XOR of the plaintexts – enough to know about the structure of the files to effectively attack them

RC4 Stream Cipher Structure

Stream cipher structure

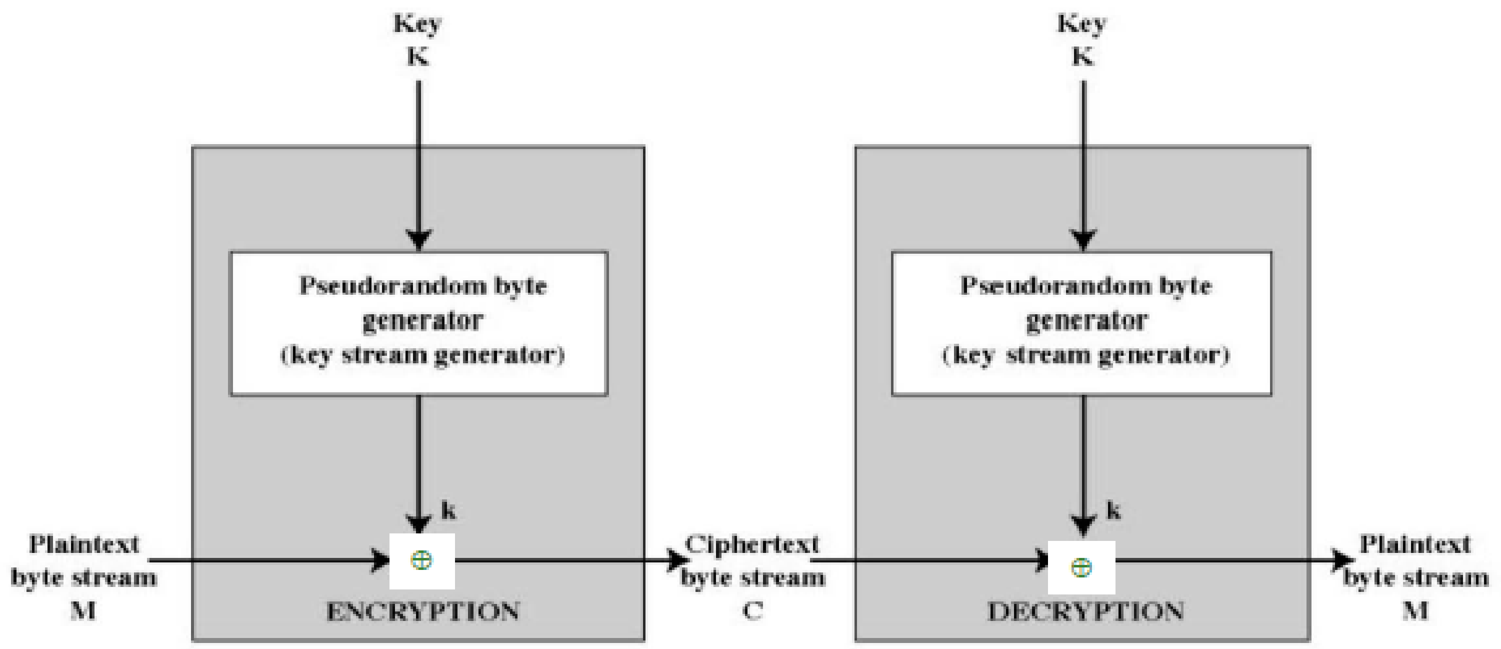
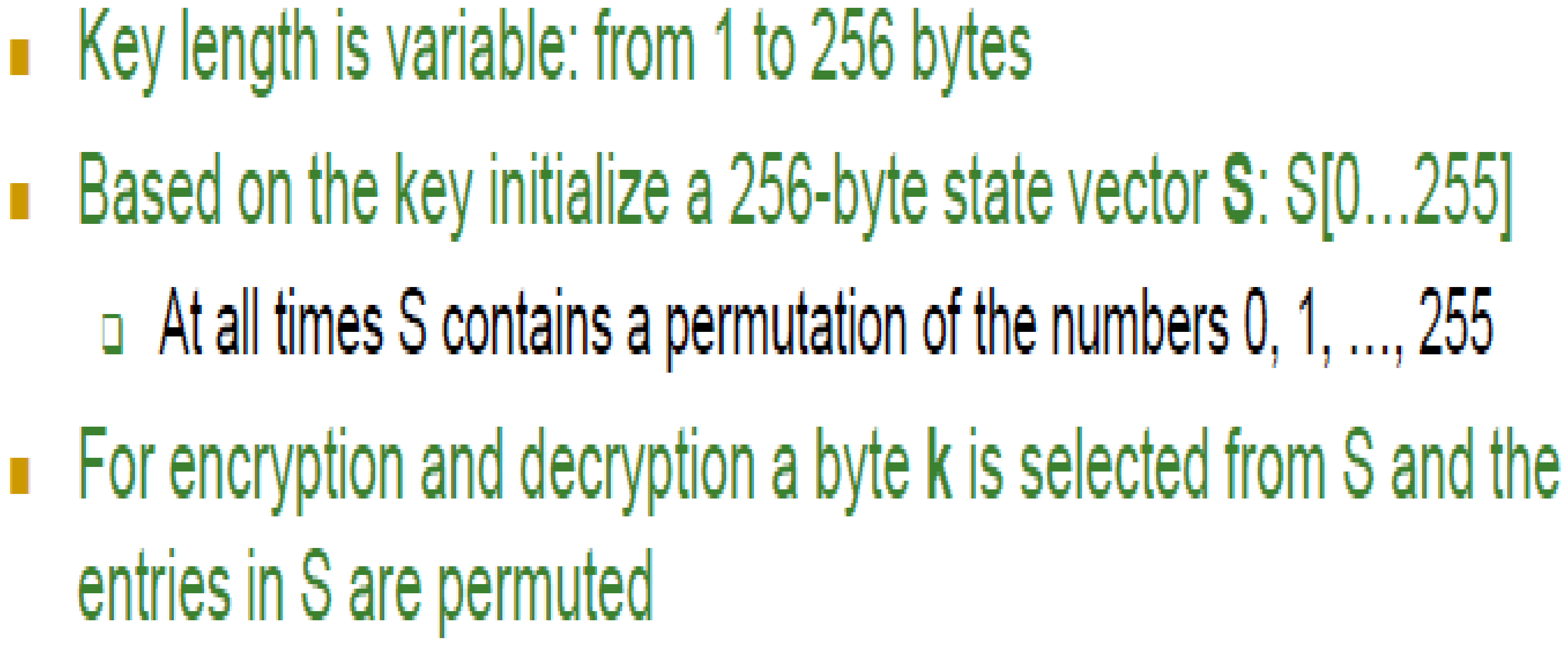


Figure 6.8 Stream Cipher Diagram

RC4 Stream Cipher Design

- Key stream should have a large period – a pseudorandom number generator uses a function that produces a **deterministic** stream of bits that eventually repeats
- Key stream should approximate the properties of a true random number generator
  - Same frequency of 0 and 1
  - If treated as a stream of bytes, all 255 values should occur with the same frequency
- Key should be long enough to protect against brute-force attack
  - At least 128 bits
- **Advantage** over block ciphers: generating the stream key is much **faster** than encrypting and decrypting and less code is needed

RC4 Stream Algorithm



- Key length is variable: from 1 to 256 bytes
- Based on the key initialize a 256-byte state vector **S**: S[0…255]
  - At all times S contains a permutation of the numbers 0, 1, …, 255
- For encryption and decryption a byte **k** is selected from S and the entries in S are permuted

RC4 – initialization of S

- Initially S[i]=i, i=0,1,…,255 and create a temporary vector T of length 256 – the key K is copied to T (if K has less than 256 bytes, repeat K as many times as necessary to fill T)

```
For i=0 to 255 do
    S[i]=i;
    T[i]=K[i mod keylen]
```

  - Input key is never used after this initialization
- Use T to produce the initial permutation of S

```
j=0;
For i=0 to 255 do
    j=(j+S[i]+T[i]) mod 256;
    Swap(S[i],S[j]);
```

RC4 – stream generation

- RC4 algorithm (key generation):

```
i,j=0;
While(true)
    i=(i+1) mod 256;
    j=(j+S[i]) mod 256;
    Swap(S[i],S[j]);
    t=(S[i]+S[j]) mod 256;
    k=S[t];
```

- *Encryption*: XOR k with the next byte of the plaintext
- *Decryption*: XOR k with the next byte of the ciphertext
- There is no practical attack against RC4 with reasonable key length such as 128 bits
- **Strength of RC4**: there has been a report of a problem in the WEP protocol (for 802.11 wireless LAN) – the problem is not with RC4 but rather with the way in which keys are generated to use as input to RC4
    - Can be easily fixed

Blowfish Algorithm

- 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/implemention
  - variable security by varying key size
- has been implemented in various products

Blowfish Algorithm

- Blowfish is a keyed, symmetric block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products. *(Wikipedia)*
- Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. *(Bruce Schneier)*
- Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms.
- It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.

Blowfish Algorithm

- Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm. Blowfish is unpatented and license-free, and is available free for all uses.
- While no effective cryptanalysis of Blowfish has been found to date, more attention is now given to block ciphers with a larger block size, such as AES or Twofish.
- At the time, many other designs were proprietary, encumbered by patents or kept as government secrets.
- Schneier has stated that, "Blowfish is unpatented, and will remain so in all countries. The algorithm is hereby placed in the public domain, and can be freely used by anyone.

Blowfish Algorithm

- The original Blowfish paper was presented at the First Fast Software Encryption workshop in Cambridge, UK (proceedings published by Springer-Verlag, Lecture Notes in Computer Science #809, 1994) and the April 1994 issue of Dr. Dobb's Journal.
- "Blowfish--One Year Later" appeared in the September 1995 issue of Dr. Dobb's Journal.

Blowfish Algorithm

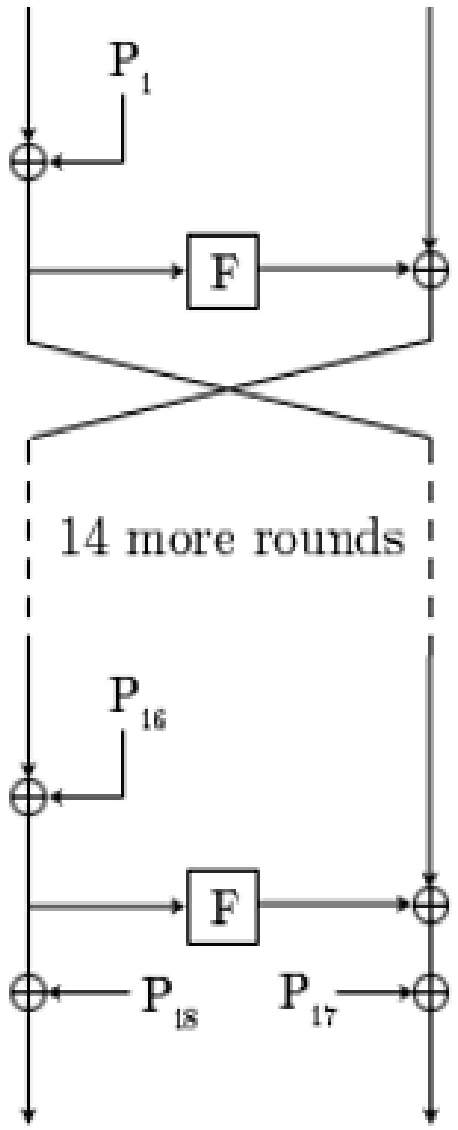
- **There are two parts to this algorithm;**
  - A part that handles the expansion of the key.
  - A part that handles the encryption of the data.
- **The expansion of the key:** break the original key into a set of subkeys. Specifically, a key of no more than 448 bits is separated into 4168 bytes. There is a P-array and four 32-bit S-boxes. The P-array contains 18 32-bit subkeys, while each S-box contains 256 entries.
- **The encryption of the data:** 64-bit input is denoted with an x, while the P-array is denoted with a Pi (where i is the iteration).

Blowfish Algorithm

- Blowfish has a 64-bit block size and a key length of anywhere from 32 bits to 448 bits (32-448 bits in steps of 8 bits; default 128 bits).
- It is a 16-round Feistel cipher and uses large key-dependent S-boxes. It is similar in structure to CAST-128, which uses fixed S-boxes.

Blowfish Algorithm

- The diagram to shows the action of Blowfish. Each line represents 32 bits. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes.
- The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries.



Blowfish Algorithm

- Initialize the P-array and S-boxes
- XOR P-array with the key bits. For example, P1 XOR (first 32 bits of key), P2 XOR (second 32 bits of key), ...
- Use the above method to encrypt the all-zero string
- This new output is now P1 and P2
- Encrypt the new P1 and P2 with the modified subkeys
- This new output is now P3 and P4
- Repeat 521 times in order to calculate new subkeys for the P-array and the four S-boxes

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

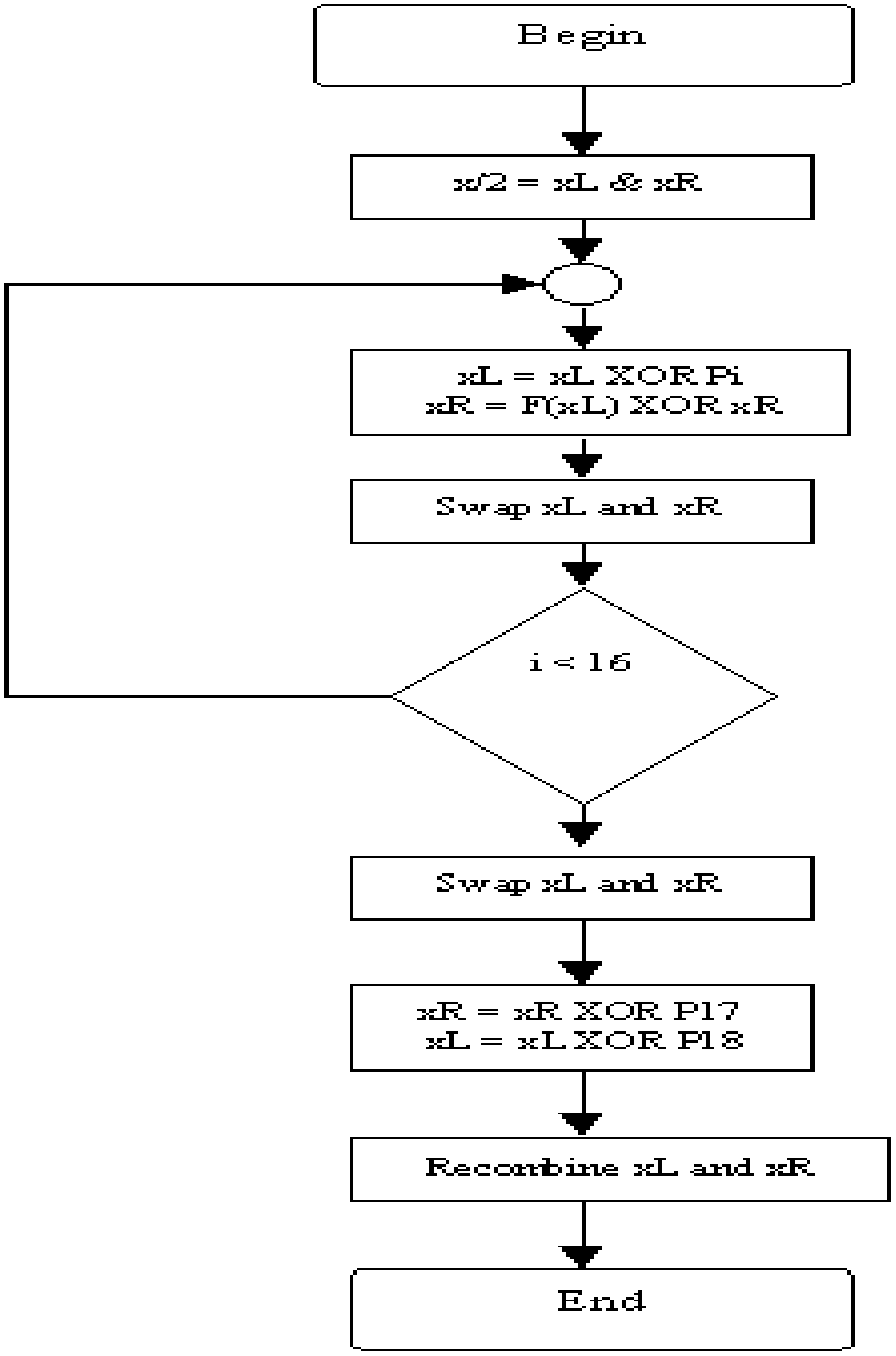
```
for $i$ = 1 to 16 do
    $R_i = L_{i-1}$ XOR $P_i$;
    $L_i = F[R_i]$ XOR $R_{i-1}$;
$L_{17} = R_{16}$ XOR $P_{18}$;
$R_{17} = L_{16}$ XOR $i_{17}$;
```

- where

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

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


Begin
x/2 = xL & xR
xL = xL XOR Pi
xR = F(xL) XOR xR
Swap xL and xR
i < 16
Swap xL and xR
xR = xR XOR P17
xL = xL XOR P18
Recombine xL and xR
End

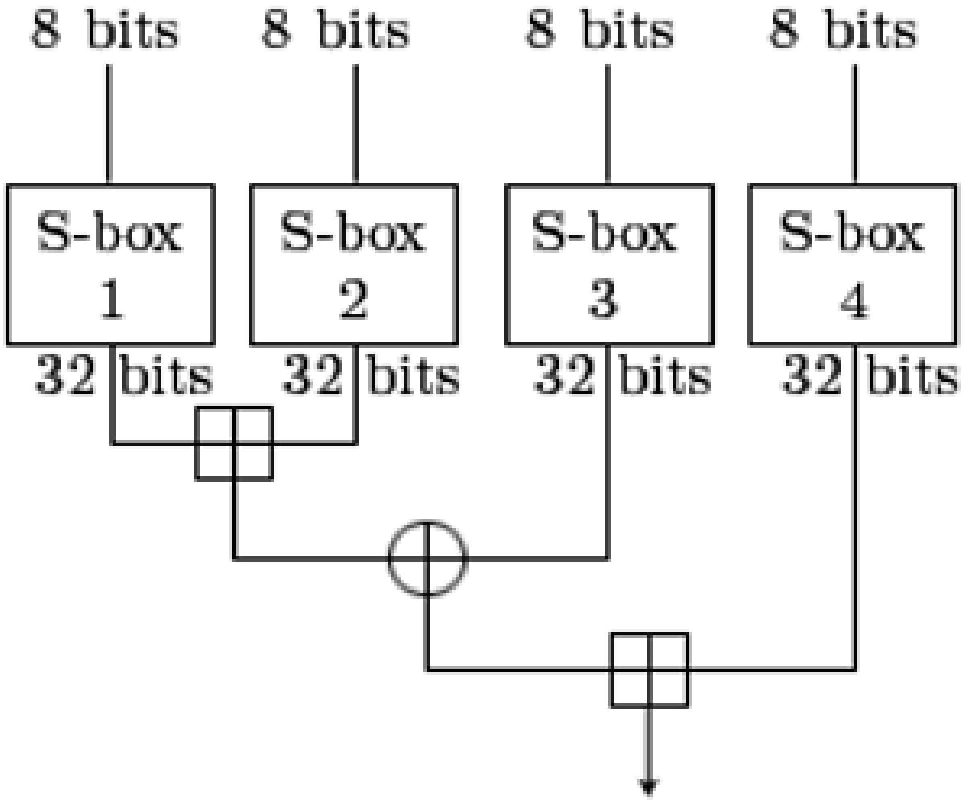
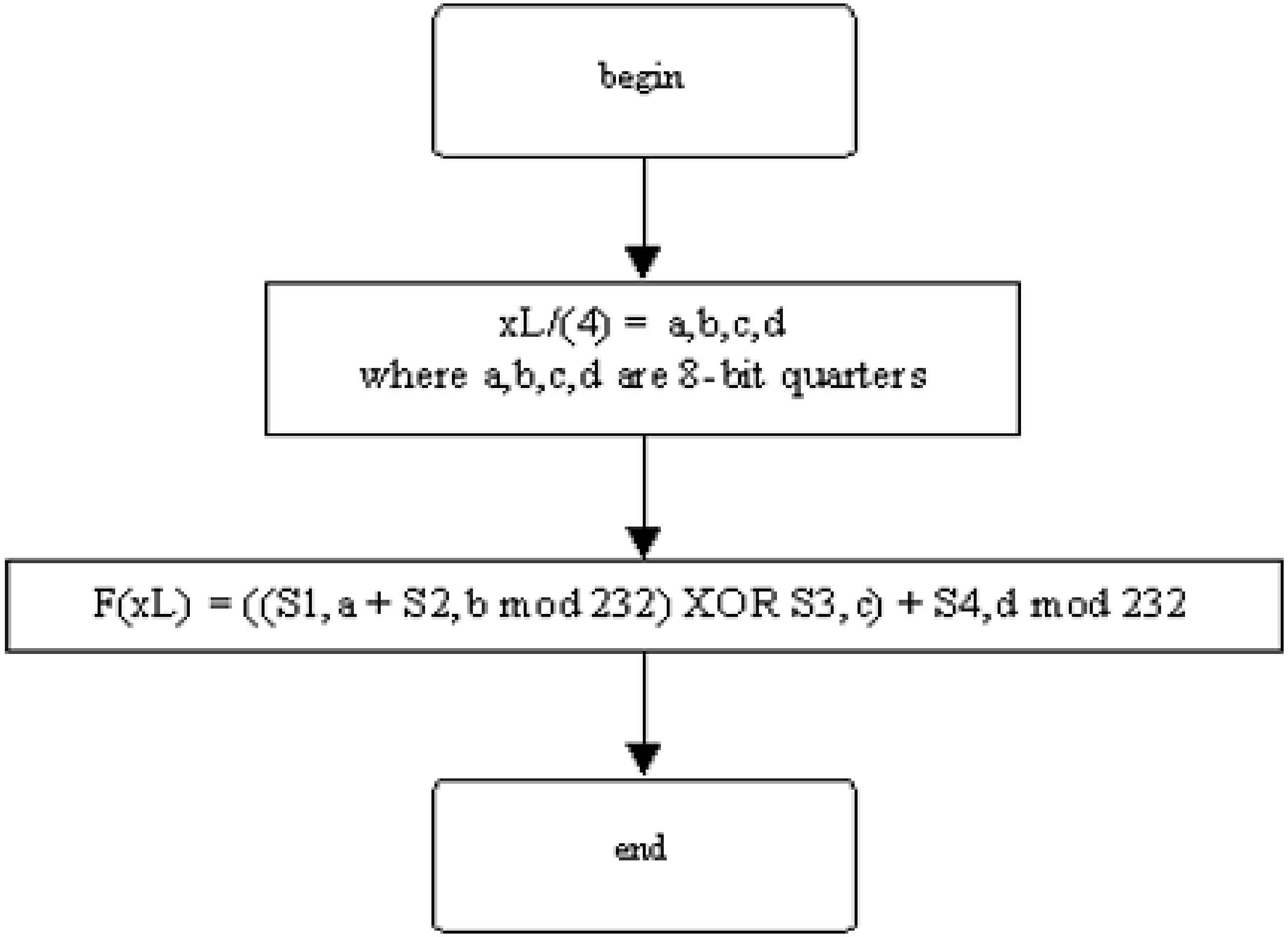


Diagram of Blowfish's F function

- The diagram to the right shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output.
- Since Blowfish is a Feistel network, it can be inverted simply by XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order.


begin
xL/(4) = a,b,c,d
where a,b,c,d are 8-bit quarters
F(xL) = ((S1,a + S2,b mod 232) XOR S3,c) + S4,d mod 232
end

The Function F

- Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern.
- The secret key is then XORed with the P-entries in order (cycling the key if necessary). A 64-bit all-zero block is then encrypted with the algorithm as it stands.
- The resultant ciphertext replaces P1 and P2. The ciphertext is then encrypted again with the new subkeys, and P3 and P4 are replaced by the new ciphertext. This continues, replacing the entire P-array and all the S-box entries.
- In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

Cryptanalysis of Blowfish

- There is no effective cryptanalysis of Blowfish known publicly as of 2005, although the 64-bit block size is now considered too short, because encrypting more than 232 data blocks can begin to leak information about the plaintext due to a birthday attack.
- Despite this, Blowfish seems thus far to be secure. While the short block size does not pose any serious concerns for routine consumer applications like e-mail, Blowfish may not be suitable in situations where large plaintexts must be encrypted, as in data archival.

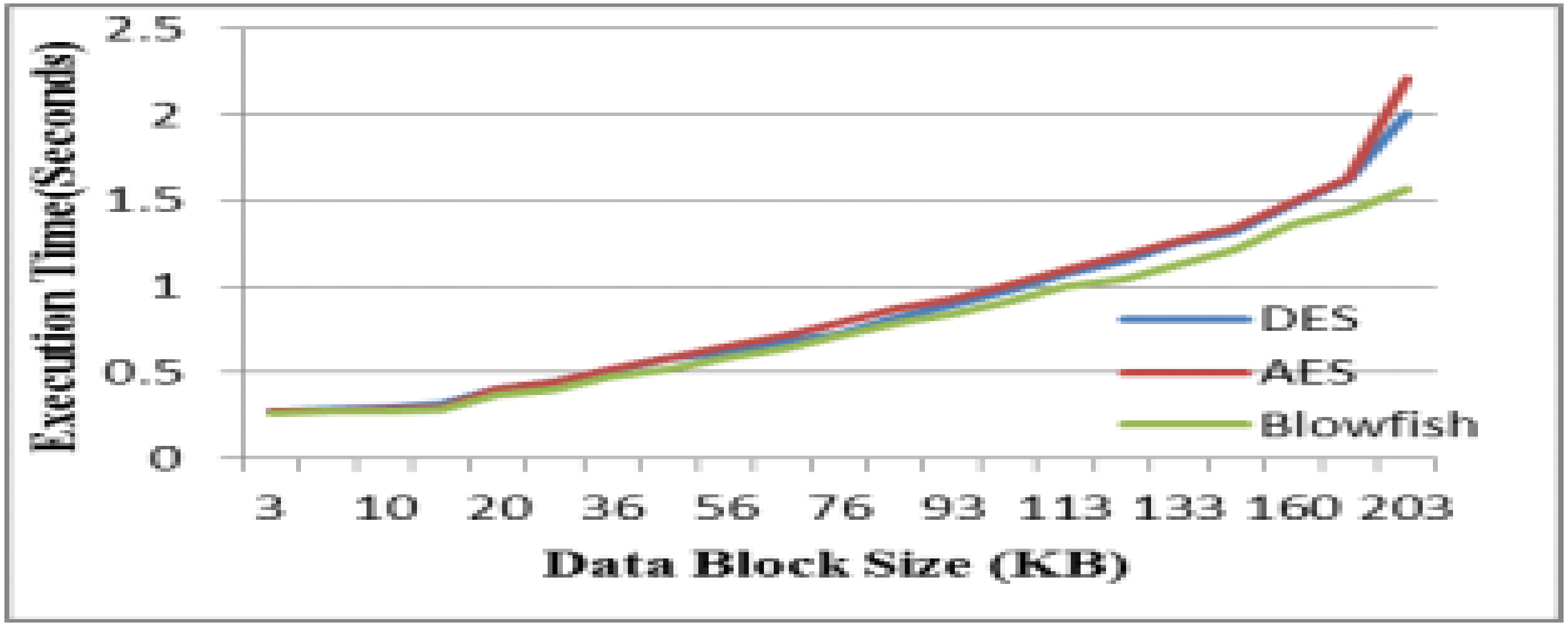
Blowfish in practice

- Blowfish is one of the fastest block ciphers in widespread use, except when changing keys.
- Each new key requires pre-processing equivalent to encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers.
- This prevents its use in certain applications, but is not a problem in others. In one application, it is actually a benefit: the password-hashing method used in OpenBSD uses an algorithm derived from Blowfish that makes use of the slow key schedule; the idea is that the extra computational effort required gives protection against dictionary attacks.

Blowfish in practice (Cont)

- In some implementations, Blowfish has a relatively large memory footprint of just over 4 kilobytes of RAM. This is not a problem even for older smaller desktop and laptop computers, but it does prevent use in the smallest embedded systems such as early smartcards.
- Blowfish is not subject to any patents and is therefore freely available for anyone to use. This has contributed to its popularity in cryptographic software.

Algorithm	Key Size(Bits)	Block Size(Bits)
DES	64	64
AES	128	128
Blowfish	128	64



Performance Results with ECB mode