교과목 : 정보보호

3. Symmetric Key Crypto

2023학년도 2학기
Suk-Hwan Lee

1. Symmetric Key Crypto

- 대칭 키 암호 (Symmetric key algorithm) 암호화와 복호화에 같은 암호 키 사용
- 암호화 측과 복호화 측이 같은 암호 키를 공유 (대칭 키의 주요 단점)
- 공개 키 (public key) 암호에서 공개 키와 비밀 키를 별도로 가지는 것과 구별됨,
- 대칭키 암호는 공개키 암호와 비교하여 계산 속도가 빨라서, 많은 암호화 통신에서는 비밀 키 암호를 사용하여 대칭 키 암호의 공통 키를 공유하고, 그 키를 기반으로 실제 통신을 암호화 하는 구조를 사용함
- 대칭키 암호는 암호화하는 단위에 따라 Stream cipher와 Block cipher로 나눔

1. Symmetric Key Crypto

- Stream cipher – like a one-time pad
  - ✓ Key is relatively short
  - ✓ Key is stretched into a long **keystream**
  - ✓ Keystream is then used like a one-time pad except provable security
  - ✓ Employ confusion only

- Examples of Stream cipher
  - A5/1: employed GSM cell phones
    - Representative stream cipher based in H/W
  - RC4: used SSL(Secure Sockets Layer) protocol
    - Almost unique stream cipher since efficiently implemented in S/W
  - eStream, Salsa20(SW+HW), Sosemaknuk etc

1. Symmetric Key Crypto

- Block cipher — based on codebook concept
  - Block cipher key determines a "electronic" codebook
  - Each key yields a different codebook
  - Employ both "confusion" and "diffusion"

- Examples of Block cipher
  - Data Encryption Stantard (DES) (1975) / Triple DES (1995): relatively simple,
  - Advanced Encryption STD (AES) (2001)
  - International Data Encryption Algorithm (IDEA) (1991)
  - Blowfish (1993), Twofish (1998), Serpent (1998), RC6 (1998)
  - Tiny Encryption Algorithm (TEA) (1994)

1. Symmetric Key Crypto

- Mode of Operation of block cipher
  - Examples of block cipher mode Op
    - Electronic codebook (EOB)
    - Cipher-block chaining (CBC)
    - Cipher feedback (CFB)
    - Output feedback (OFB)
    - Counter (CTR)

- Data integrity of block cipher
  - Message Authentication code (MAC)

Stream Ciphers

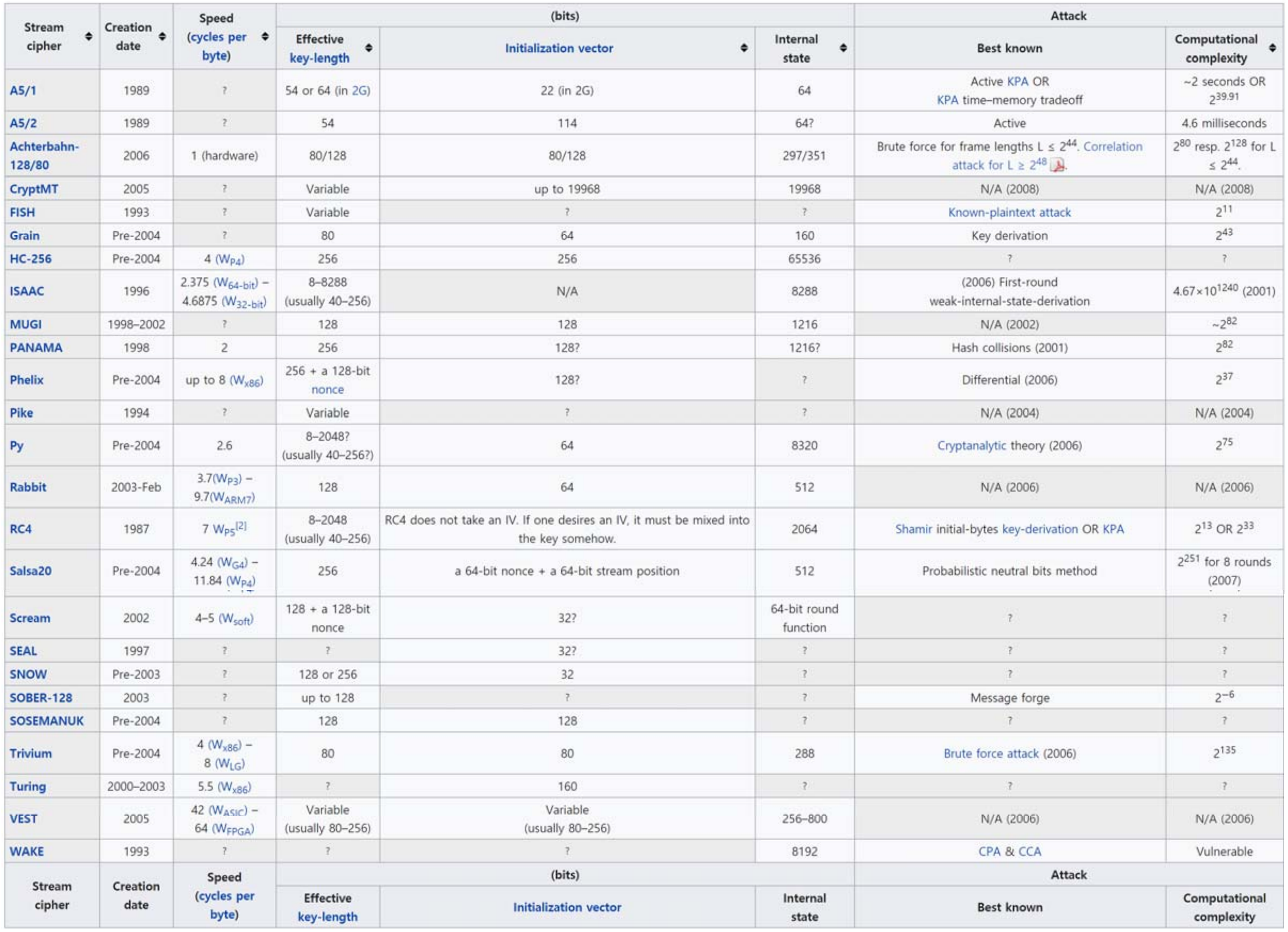
2. Stream Ciphers

- A stream cipher is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (keystream).
- In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the ciphertext stream.
- Since encryption of each digit is dependent on the current state of the cipher, it is also known as state cipher.
- In practice, a digit is typically a bit and the combining operation is an exclusive-or (XOR).

2. Stream Ciphers

Comparison of stream ciphers

[Wikipedia]



Stream cipher	Creation date	Speed (cycles per byte)	(bits)			Attack	
			Effective key-length	Initialization vector	Internal state	Best known	Computational complexity
A5/1	1989	?	54 or 64 (in 2G)	22 (in 2G)	64	Active KPA OR KPA time–memory tradeoff	~2 seconds OR $2^{39.91}$
A5/2	1989	?	54	114	64?	Active	4.6 milliseconds
Achterbahn-128/80	2006	1 (hardware)	80/128	80/128	297/351	Brute force for frame lengths $L \le 2^{44}$. Correlation attack for $L \ge 2^{48}$.	2^{80} resp. 2^{128} for $L \le 2^{44}$.
CryptMT	2005	?	Variable	up to 19968	19968	N/A (2008)	N/A (2008)
FISH	1993	?	Variable	?	?	Known-plaintext attack	2^{11}
Grain	Pre-2004	?	80	64	160	Key derivation	2^{43}
HC-256	Pre-2004	4 (W_{P4})	256	256	65536	?	?
ISAAC	1996	2.375 ($W_{64\text{-bit}}$) – 4.6875 ($W_{32\text{-bit}}$)	8–8288 (usually 40–256)	N/A	8288	(2006) First-round weak-internal-state-derivation	4.67×10^{1240} (2001)
MUGI	1998–2002	?	128	128	1216	N/A (2002)	$\sim2^{82}$
PANAMA	1998	2	256	128?	1216?	Hash collisions (2001)	2^{82}
Phelix	Pre-2004	up to 8 (W_{x86})	256 + a 128-bit nonce	128?	?	Differential (2006)	2^{37}
Pike	1994	?	Variable	?	?	N/A (2004)	N/A (2004)
Py	Pre-2004	2.6	8–2048? (usually 40–256?)	64	8320	Cryptanalytic theory (2006)	2^{75}
Rabbit	2003-Feb	3.7(W_{P3}) – 9.7(W_{ARM7})	128	64	512	N/A (2006)	N/A (2006)
RC4	1987	7 W_{P5}[2]	8–2048 (usually 40–256)	RC4 does not take an IV. If one desires an IV, it must be mixed into the key somehow.	2064	Shamir initial-bytes key-derivation OR KPA	2^{13} OR 2^{33}
Salsa20	Pre-2004	4.24 (W_{G4}) – 11.84 (W_{P4})	256	a 64-bit nonce + a 64-bit stream position	512	Probabilistic neutral bits method	2^{251} for 8 rounds (2007)
Scream	2002	4–5 (W_{soft})	128 + a 128-bit nonce	32?	64-bit round function	?	?
SEAL	1997	?	?	32?	?	?	?
SNOW	Pre-2003	?	128 or 256	32	?	?	?
SOBER-128	2003	?	up to 128	?	?	Message forge	2^{-6}
SOSEMANUK	Pre-2004	?	128	128	?	?	?
Trivium	Pre-2004	4 (W_{x86}) – 8 (W_{LG})	80	80	288	Brute force attack (2006)	2^{135}
Turing	2000–2003	5.5 (W_{x86})	?	160	?	?	?
VEST	2005	42 (W_{ASIC}) – 64 (W_{FPGA})	Variable (usually 80–256)	Variable (usually 80–256)	256–800	N/A (2006)	N/A (2006)
WAKE	1993	?	?	?	8192	CPA & CCA	Vulnerable
Stream cipher	Creation date	Speed (cycles per byte)	(bits)			Attack	
			Effective key-length	Initialization vector	Internal state	Best known	Computational complexity

2. Stream Ciphers

- Not as popular today as block ciphers
- Key K of n bits stretches it into a long keystream
- Function of stream cipher
  - StreamCipher(K) = S

    where K:key, S:keystream, s_0, s_1, s_2, ⋯
  - S is used like a one-time pad with a plaintext P=p_0,p_1,p_2,..., a ciphertext C=c_0,c_1,c_2....
    - $c_0 = p_0 \oplus s_0$, $c_1 = p_1 \oplus s_1$, $c_2 = p_2 \oplus s_2$, ⋯
    - $p_0 = c_0 \oplus s_0$, $p_1 = c_1 \oplus s_1$, $p_2 = c_2 \oplus s_2$, ⋯
- Sender and receiver have same stream cipher algorithm and both know the key K

2. Stream Ciphers

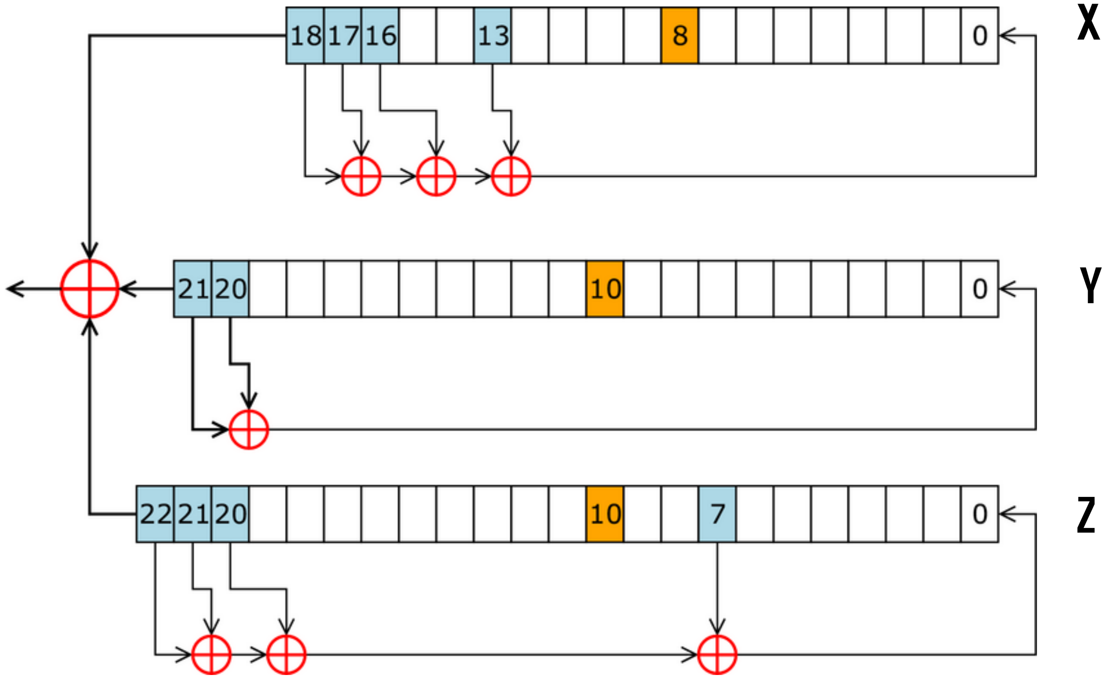
- We'll discuss two examples
- A5/1 (1987) (A5/2, 1989)
  - Based on linear feedback shift registers
  - Used in GSM mobile phone system
    - A5/1 is used in Europe and the United States;
    - A5/2, is used in countries that are not considered trustworthy enough to have strong crypto.
- RC4 (Designed in 1987, First leaked in 1994)
  - Based on a changing lookup table
  - Used many places – SSL

2. Stream Ciphers – A5/1

- A5/1 is Representative stream cipher based in H/W
- Consists of 3 LFSR(Linear feedback shift registers)
  - X: 19 bits (x_0, x_1, x_2, ···, x_{18})
  - Y: 22 bits (y_0, y_1, y_2, ·········, y_{21})
  - Z: 23 bits (z_0, z_1, z_2, ············,z_{22})
  - X+Y+Z = 64 bits (Key)

2. Stream Ciphers – A5/1

- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: maj(0,1,0) = 0 and maj(1,1,0) = 1
- If $x_8 = m$ then X steps
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for i = 18, 17, ⋯, 1 and $x_0 = t$
- If $y_{10} = m$ then Y steps
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for i = 21, 20, ⋯, 1 and $y_0 = t$
- If $z_{10} = m$ then Z steps
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for i = 22, 21, ⋯, 1 and $z_0 = t$
- **Keystream bit is $x_{18} \oplus y_{21} \oplus z_{22}$**



From Wikipedia
(https://en.wikipedia.org/wiki/Stream_cipher)

(maj = majority)

암호해야하는 비트수만큼 반복하여 keystream 생성한 다음,
keystream과 plaintext를 XOR 연산하여 Ciphertext 생성

2. Stream Ciphers – RC4

- RC4 was designed by Ron Rivest of RSA Security in 1987. ("Rivest Cipher 4", "Ron's Code")
- RC4 Optimized for software implementation, whereas A5/1 for hardware
- RC4 produces a keystream BYTE at each step, whereas A5/1 only produce a single keystream bit
- RC4 generates a pseudo-random stream of bits (a keystream). As with any stream cipher, these can be used for encryption by combining it with the plaintext using bit-wise exclusive-or. Decryption is performed the same way (since exclusive-or is a symmetric operation).

- RC4 is remarkably simple
  - Because it is essentially just lookup table containing permutation of the 256(2^8)-byte values
  - Each time a byte of keystream is produced, the lookup table is modified in such a way that the table always contains a permutation of {0,1,2,···256}

RC4 Initialization : Key-scheduling algorithm (KSA)

- The first phase – initialize the lookup table (S) using key (KSA에 의하여 256byte의 S배열을 permutation)
  - Key: key[i] for i=0,1,⋯,N-1 where key[i] is a byte
  - N is keylength (1≤N≤256) (Typically between 5 and 16)
  - Key is only use to initialize the permutation S
  - Lookup table: S[i] is a byte

- S[] is permutation of 0,1,⋯,255
- key[] contains N bytes of key

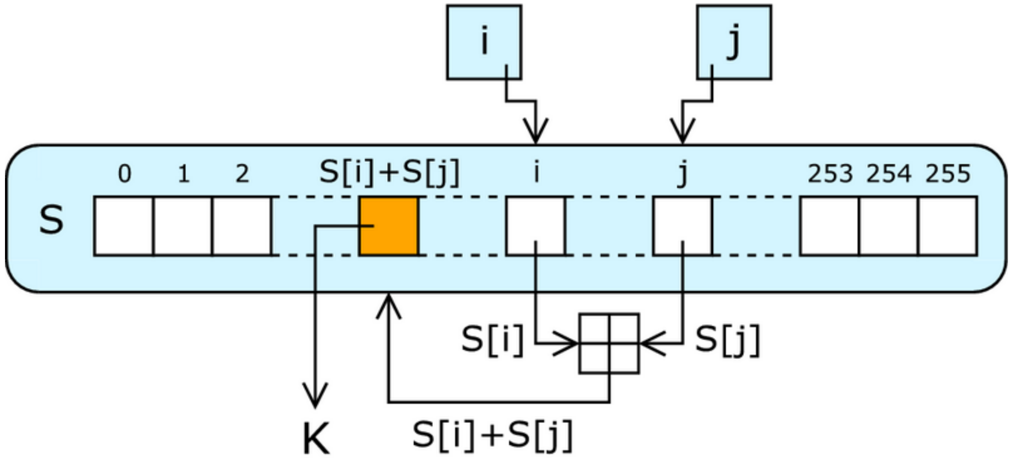
```
for i = 0 to 255
        S[i] = i
        T[i] = key[i (mod N)]

j = 0
for i = 0 to 255
        j = (j + S[i] + T[i]) mod 256
        swap(S[i], S[j])

i = j = 0
```

RC4 Keystream : Pseudo-random generation algorithm (PRGA)

- The next phase – each keystream byte is generated according the following algorithm

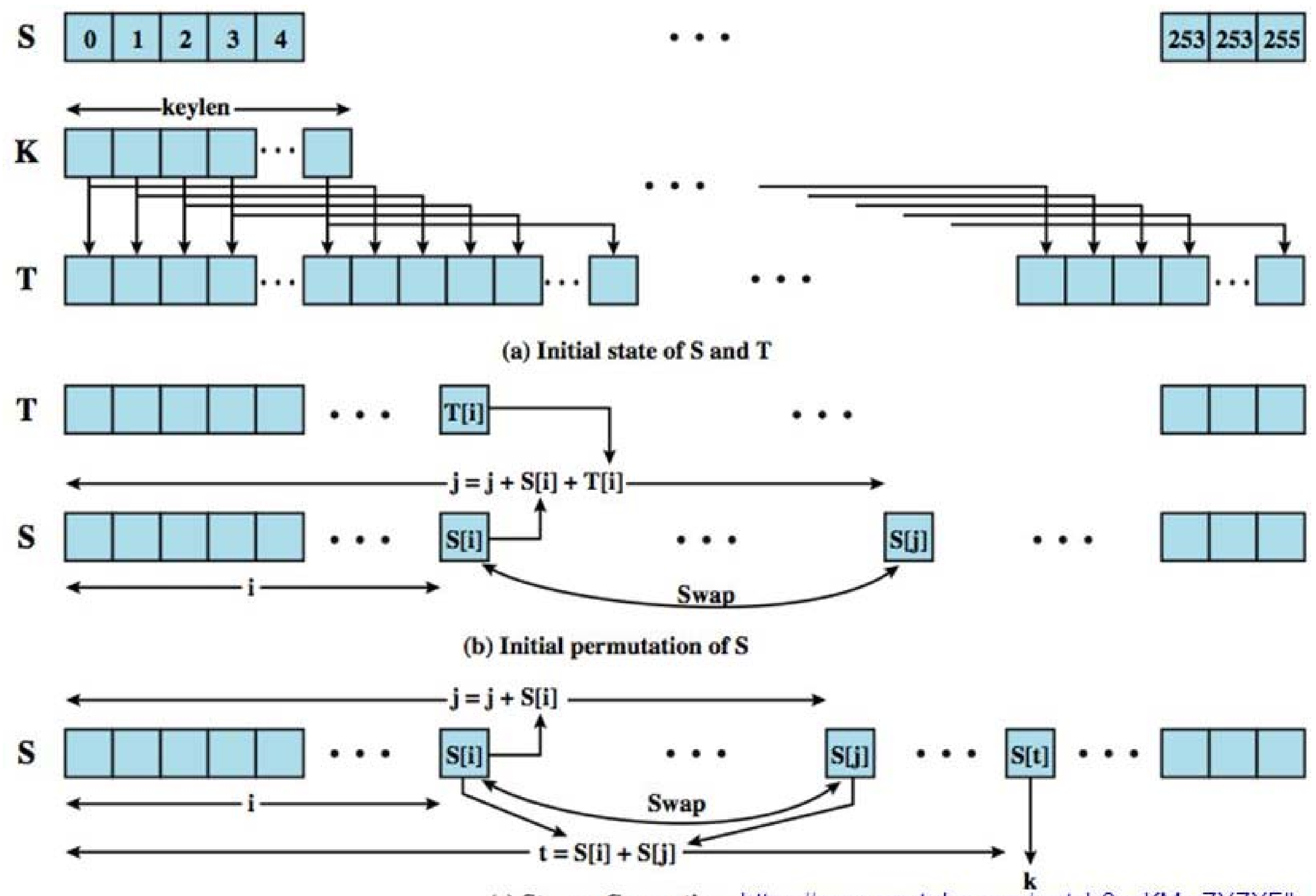
```
i=0, j=0
while GeneratingOutput:
    • i = (i + 1) mod 256
    • j = (j + S[i]) mod 256
    • swap(S[i], S[j])
    • k = (S[i] + S[j]) mod 256
    • keystreamByte = S[k]
    • XOR with next byte of (plaintext or ciphertext) message
```

- **Note:** first 256 bytes must be discarded
  - Otherwise attacker may be able to recover key



The lookup stage of RC4. The output byte is selected by looking up the values of S[i] and S[j], adding them together modulo 256, and then using the sum as an index into S; S(S[i] + S[j]) is used as a byte of the key stream, K.

2. Stream Ciphers – RC4



(c) Stream Generation https://www.youtube.com/watch?v=KM-xZYZXElk

```
for i = 0 to 255
        S[i] = i
        T[i] = key[i (mod N)]
```

```
j = 0
for i = 0 to 255
        j = (j + S[i] + T[i]) mod 256
        swap(S[i], S[j])

i = j = 0
```

```
i=0, j=0
while GeneratingOutput:
        i = (i + 1) mod 256
        j = (j + S[i]) mod 256
        swap(S[i], S[j])
        t = (S[i] + S[j]) mod 256
        keystreamByte = S[t]
```

2. Stream Ciphers – RC4

RC4 python (https://github.com/g2jun/RC4-Python)

```
def crypt(PlainBytes, KeyBytes):
    keystreamList = []
    cipherList = []

    keyLen = len(KeyBytes)
    plainLen = len(PlainBytes)
    S = range(256)

    j = 0
    for i in range(256):
      j = (j + S[i] + KeyBytes[i % keyLen]) % 256
      S[i], S[j] = S[j], S[i]

    i = 0
    j = 0
    for m in range(plainLen):
      i = (i + 1) % 256
      j = (j + S[i]) % 256
      S[i], S[j] = S[j], S[i]
      k = S[(S[i] + S[j]) % 256]
      keystreamList.append(k)
      cipherList.append(k ^ PlainBytes[m])

    return keystreamList, cipherList
```

```
for i = 0 to 255
      S[i] = i
      T[i] = key[i (mod N)]
```

```
j = 0
for i = 0 to 255
      j = (j + S[i] + T[i]) mod 256
      swap(S[i], S[j])

i = j = 0
```

```
i=0, j=0
while GeneratingOutput:
      i = (i + 1) mod 256
      j = (j + S[i]) mod 256
      swap(S[i], S[j])
      t = (S[i] + S[j]) mod 256
      keystreamByte = S[t]
```

^ : bitwise XORc

2. Stream Ciphers

- Stream ciphers were big in the past
  - Efficient in hardware
  - Speed needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is fast enough
- Future of stream ciphers?
  - Shamir: “the death of stream ciphers”
  - May be exaggerated…

Stream Ciphers and Block Ciphers

- Stream ciphers
  - ✓ Encrypts a data stream one bit or one byte at a time.
  - ✓ Examples : the autokeyed Vigenre cipher and the Vernam cipher. In the ideal case, a one-time pad version of the Vernam cipher would be used, in which the keystream (k_i) is as long as the plaintext bit stream (p_i).
  - ✓ If the cryptographic keystream is random, then this cipher is unbreakable by any means other than acquiring the keystream. However, the keystream must be provided to both users in advance via some independent and secure channel. This introduces insurmountable logistical problems if the intended data traffic is very large.

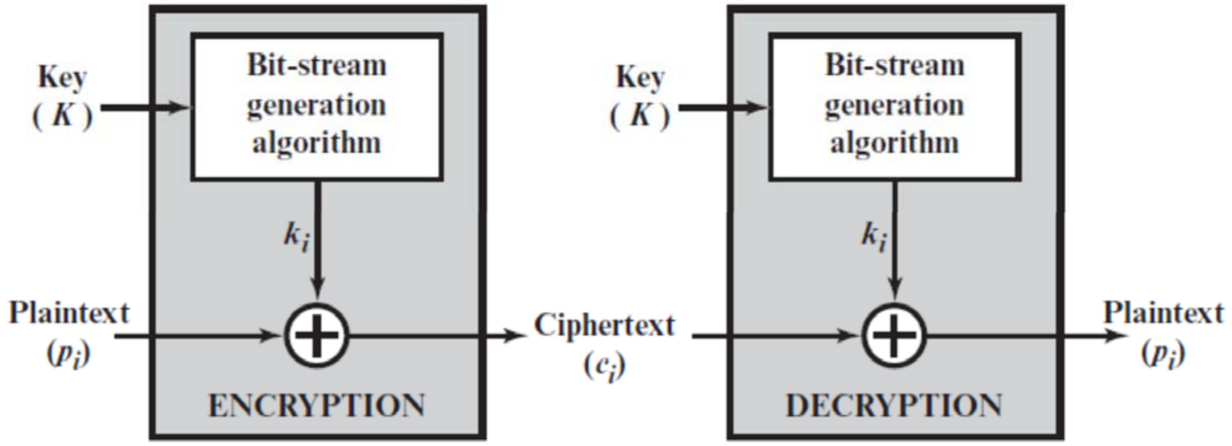


Figure 4.1(a). Stream cipher using algorithmic bit-stream generator

- ✓ The bit-stream generator must be implemented, so that the cryptographic bit stream can be produced by both users.
- ✓ The bit-stream generator is a key-controlled algorithm and must produce a bit stream that is cryptographically strong.
- ✓ The two users need only share the generating key, and each can produce the keystream.

Stream Ciphers and Block Ciphers

- Block ciphers
  - ✓ A block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.
  - ✓ As with a stream cipher, the two users share a symmetric encryption key (Figure 4.1b). Using some of the modes of operation, a block cipher can be used to achieve the same effect as a stream cipher.
  - ✓ In general, they seem applicable to a broader range of applications than stream ciphers. The majority of network-based symmetric cryptographic applications make use of block ciphers.

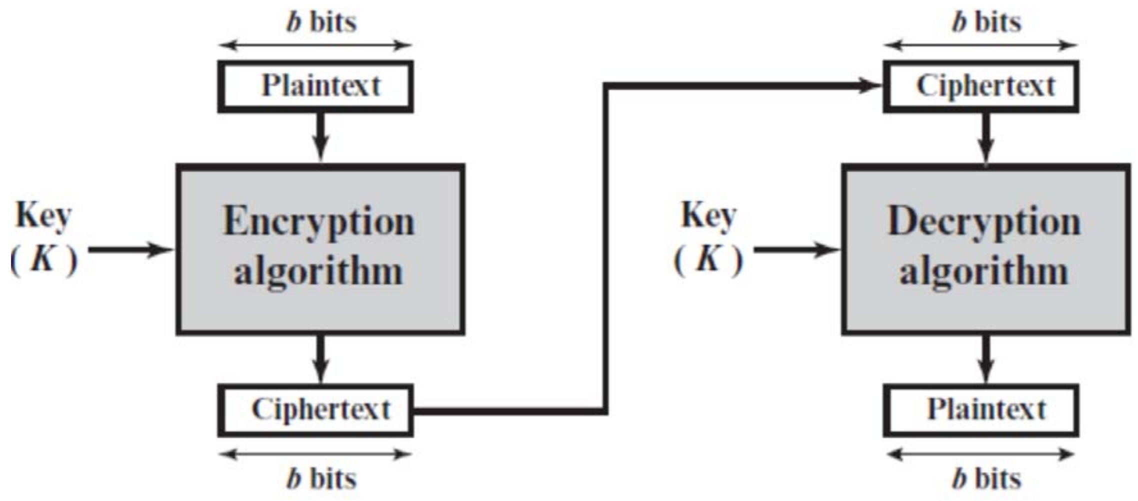
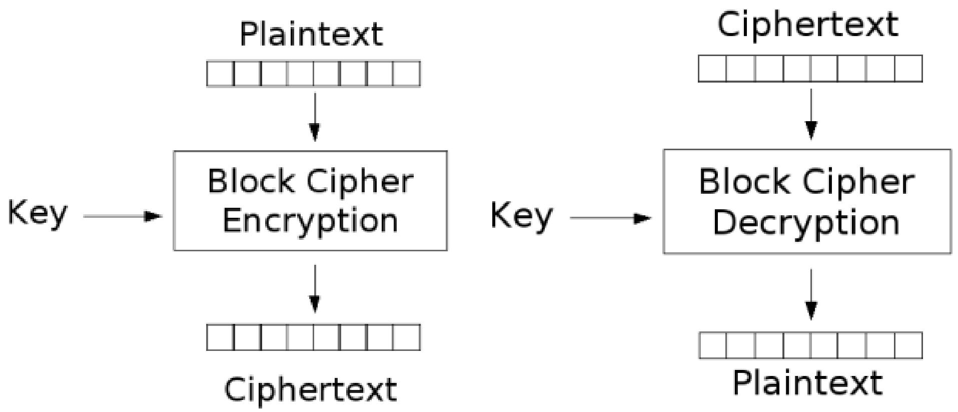


Figure 4.1(b). Block cipher

Block Ciphers

3. Block Ciphers

- Plaintext and ciphertext consists of fixed sized blocks
- Design goal: security and efficiency
  - It is not easy to design a block cipher that is secure and efficient



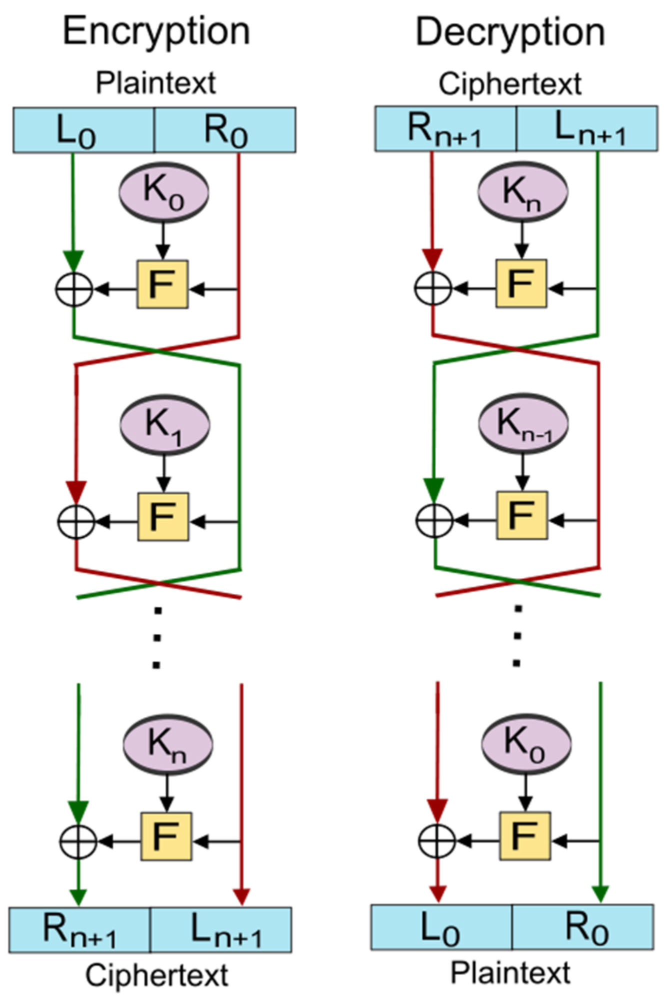
[Wikipedia]

- 블록 암호(block cipher)란 암호학 용어로, 기밀성 있는 정보를 정해진 블록 단위로 암호화하는 대칭키 암호 시스템이다. 만약 암호화하려는 정보가 블록 길이보다 길 경우에는 특정한 운용 모드가 사용된다. (예: **ECB, CBC, OFB, CFB, CTR**)
- 블록 암호에 대한 안전성 증명 방법으로는 선택평문공격인 차분공격(Differential Cryptanalysis)과 알려진평문공격인 선형공격(Linear Cryptanalysis)등이 있다.
- 블록 암호 구조에는 Feistel구조와 SPN구조가 있다.
- *Feistel* 구조 : **암복호화 과정에서 역함수가 필요 없다**는 장점이 있지만 구현 시 **Swap 단계 때문에 연산량이 많이 소요**되며 암호에 사용되는 **round 함수를 안전하게 설계**해야 한다는 단점이 있다. 대표적인 암호로는 DES가 있으며 Single DES는 안전성 문제로 현재 사용하고 있지 않다.
- 한국에서는 개발된 암호 중에서는 Feistel 기반 설계된 SEED가 있다.
- *SPN* 구조 : 암복호화 과정에서 **역함수가 필요**하도록 설계되어야 한다는 단점이 있지만 **중간에 비트의 이동없이 한번에 암복호화**가 가능하기 때문에 **Feistel 구조에 비해 효율적**으로 설계할 수 있다. 대표적인 암호로는 AES가 있으며 AES는 현재 널리 상용되고 있다.

3. Block Ciphers

(Iterated) Block Cipher

- Ciphertext obtained from plaintext by iterating a round function
- Input to round function consists of key and the output of previous round
- Usually implemented in software
- Typical Type is Feistel Cipher



Feistel Cipher [Wikipedia]

3. Block Ciphers

Feistel Cipher

- Feistel cipher refers to a type of block cipher design, not a specific cipher

✓ Let F be the round function and let K_0, K_1, …, K_n be the sub-keys for the rounds 0, 1, …, n respectively.

✓ Then the basic operation is as follows:

✓ Split the plaintext block into two equal pieces, (L_0, R_0)

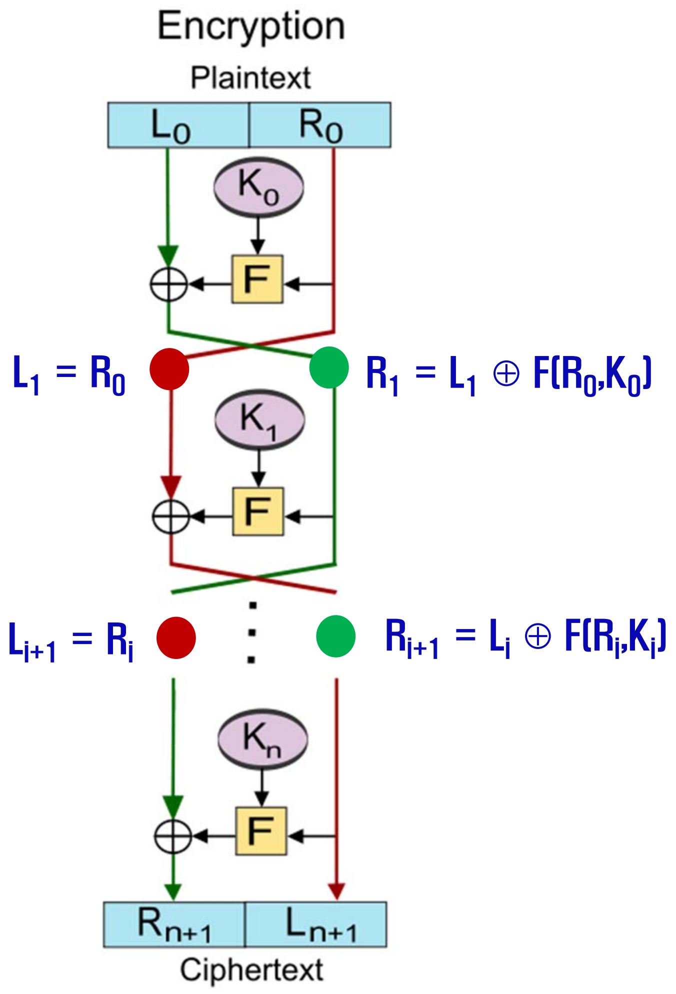
✓ For each round i=0, 1, …, n compute

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

✓ Where $\oplus$ means XOR, Then the ciphertext is

$$(R_{n+1}, L_{n+1})$$



3. Block Ciphers

Feistel Cipher

- Decryption: Ciphertext = (R_{n+1}, L_{n+1})
- ✓ For each round i=n,n−1,⋯,1, compute

$R_i = L_{i+1}$
$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$

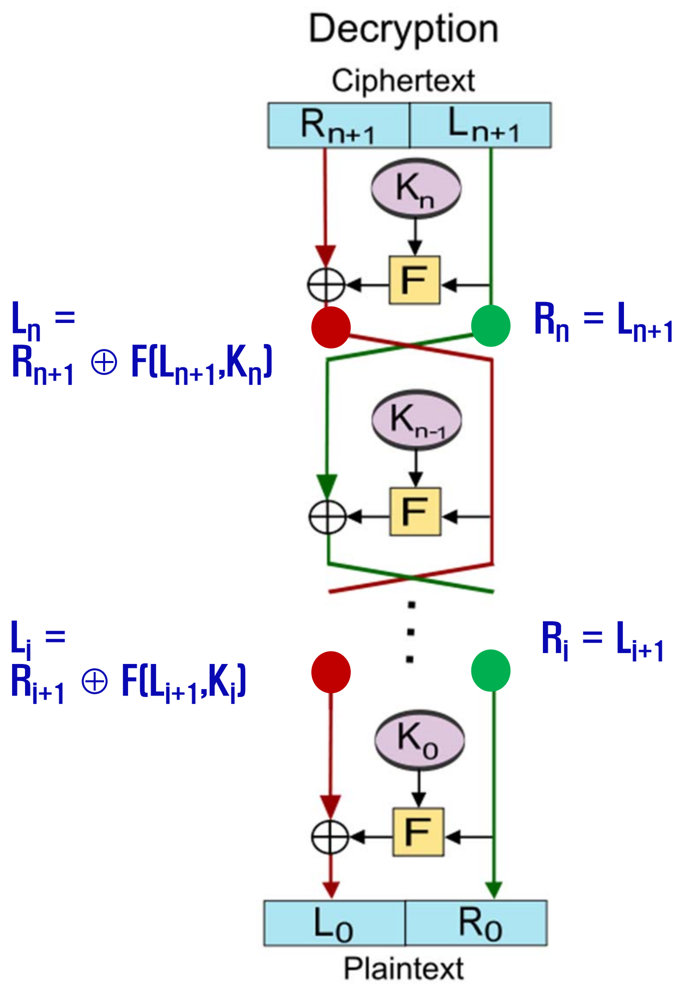
$L_{i+1} = R_i$
$R_{i+1} = L_i \oplus F(R_i, K_i)$

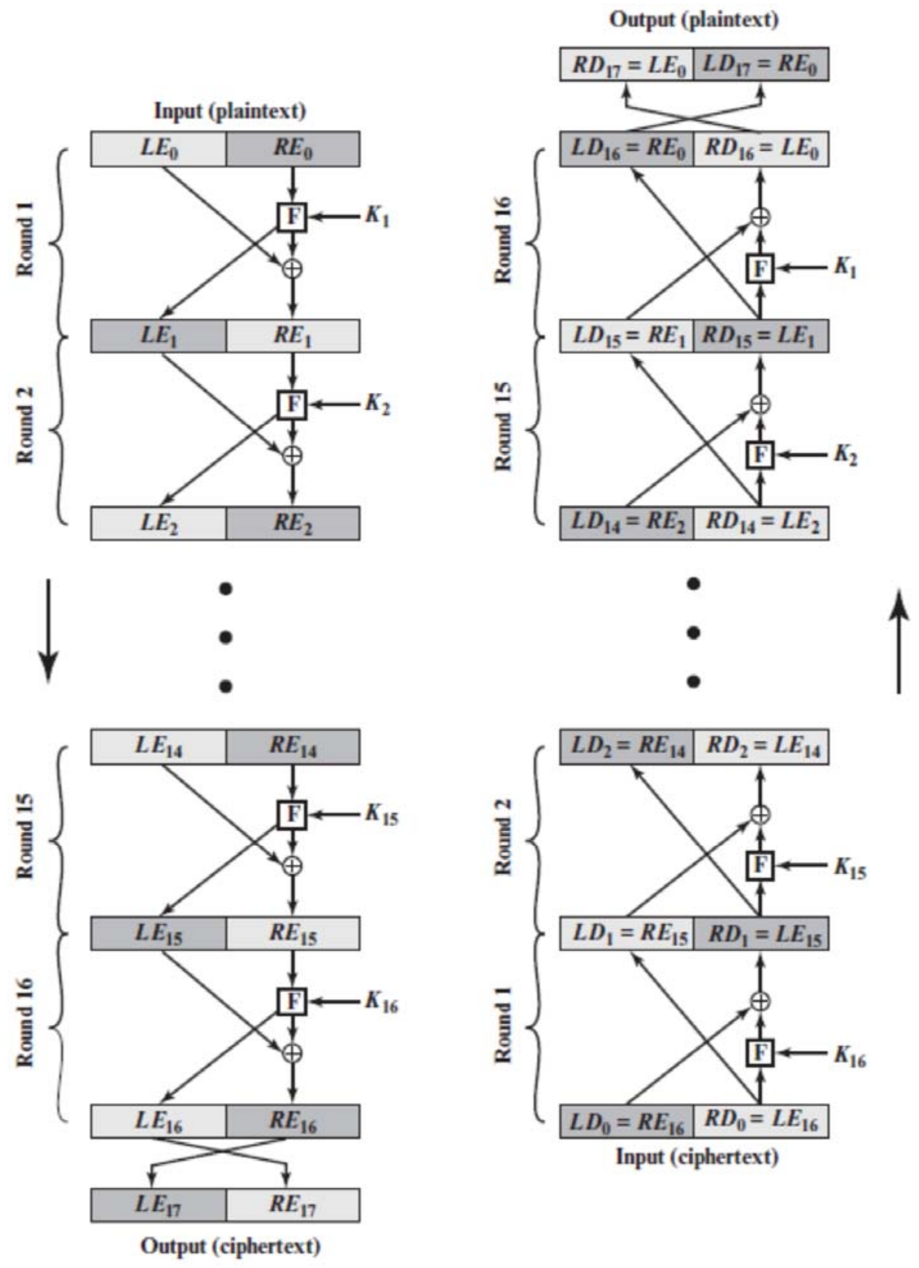
where F is round function and K_i is subkey

- ✓ Plaintext = (L_0, R_0)

- Formula "works" for any function F
- But only secure for certain functions F
  - ✓ Ex: $F(R_{i-1}, K_i) = 0$ for all R_{i-1} and K_i −> not secure

- One advantage of the Feistel model compared to a substitution–permutation network is that the round function F does not have to be invertible.





The exact realization of a Feistel network depends on the choice of the following parameters and design features

- **Block size** : Generally 64bits. However, the AES uses a 128-bit block size.
- **Key size** : 128bits has become a common size.
- **Number of rounds** : A Typical size is 16 rounds
- **Subkey generation algorithm** : Lead to difficulty of cryptanalysis
- **Round Function** F : Resistance to cryptanalysis

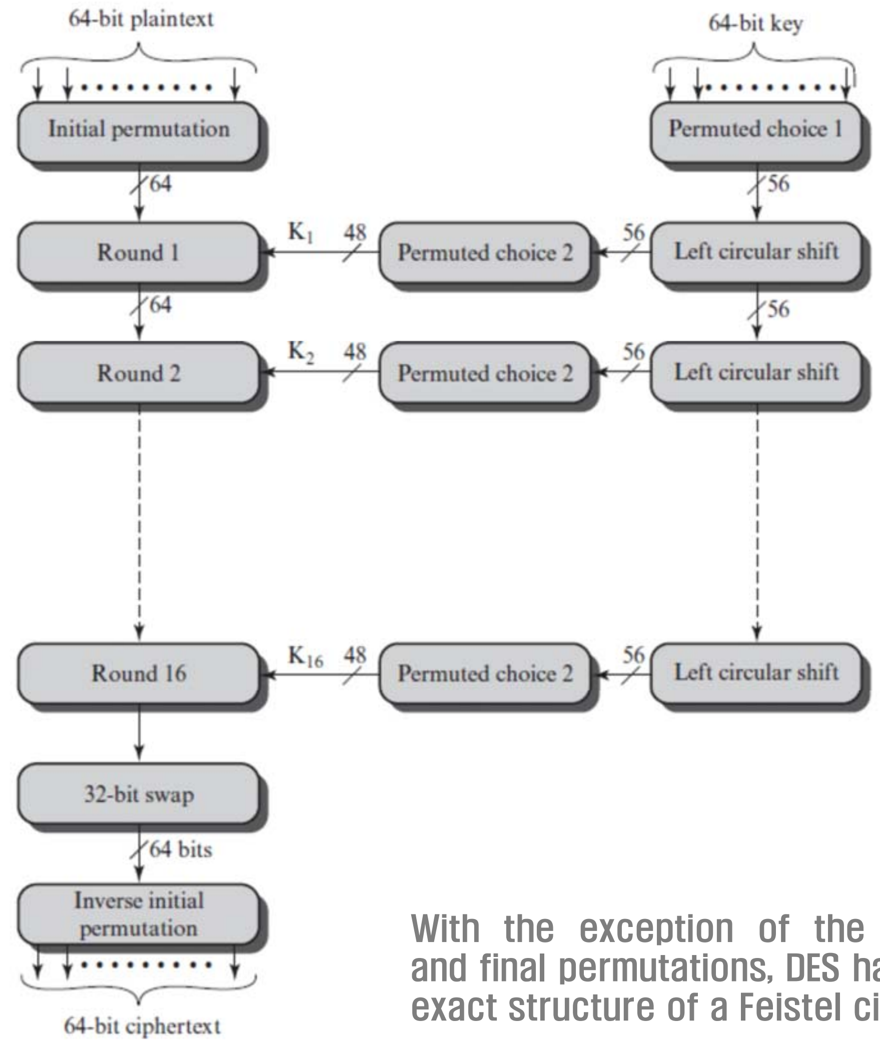
Figure. Feistel Encryption and Decryption (16 rounds)

Data Encryption Standard, DES

[Wikipedia]

- DES는 블록 암호의 일종으로, USA NBS (National Bureau of Standards, 현재 NIST)에서 국가 표준으로 정한 암호
- DES는 대칭키 암호이며, 56비트의 키를 사용
- DES는 현재 취약한 것으로 알려져 있다. 56비트의 키 길이는 현재 컴퓨터 환경에 비해 너무 짧다는 것이 하나의 원인이며, DES에 백도어가 포함되어 있어 특수한 방법을 사용하면 정부 기관에서 쉽게 해독할 수 있을 것이라는 주장도 제기됨
- 1998년에 전자 프론티어 재단(EFF)에서는 56시간 안에 암호를 해독하는 무차별 대입 공격 하드웨어를 만들었으며, 1999년에는 22시간 15분 안에 해독하는 하드웨어를 만들었다.
- DES를 세 번 반복해서 사용하는 Triple-DES는 DES에 비해 안전한 것으로 알려져 있으며, 또한 현재는 DES 대신 AES(Advanced Encryption Standard)가 새 표준으로 정해져 사용되고 있다

3. Block Ciphers – DES



With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher

Figure 4.5 General Depiction of DES Encryption Algorithm

The overall scheme for DES encryption

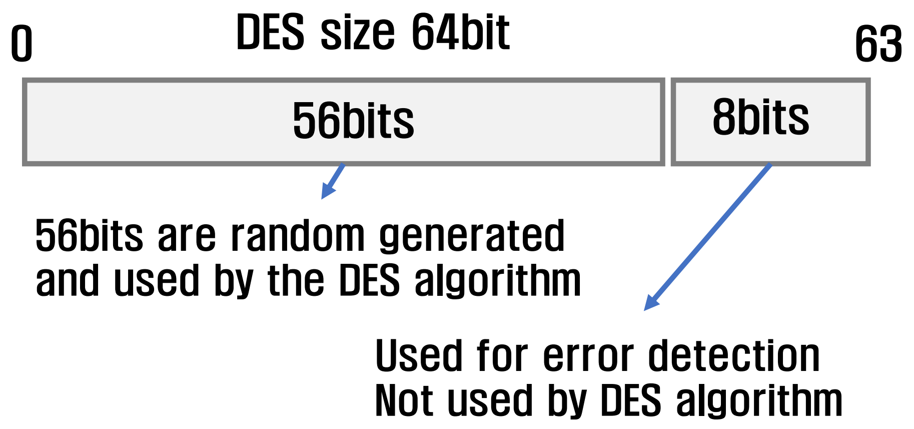
- Two inputs : the plaintext (64bits) and the key (56bits).
- Left-hand side, the processing of the plaintext proceeds in three phases.
  1) The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*.
  2) 16 rounds of the same function, which involves both *permutation* and *substitution* functions. The output of the last round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput.
  3) The preoutput is passed through a permutation [IP-1] that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.
- Right-hand side, 54-bit key is used
  - ✓ Initially, the key is passed through a permutation function.
  - ✓ Then, for each of 16 rounds, a subkey (Ki) is produced by the combination of a left circular shift and a permutation.
  - ✓ The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

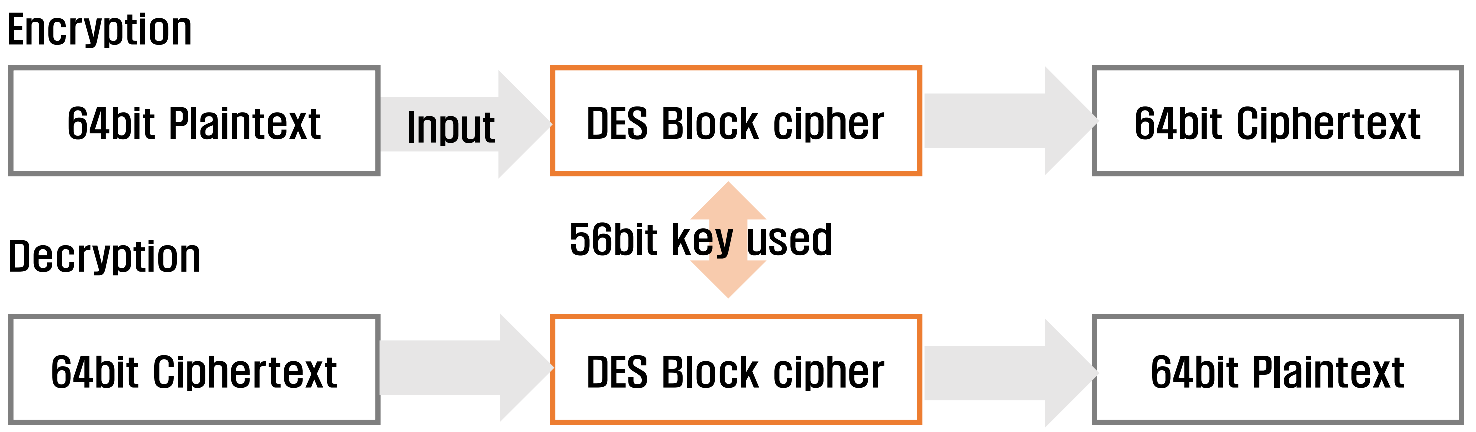
3. Block Ciphers – DES

DES Numerology

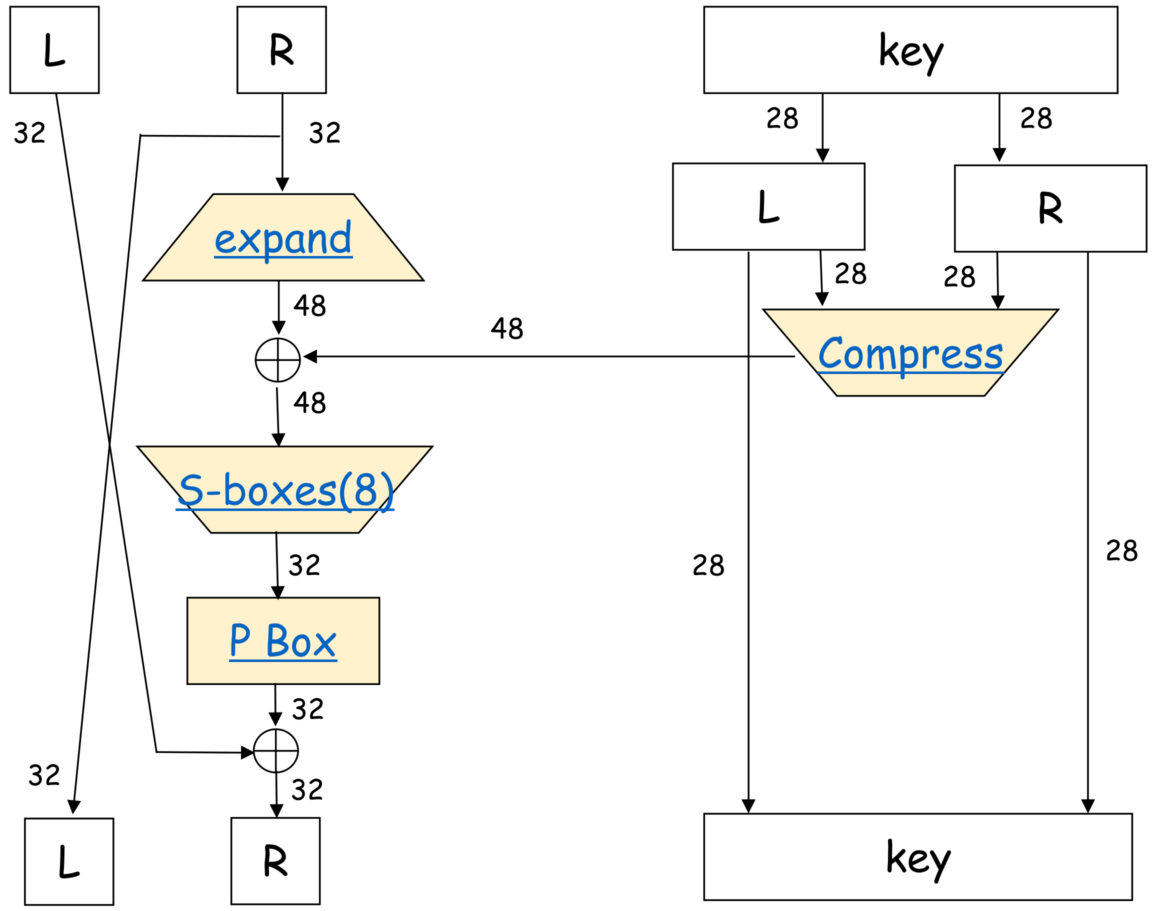
- DES is a Feistel cipher
  - ✓ 64 bit block length 56 bit key length
  - ✓ 16 rounds
  - ✓ 48 bits of key used each round (subkey)
- Each round is simple (for a block cipher)
- Security depends primarily on "S-boxes"
  - ✓ Each S-boxes maps 6 bits to 4 bits, Total 8 S-boxes

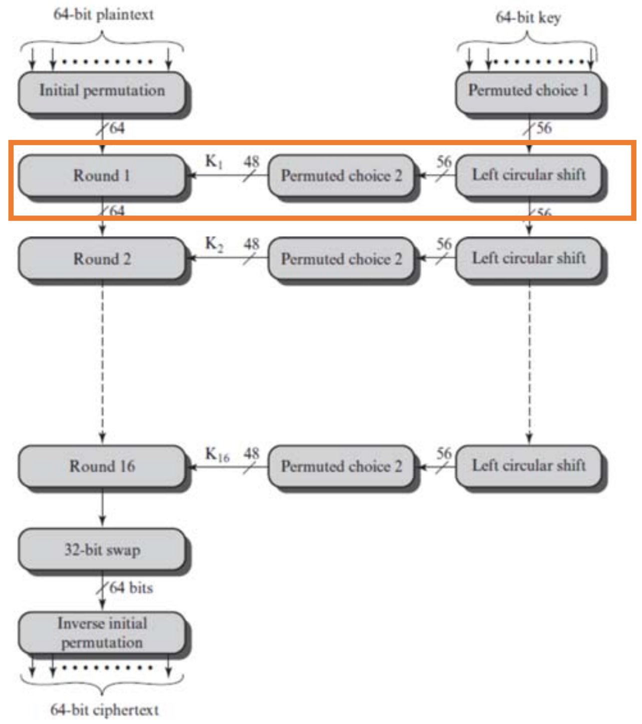
Cases


0
DES size 64bit
63
56bits
8bits
56bits are random generated and used by the DES algorithm
Used for error detection
Not used by DES algorithm



One Round of DES





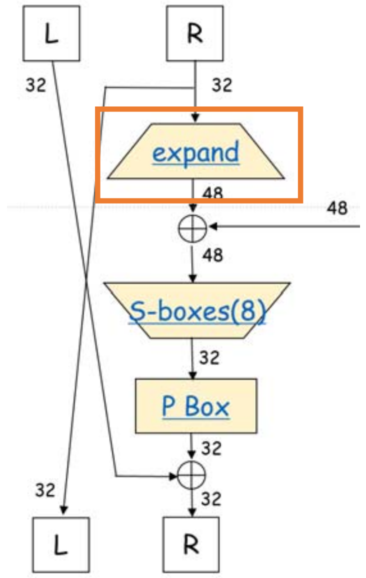
DES Expansion Permutation

- Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- Output 48 bits

31	0	1	2	3	4	3	4	5	6	7	8
7	8	9	10	11	12	11	12	13	14	15	16
15	16	17	18	19	20	19	20	21	22	23	24
23	24	25	26	27	28	27	28	29	30	31	0

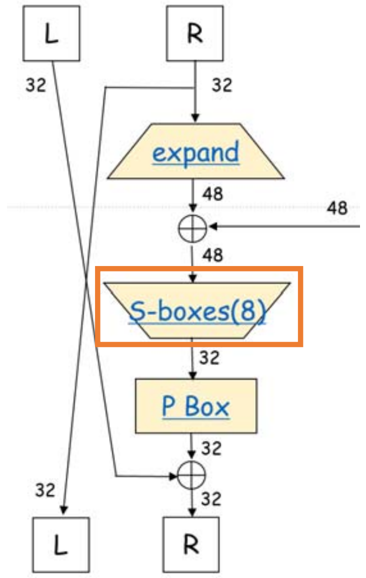


3. Block Ciphers – DES

DES S-box (substitution-box)

- 8 "substitution boxes" or S-boxes
- Each S-box maps 6 bits to 4 bits
- S-box number 1

48bits → 32bits
6bits x 8 → 4bits x 8
8개 S-boxes



input bits (0,5)
↓

input bits (1,2,3,4)

1번 S-box

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0111	1011	0011	1110	1010	0000	0110	1101

3. Block Ciphers – DES

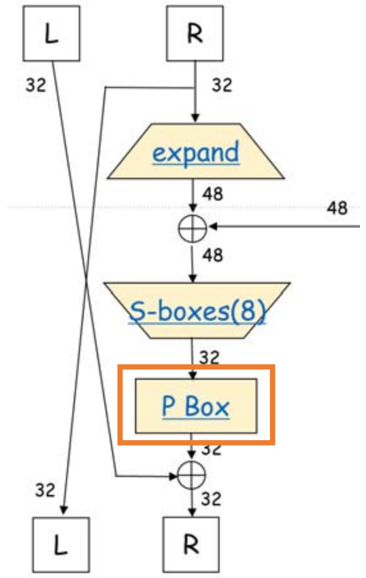
DES P-box (permutation-box)

- Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- Output 32 bits

15	6	19	20	28	11	27	16	0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8	18	12	29	5	21	10	3	24



3. Block Ciphers – DES

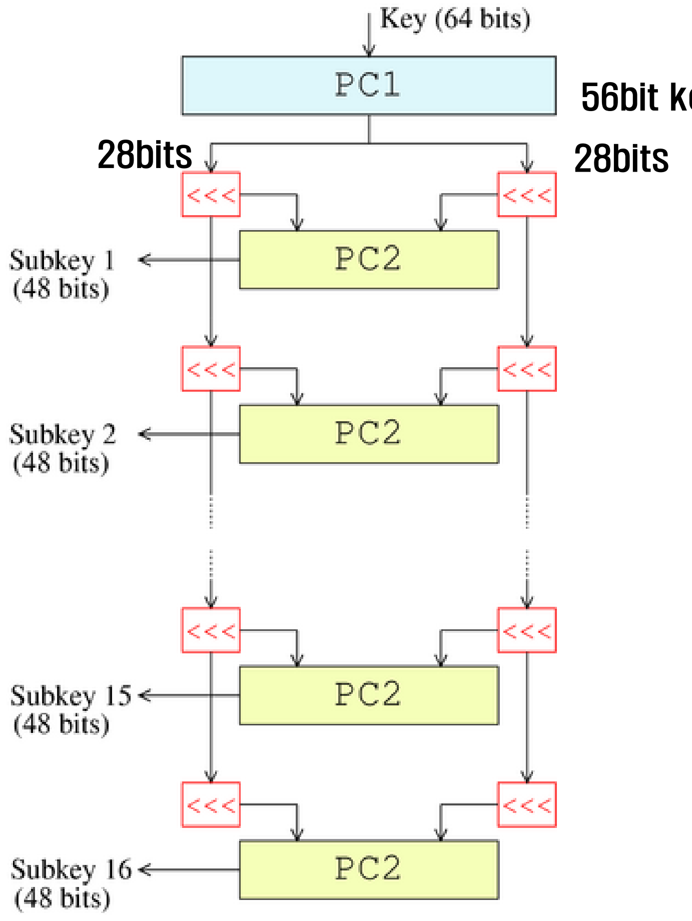
DES Subkey (Key schedule)

- 56 bit DES key, numbered 0,1,2,⋯,55
- Left half key bits (28), LK

49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31

- Right half key bits (28), RK

55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3



Both halves are rotated left by one or two bits (specified for each round), and then 48 round key bits are selected by Permuted Choice 2 (PC–2) – 24 bits from the left half and 24 from the right. The rotations have the effect that a different set of bits is used in each round key; each bit is used in approximately 14 out of the 16 round keys.

The key schedule of DES ("<<<" denotes a left rotation), showing the calculation of each round key ("Subkey").

DES Subkey (Key schedule)

- For rounds i=1,2,...,16
  - ✓ Let LK = (LK circular shift left by r_i)
  - ✓ Let RK = (RK circular shift left by r_i)

  - ✓ Left half of subkey K_i is LK bits

28bits → 24bits

```
13 16 10 23  0  4  2 27 14  5 20  9
22 18 11  3 25  7 15  6 26 19 12  1
```

  - ✓ Right half of subkey K_i is RK bits

28bits → 24bits

```
12 23  2  8 18 26  1 11 22 16  4 19
15 20 10 27  5 24 17 13 21  7  0  3
```

- For rounds 1, 2, 9 and 16 the shift r_i is 1, and in all other rounds r_i is 2
- Bits 8,17,21,24 of LK omitted each round
- Bits 6,9,14,25 of RK omitted each round
- Compression permutation yields 48 bit subkey K_i from 56 bits of LK and RK

- Key schedule generates subkey

DES Last process

- An initial permutation P before round 1
- Halves are swapped after last round
- A final permutation (inverse of P) is applied to (R_{16},L_{16}) to yield ciphertext
- None of these serve any security purpose

Security of DES

- Security of DES depends a lot on S-boxes
  - ✓ Everything else in DES is linear
- Thirty years of intense analysis has revealed no "back door"
- Attacks today use exhaustive key search (전수키 조사)
- Inescapable conclusions
  - ✓ Designers of DES knew what they were doing
  - ✓ Designers of DES were ahead of their time

Block Ciphers – Notation

- P = plaintext block
- C = ciphertext block
- Encrypt P with key K to get ciphertext C

$$C = E(P, K)$$

- Decrypt C with key K to get plaintext P

$$P = D(C, K)$$

- Note that

$$P = D(E(P, K), K) \text{ and } C = E(D(C, K), K)$$

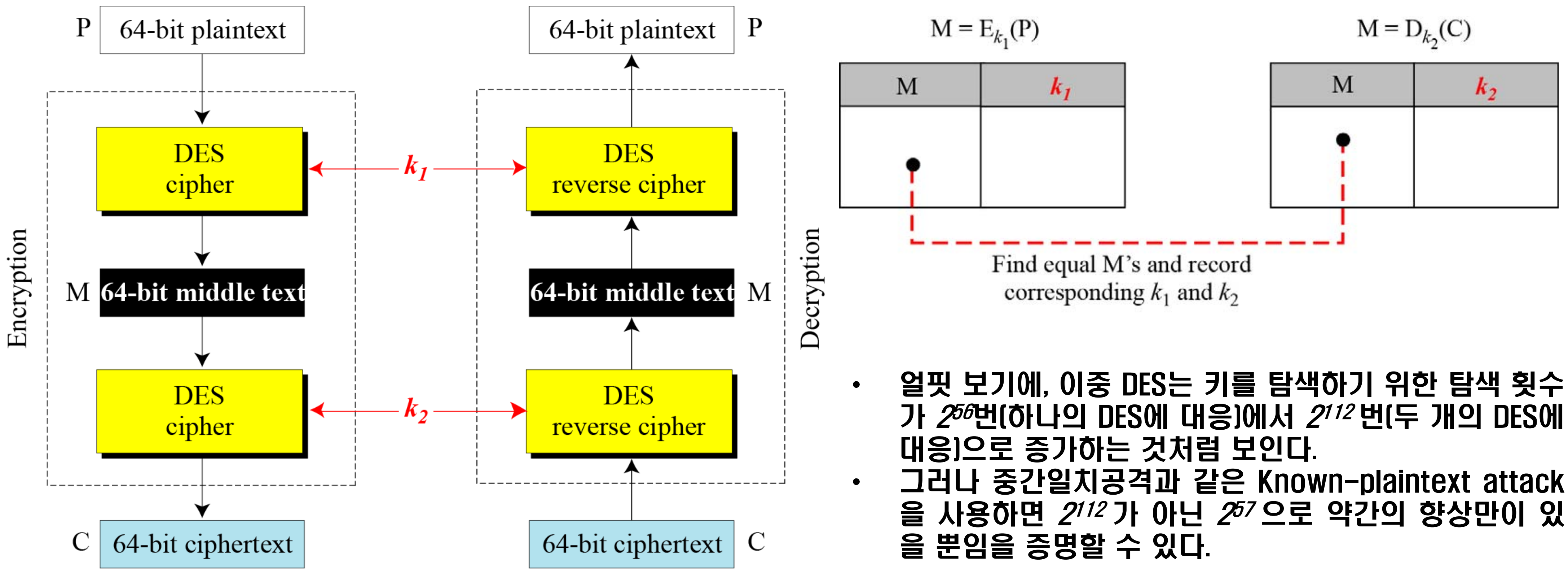
Double DES

- DES's key length is insufficient today.
- A clear way to use DES with a larger key length: intuitively "double DES"

- $C = E(E(P,K),K)$?
  - Problem: Still just 56 bit key

- $C = E(E(P,K_1),K_2)$?
  - There is an attack that is more-or-less equivalent to single DES
  - Although the attack is somewhat impractical, it' s close enough to being practical

Double DES Attack (Meet-in-the-Middle-Attack)



- 얼핏 보기에, 이중 DES는 키를 탐색하기 위한 탐색 횟수가 2^{56}번(하나의 DES에 대응)에서 2^{112} 번(두 개의 DES에 대응)으로 증가하는 것처럼 보인다.
- 그러나 중간일치공격과 같은 Known-plaintext attack을 사용하면 2^{112} 가 아닌 2^{57} 으로 약간의 향상만이 있을 뿐임을 증명할 수 있다.

Double DES Attack (Meet-in-the-Middle-Attack)

- $C = E(E(P,K_1),K_2)$ Attack: chosen plaintext attack
  - ✓ For particular P, precompute table of E(P,K) for every possible key K (resulting table has 2^{56} entries)
  - ✓ Then for each possible K_2, compute $D(C,K_2)$ until a match in table is found
    - Here, $M = D(C,K_2) \rightarrow E(M,K_2) = E(D(C,K_2), K_2) = C$, that is, $D(C,K_2)$ should be in the table
  - ✓ When match is found, have $E(P,K_1) = D(C,K_2) = M$
  - ✓ Result is keys: $C = E(E(P,K_1),K_2)$ where K_1 and K_2 are known, i.e. C is decrypted
  - ✓ Given this table(M= E(P,K), K) and C corresponding to chosen P

- Neglecting the work needed to precompute the table, the work consists of computing D(C,K) until we find a match in the table
- So, double DES is not secure

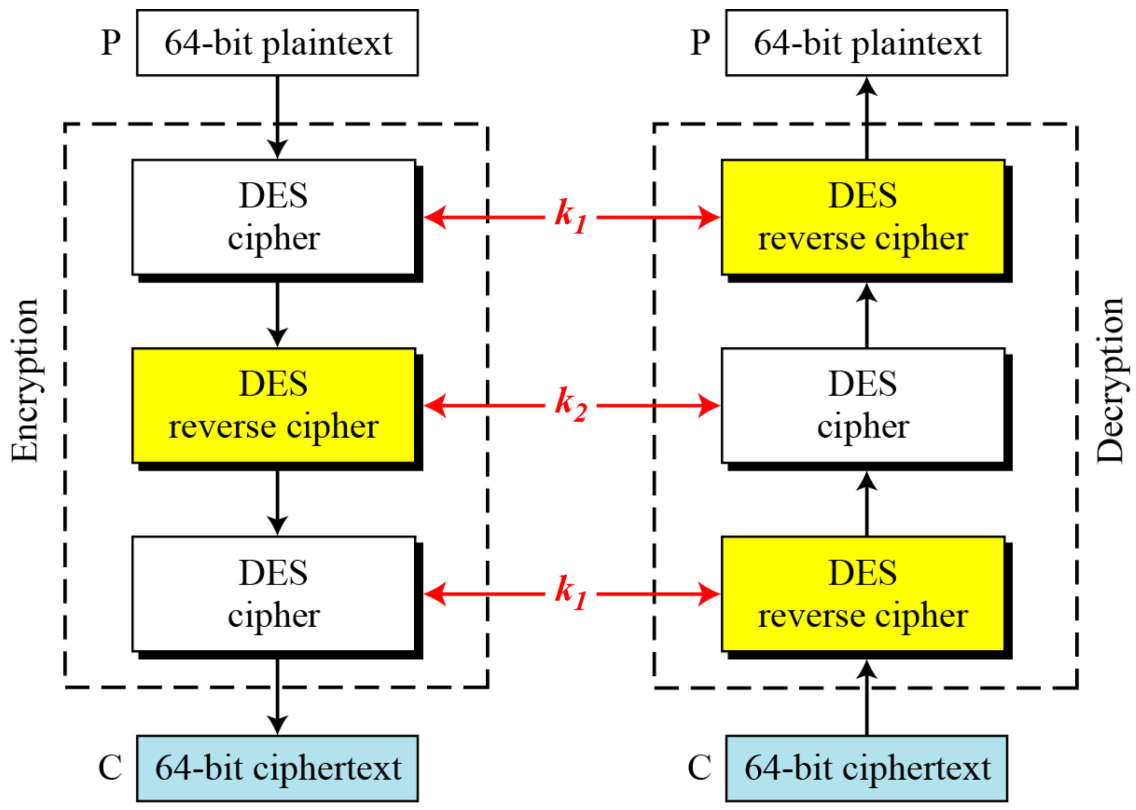
3. Block Ciphers – DES

Triple DES

- Logical approach to triple DES

- But practically, Triple DES is
  - ✓ $C = E[\ D[\ E[P,K_1],\ K_2],\ \ K_1]$
  - ✓ $P = D[\ E[\ D[C,K_1],\ K_2],\ \ K_1]$
  - ✓ (112 bit key)

✓ 2개의 키를 갖는 삼중 DES에 대한 Know-plaintext attac의 가능성 때문에 어떤 응용프로그램은 3개의 키(K_1,K_2,K_3)를 갖는 삼중 DES를 사용한다.



3. Block Ciphers – DES

Triple DES

- Why use Encrypt-Decrypt-Encrypt (EDE) with 2 keys? (Why not EEE and not 3 keys?)
  - Backward compatible with single DES:
    If K_1=K_2=K then E(D(E(P,K), K), K) = E(P,K)
  - And 112 bits is enough

3. Block Ciphers – DES

Python Code

- DES algorithm : C++, Java, Python

https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/