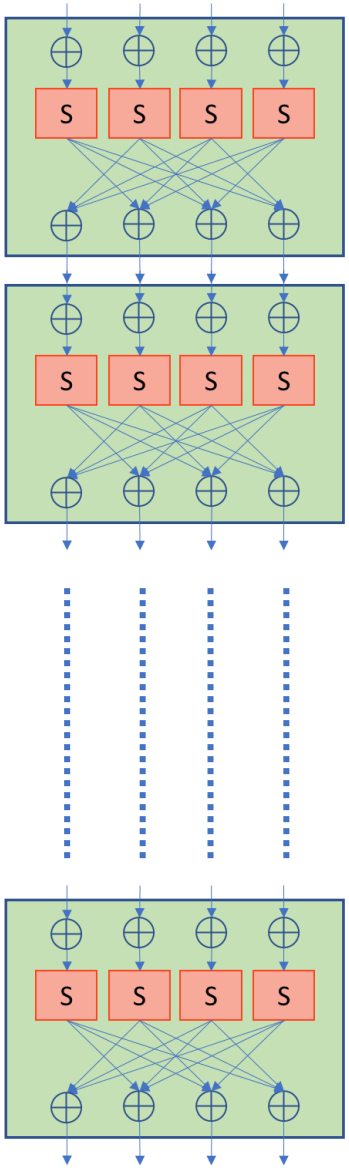
Cryptanalysis
(암호분석)

Chapter 4 – Part 1

2020.4

Contents

- Data type conversion
- Block cipher
- Toy Cipher TC20



바이트 표현

- 1바이트 데이터의 다양한 표현 방법
  - 1바이트는 2^8가지 데이터를 나타낼 수 있음
  - 10진수: 0, 1, 2, … , 255
  - 16진수: 00, 01, 02, …, FE, FF
  - ASCII 문자: 영문자, 숫자, 특수기호를 7비트로 표현한 것

- 예 (동일한 데이터의 여러가지 표현)
  - '1'(문자) ↔ 49(10진수) ↔ 31(16진수)
  - 'A'(문자) ↔ 65(정수) ↔ 41(16진수)
  - 'z'(문자) ↔ 122(정수) ↔ 7A(16진수)

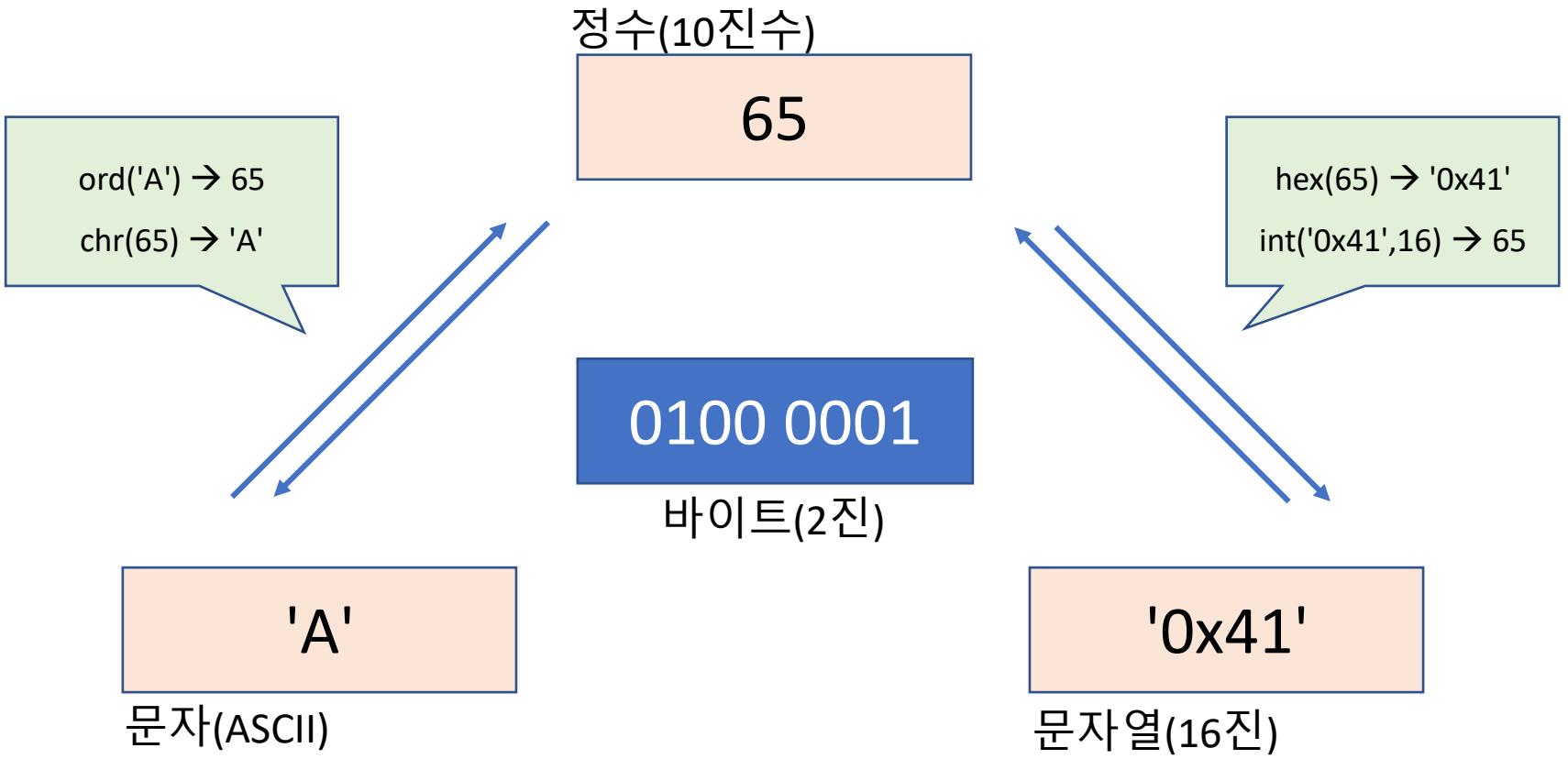
ASCII 코드표

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

바이트 데이터 변환함수


정수(10진수)
65
ord('A') → 65
chr(65) → 'A'
hex(65) → '0x41'
int('0x41',16) → 65
0100 0001
바이트(2진)
'A'
문자(ASCII)
'0x41'
문자열(16진)

바이트 데이터 변환함수

- 변환함수의 활용
  - ord(), chr(), hex(), int()

문자(ASCII)로 시작

```
ch1 = 'A'
num1 = ord(ch1)
hex1 = hex(num1)

list1 = []
list1.append(ch1)
list1.append(num1)
list1.append(hex1)

print(list1)
print(len(hex1))
```

```
['A', 65, '0x41']
```

정수(10진)로 시작

```
num2 = 66
ch2 = chr(num2)
hex2 = hex(num2)

list2 = []
list2.append(ch2)
list2.append(num2)
list2.append(hex2)

print(list2)
```

```
['B', 66, '0x42']
```

16진 문자열로 시작

```
hex3 = '0x43'
num3 = int(hex3,16)
ch3 = chr(num3)

list3 = []
list3.append(ch3)
list3.append(num3)
list3.append(hex3)

print(list3)
```

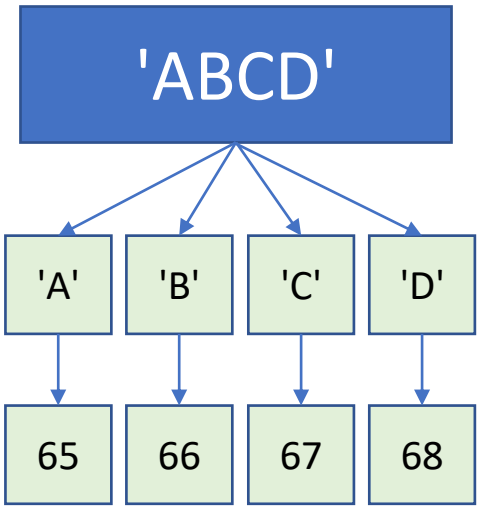
```
['C', 67, '0x43']
```

문자열 데이터

- 문자열을 리스트로
  - 문자열을 각 문자로 나누어 리스트로 만든다.
  - 향후 만들 블록암호의 입력으로 리스트를 사용한다.

```
str1 = 'ABCD'

list1 = [ch for ch in str1]
print(list1)

list2 = [ord(ch) for ch in str1]
print(list2)

list3 = list(str1)
```

list1과 list3는
같은 결과



str1
'ABCD'

list1
['A', 'B', 'C', 'D']

list2
[65, 66, 67, 68]

문자열 데이터

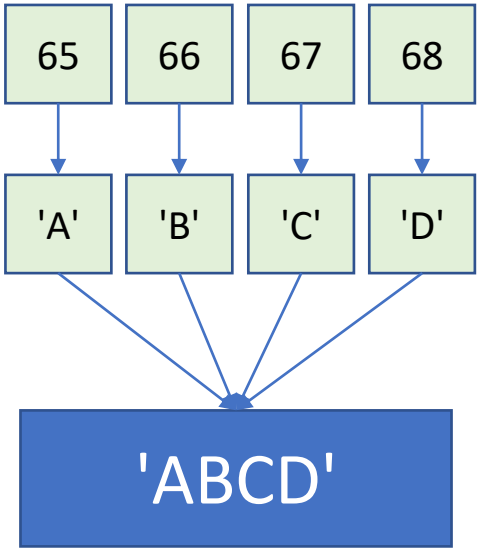
- 리스트를 바이트열/문자열로
  - 리스트의 내용을 합쳐 바이트열 또는 문자열로 만든다.
  - 향후 만들 블록암호의 출력인 리스트를 문자열로 변환한다.

```
list1 = [ 65, 66, 67, 68]
byte1 = bytes(list1)
str1 = byte1.decode('utf8')
print(str1)
```

UTF-8 (8-bit Unicode Transformation Format)
8비트 유니코드로 확장된 문자열을 다룬다.

```
list2 = [ 65, 66, 67, 68]
list3 = [chr(k) for k in list2]
str2 = ''.join(list3)
print(str2)
```

list2의 내용이 ASCII 문자열인 경우만 가능함!



list2
[65, 66, 67, 68]

list3
['A', 'B', 'C', 'D']

str2
'ABCD'

바이트열

- 바이트열 vs 문자열
  - 문자열(string): ASCII 문자로 구성된 배열
  - 바이트열(bytes): 바이트(0~255)로 구성된 배열
- 변환 함수
  - encode(): 문자열 → 바이트열
  - decode(): 바이트열 → 문자열
- 리스트와 문자열/바이트열 변환
  - list(), join(): 리스트 → 문자열
  - list(), bytes(): 리스트 → 바이트열

바이트열 예제

- 문자열, 바이트열, 리스트 변환 예제

```
list1 = [65, 66, 67, 68]
byte1 = bytes(list1)
str1 = byte1.decode('utf8')
print(str1)
```

```
ABCD
```

```
str2 = 'ABCDE'
bytes2 = bytes(str2, 'utf8')
print(bytes2)
```

```
b'ABCDE'
```

```
str3 = 'ABCDE'
bytes3 = str3.encode('utf8')
list_s3 = list(str3)
list_b3 = list(bytes3)
print(list_s3)
print(list_b3)
```

bytes(str3, 'utf8')
와 동일한 결과

```
['A', 'B', 'C', 'D', 'E']
[65, 66, 67, 68, 69]
```

파일 입출력

- 데이터를 문자열 또는 바이트열로 변환 후 파일에 출력
  - 문자열 출력: ASCII 문자를 텍스트(text) 파일로 출력
  - 바이트열 출력: 바이트열을 이진(binary) 파일로 출력

```
list1 = [1, 2, 3, 4, 65, 66, 67, 68]
byte1 = bytes(list1)

f = open('data1.txt', 'w+b')
f.write(byte1)
f.close()
```

이진파일에는
'b' 를 추가해야 한다

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   01 02 03 04 41 42 43 44                          ....ABCD
```

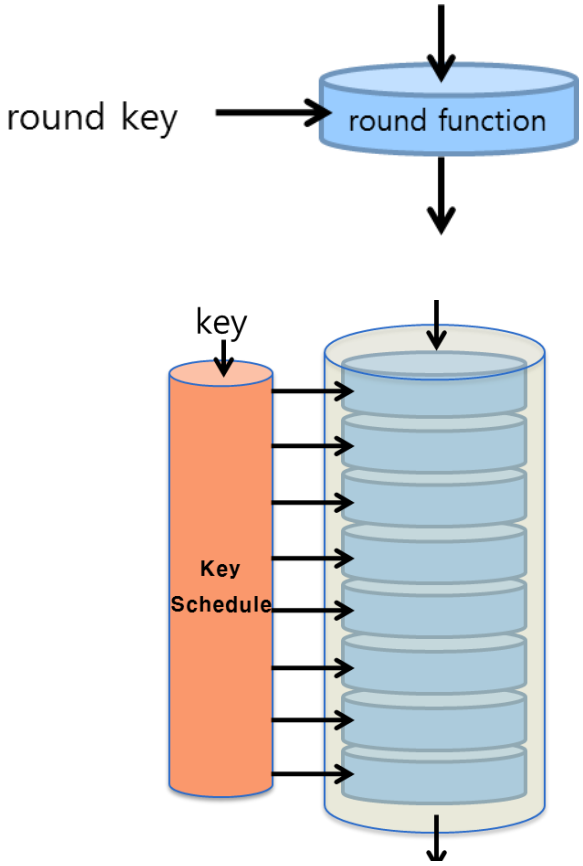
Hex Editor
프로그램으로
확인한 파일저장 결과

```
list2 = ['A', 'B', 'C', 'D']
str2 = ''.join(list2)

f = open('data2.txt', 'w+')
f.write(str2)
f.close()
```

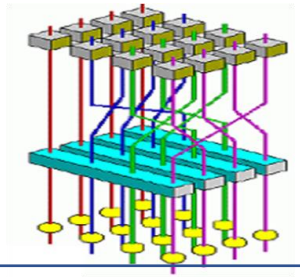
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   41 42 43 44                                      ABCD
```

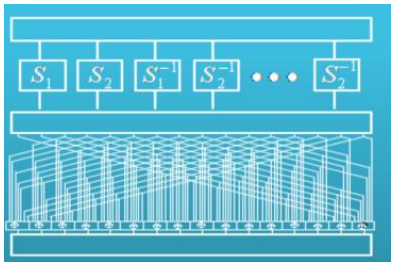
블록암호 개요

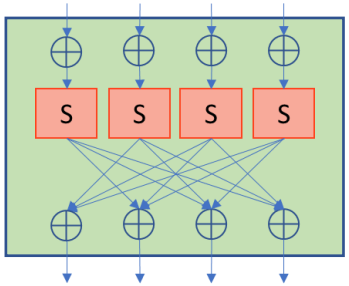
- Building block (구성요소)
  - Round function (라운드 함수)
  - Key schedule (키 스케줄)
- Design consideration
  - Security and Efficiency
  - Determine the number of rounds with security margin

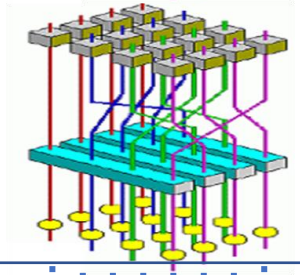


블록암호의 구조

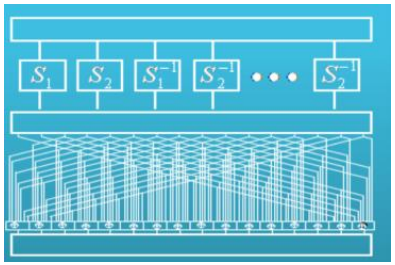


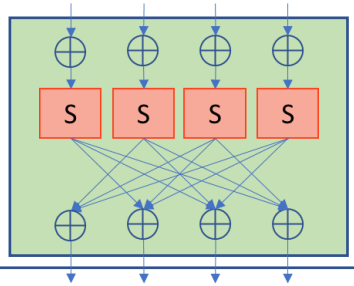


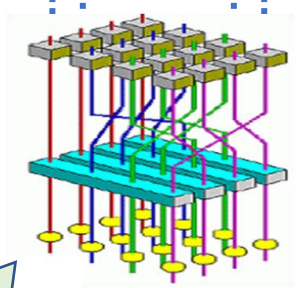


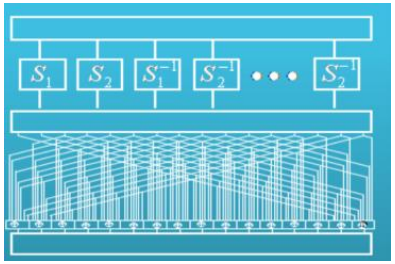


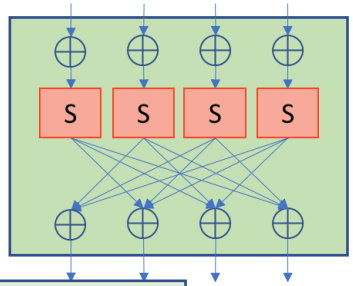
Round function











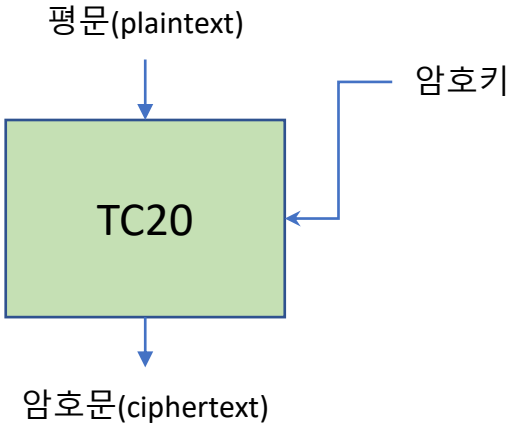
미국 표준 AES

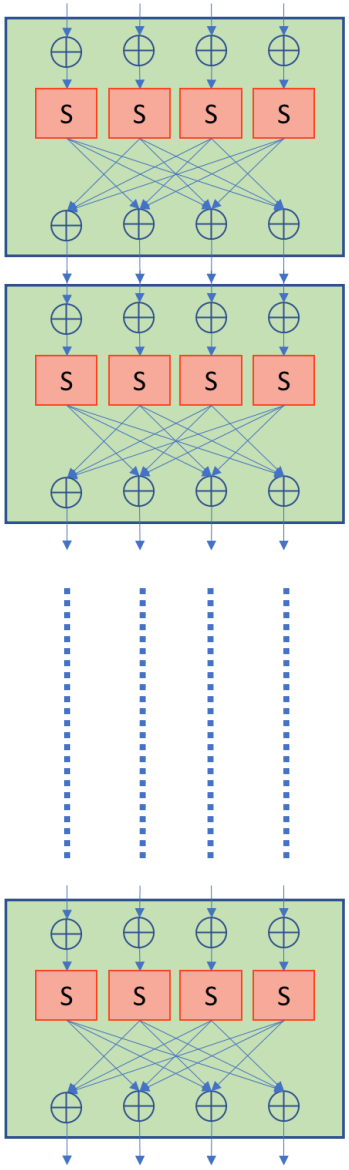
한국 KS표준 ARIA

연습용 Toy Cipher TC20

Toy Cipher – TC20

- TC20 블록암호
  - 블록 크기: 32비트
  - 키 크기: 32비트
  - 라운드 수: 10
  - 구조: SPN
    (Substitution Permutation Network)





TC20 – AR (라운드키 적용)

- 라운드키 RK의 적용 방법
  - 라운드키: $RK = (rk_0, rk_1, rk_2, rk_3)$
  - XOR(Exclusive or)를 이용한 라운드 키 덧셈

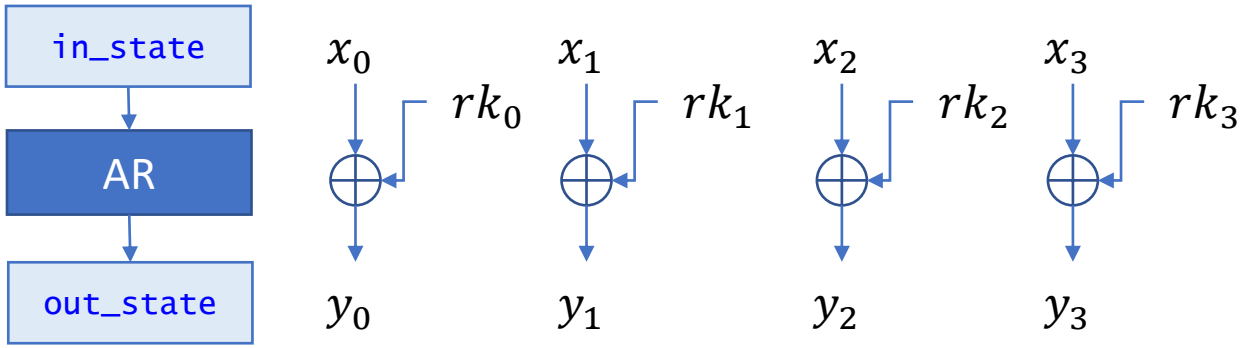
$$(y_0, y_1, y_2, y_3) = (x_0 \oplus rk_0, x_1 \oplus rk_1, x_2 \oplus rk_2, x_3 \oplus rk_3)$$

- 키 스케줄
  - 단순한 구조를 위해 키 스케줄을 사용하지 않음
  - 라운드키를 암호키 32비트와 동일한 값으로 사용함

TC20 – AR (라운드키 적용)

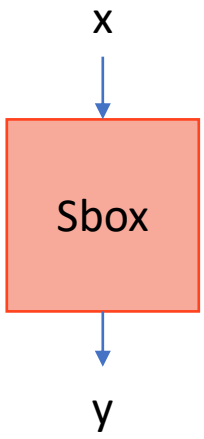
- AR (Add Roundkey) 연산

```
##-- Add Roundkey 함수
def AR(in_state, rkey):
    out_state = [0,0,0,0]
    for i in range(len(in_state)):
        out_state[i] = in_state[i] ^ rkey[i]

    return out_state
```



TC20 – Sbox (비선형 변환)

- Sbox 구조
  - AES와 동일한 Sbox 사용 (8비트 → 8비트)
  - 각 라운드마다 4개의 Sbox를 각 바이트에 적용

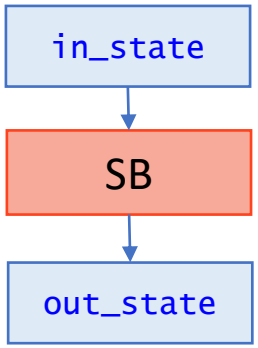


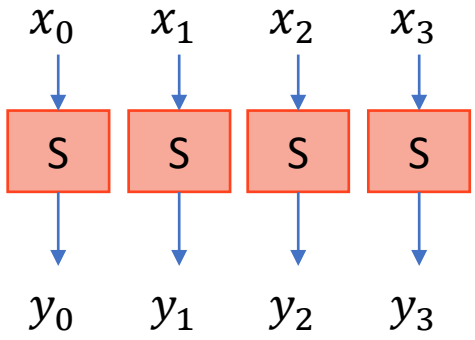

```
Sbox = [
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
        ]
```

TC20 – Sbox (비선형 변환)

- SB (Sbox) 연산 – 바이트 단위의 비선형 변환

```
##-- Sbox layer 함수
def SB(in_state):
    out_state = [0,0,0,0]
    for i in range(len(in_state)):
        out_state[i] = Sbox[in_state[i]]

    return out_state
```





TC20 – LM (선형변환)

- 선형함수 LM(Linear Map)
  - 4바이트에 대한 바이트 단위의 선형변환
  - 4x4 이진(binary)행렬 A로 표현되는 함수: $Y = AX$

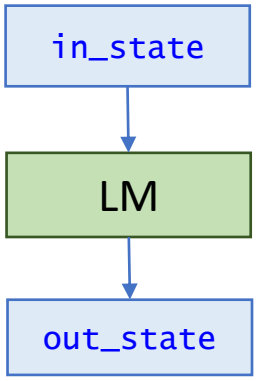
$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \oplus x_2 \oplus x_3 \\ x_0 \oplus x_2 \oplus x_3 \\ x_0 \oplus x_1 \oplus x_3 \\ x_0 \oplus x_1 \oplus x_2 \end{pmatrix}$$

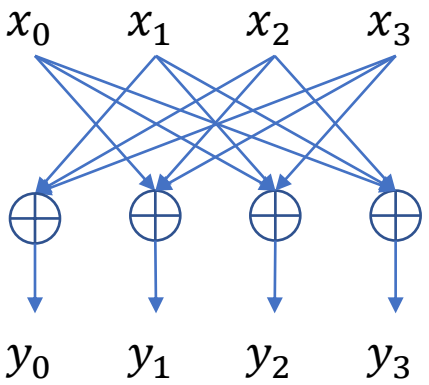
- 행렬의 특징
  - Involution ($A^2 = I$) : $A^{-1} = A$
  - 암호화, 복호화에 동일한 선형함수를 사용함

TC20 – LM (선형변환)

- LM (Linear Map) – 선형 변환

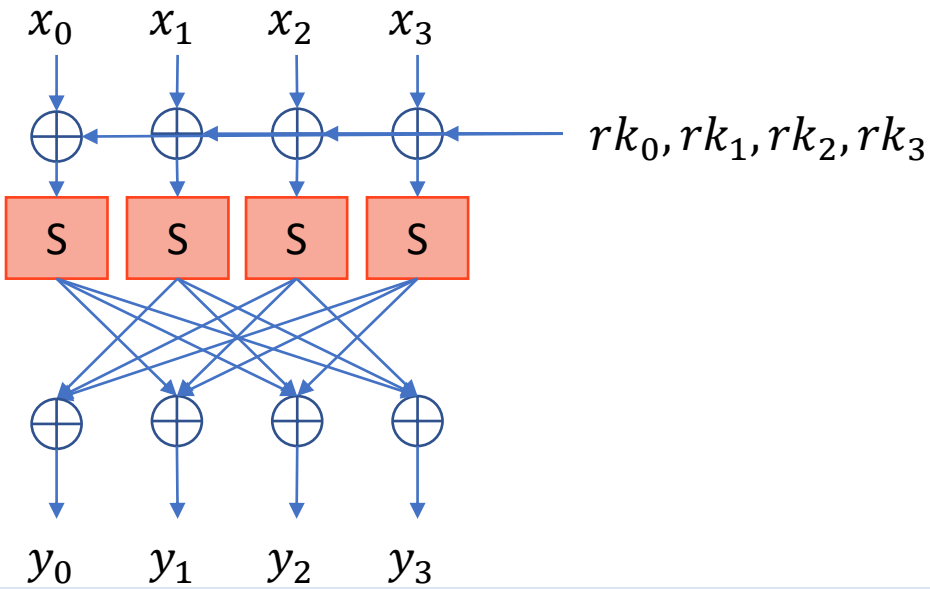
```
##-- Linear Mixing 함수
def LM(in_state):
    out_state = [0,0,0,0]
    all_xor = in_state[0] ^ in_state[1] ^ in_state[2] ^ in_state[3]
    for i in range(len(in_state)):
        out_state[i] = all_xor ^ in_state[i]

    return out_state
```





TC20 – 라운드 함수

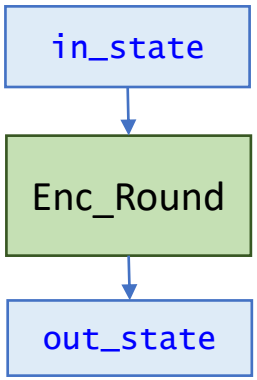
- 라운드 구조
  - AR: 라운드키 적용
  - Sbox: 각 바이트에 비선형 Sbox 적용
  - LM: 선형변환

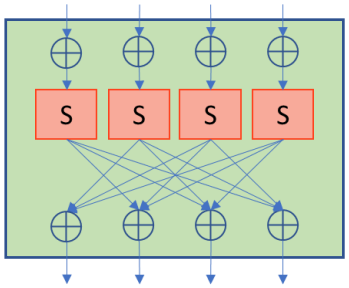


TC20 – 라운드 함수

- 라운드 함수

```
##-- 라운드 암호화 함수
def Enc_Round(in_state, rkey):
    out_state = [0,0,0,0]
    out_state = AR(in_state, rkey)
    in_state = SB(out_state)
    out_state = LM(in_state)

    return out_state
```





Toy Cipher – TC20

- TC20의 특징 – 극단적인 단순화
  - 작은 블록, 키 크기: 32비트
  - 키 스케줄 없음: 라운드키=암호키
  - 출력의 난수성: 통계적으로는 우수함 (10라운드)
  - SPN구조: 복호화를 위해 모든 단계의 역연산을 만들어야 함 (단, 선형함수 LM은 역함수가 자기자신과 같다)

- TC20 설계의 문제점
  - 32비트 암호키의 전수조사 공격(brute force attack) 가능
  - 마지막 AR이후 연산은 사실상 불필요함 (why?)

TC20 – 활용 방향

- TC20을 이용한 암호분석
  - TC20의 복호화 알고리즘 (과제)
  - 64비트 Feistel 구조의 암호로 확장 (라운드 함수 공유)
  - 다양한 암호분석 기술의 이해를 위한 예제로 활용