CHAPTER 3
SYMMETRIC KEY CRYPTO

BRAC UNIVERSITY
Inspiring Excellence

PREPARED BY:

DR. MUHAMMAD IQBAL HOSSAIN

ASSOCIATE PROFESSOR

DEPARTMENT OF CSE, BRAC UNIVERSITY

APPENDIX

STREAM CIPHERS

BLOCK CIPHERS

BLOCK CIPHER MODES

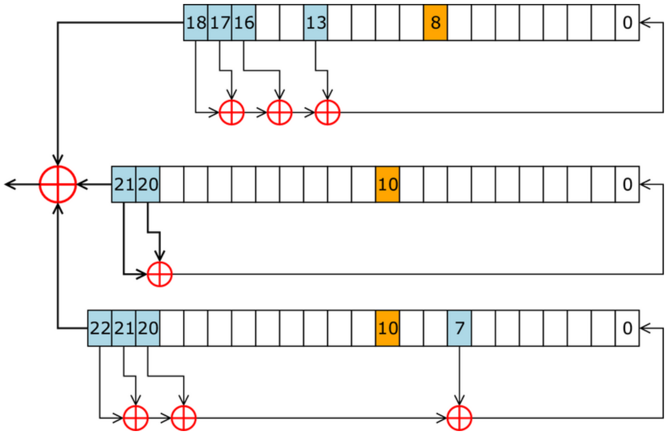
INTEGRITY

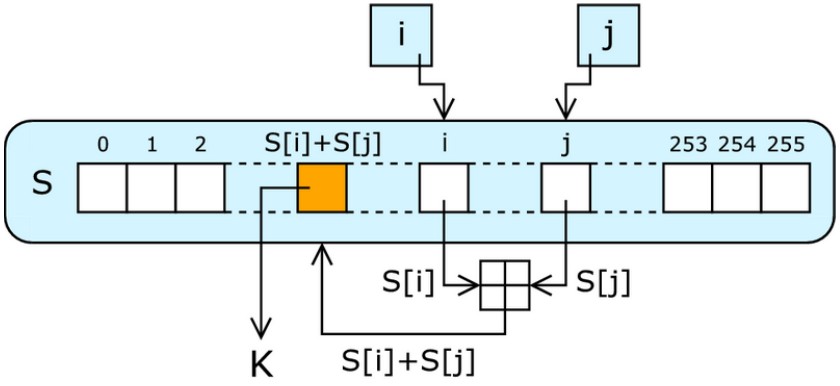
SYMMETRIC KEY CRYPTO

- 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

SYMMETRIC KEY CRYPTO

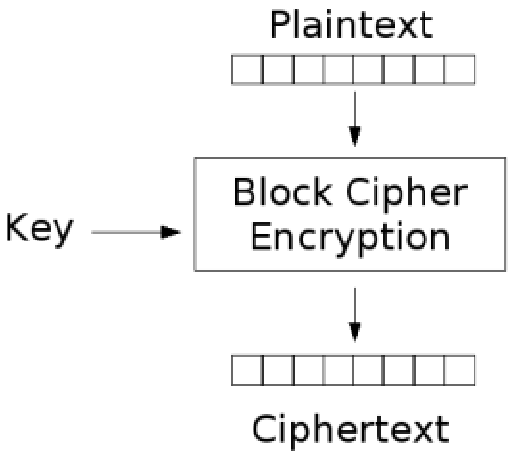
- Examples of Stream cipher
  - A5/1: employed GSM cell phones
    - Representative stream cipher based in H/W (shift register)
  - RC4: used SSL protocol (lookup table)
    - Almost unique stream cipher since efficiently implemented in S/W

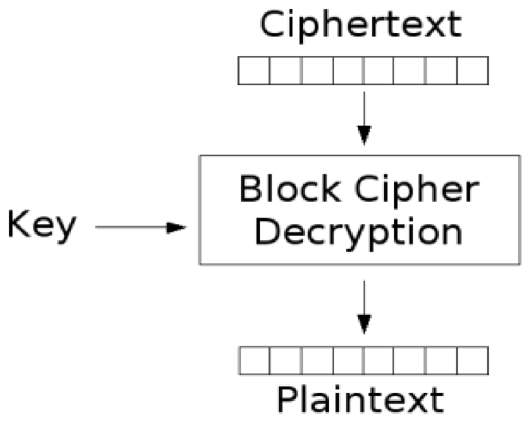




SYMMETRIC KEY CRYPTO

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





SYMMETRIC KEY CRYPTO

- Examples of Block cipher
  - Data Encryption Stantard(DES): relatively simple,
  - Advanced Encryption STD(AES)
  - International Data Encrytption Alg.(IEDA)
  - Blowfish,
  - RC6
  - Tiny Encryption Algorithm

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

- 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 is used like a one-time pad
    - $c_0 = p_0 \oplus s_0, c_1 = p_1 \oplus s_1, c_2 = p_2 \oplus s_2, \ldots$
    - $p_0 = c_0 \oplus s_0, p_1 = c_1 \oplus s_1, p_2 = c_2 \oplus s_2, \ldots$
- Sender and receiver have same stream cipher algorithm and both know the key K

STREAM CIPHERS

- A5/1
  - 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
  - Based on a changing lookup table
  - Used many places – SSL

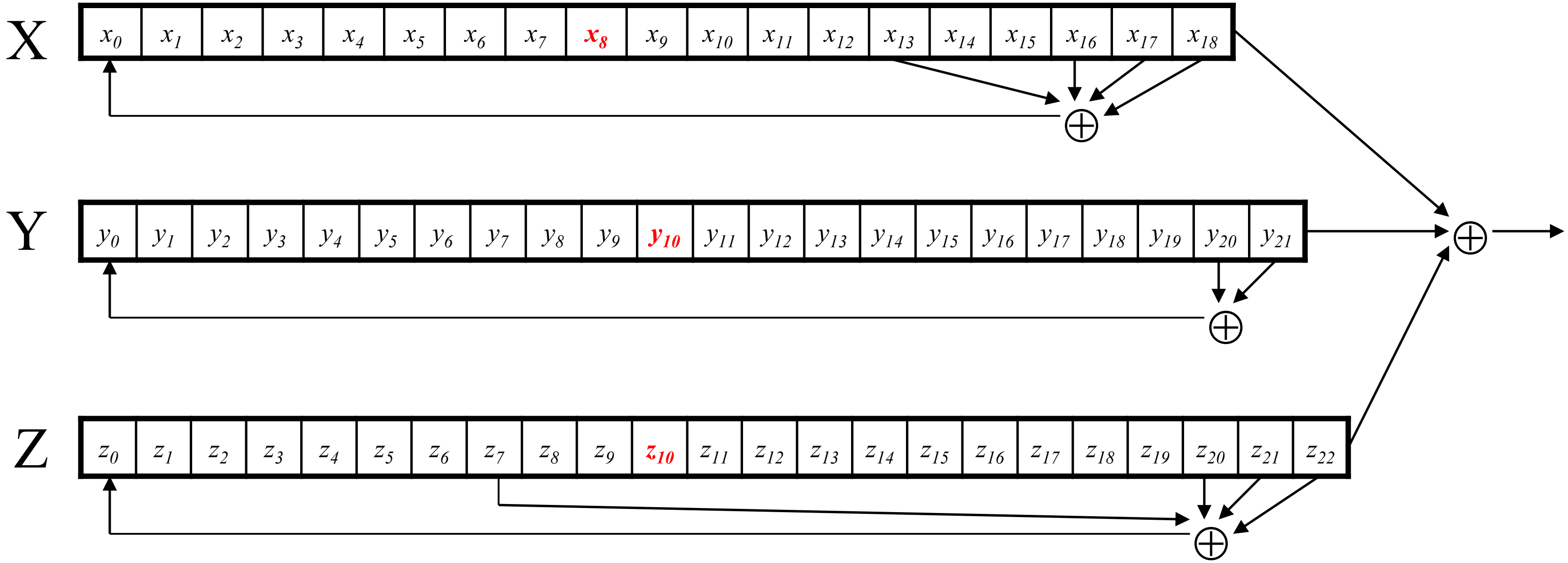
A5/1

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

A5/1

- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$;major
  - 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 $i = 18, 17, \ldots, 1$ and $x_0 = t$
- If $y_{10} = m$ then Y steps
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21, 20, \ldots, 1$ 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 $i = 22, 21, \ldots, 1$ and $z_0 = t$
- **Keystream bit is $x_{18} \oplus y_{21} \oplus z_{22}$**

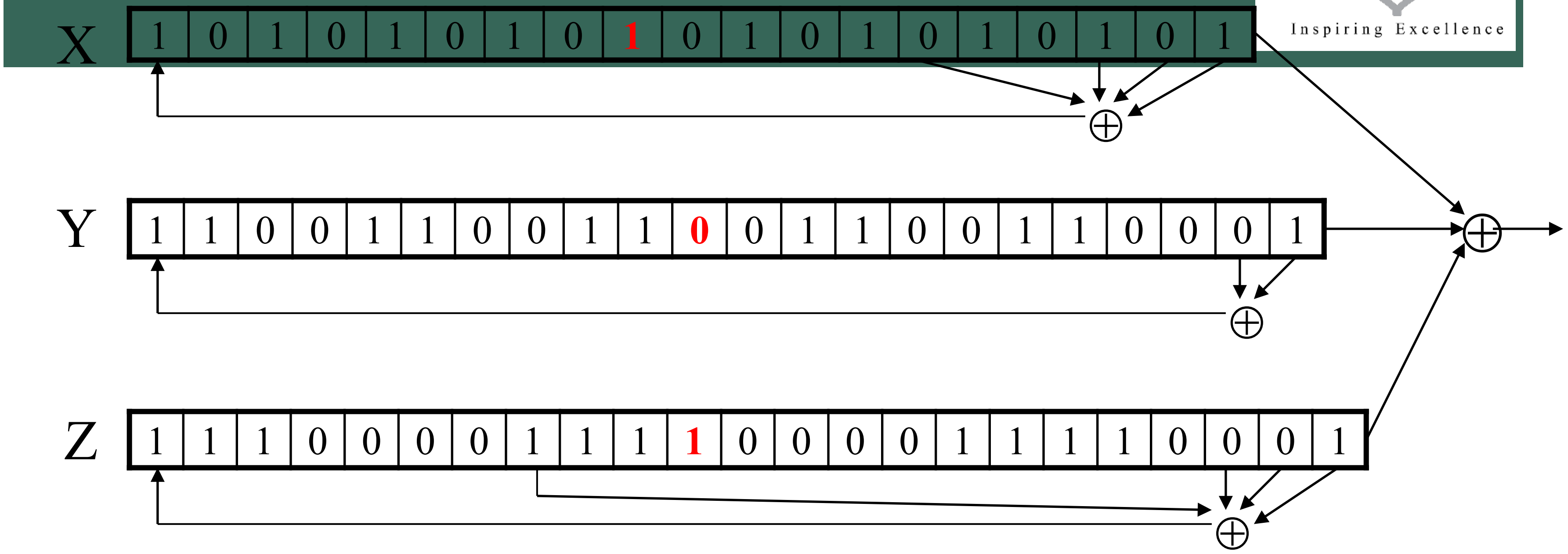
A5/1



X
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17} x_{18}
Y
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12} y_{13} y_{14} y_{15} y_{16} y_{17} y_{18} y_{19} y_{20} y_{21}
Z
z_0 z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10} z_{11} z_{12} z_{13} z_{14} z_{15} z_{16} z_{17} z_{18} z_{19} z_{20} z_{21} z_{22}
$\oplus$

- Each value is a single bit
- Key is used as **initial fill** of registers
- Each register steps or not, based on (x_8, y_{10}, z_{10})
- Keystream bit is XOR of right bits of registers

A5/1



X 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Y 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 1

Z 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1

- In this example, $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(1,0,1) = 1$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be $0 \oplus 1 \oplus 0 = 1$

SHIFT REGISTER CRYPTO

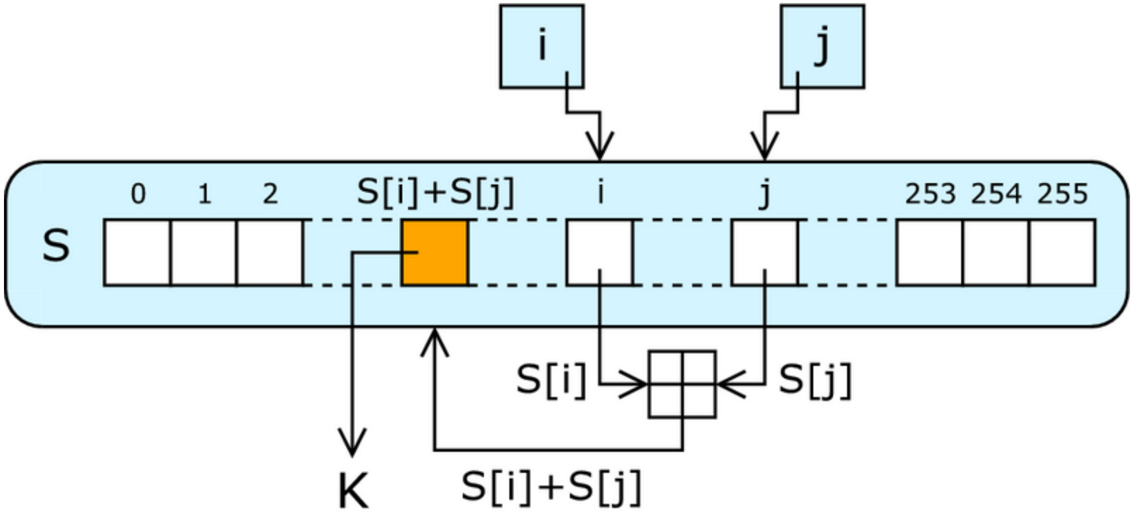
- Shift register crypto efficient in hardware
- Often, slow if implemented in software
- In the past, very, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used some
  - Especially in resource-constrained devices

RC4

- A self-modifying lookup table
- Table always contains a permutation of the byte values 0,1,…,255
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a **byte**
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware

RC4

- 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

- 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}

STREAM CIPHERS

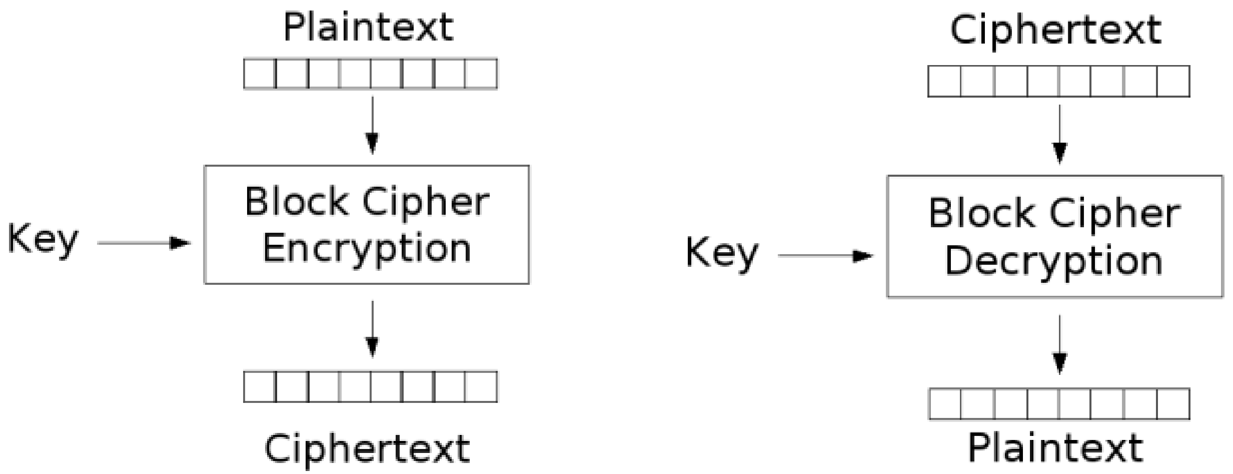
- Stream ciphers were big in the past
  - Efficient in hardware
  - Speed needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is fast enough
- Future of stream ciphers?
  - Shamir: "the death of stream ciphers"
  - May be exaggerated…

End of segment

BLOCK CIPHER AND DES

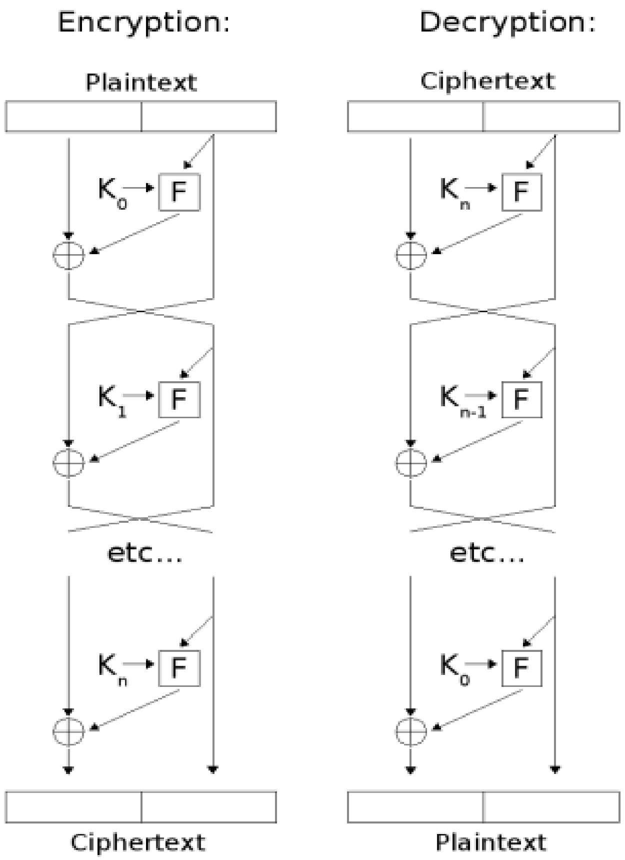
BLOCK CIPHER

- 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



(ITERATED) BLOCK CIPHER

- Plaintext and Ciphertext consist of fixed-sized blocks
- 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

FEISTEL CIPHER

- Feistel cipher refers to a type of block cipher design, not a specific cipher
- Split plaintext block into left and right halves: Plaintext = (L_0,R_0)
- For each round $i=1,2,\ldots,n$, compute

  $L_i= R_{i-1}$

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

  where F is **round function** and K_i is **subkey**
- Ciphertext = (L_n,R_n)

FEISTEL CIPHER

- Decryption: Ciphertext = (L_n,R_n)
- For each round $i=n,n-1,\ldots,1$, compute

  $R_{i-1} = L_i$

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

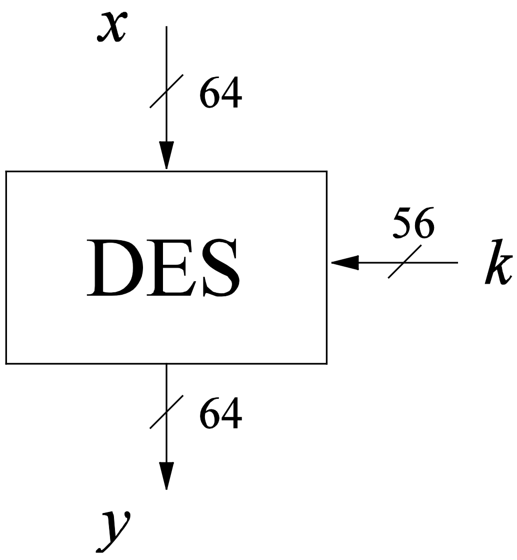
  where F is round function and K_i is subkey
- Plaintext = (L_0,R_0)
- Formula "works" for any function F
- But only secure for certain functions F
  - Ex: $F(R_{i-1}, K_i) = 0$ for all R_{i-1} and K_i -> not secure

DATA ENCRYPTION STANDARD (DES)

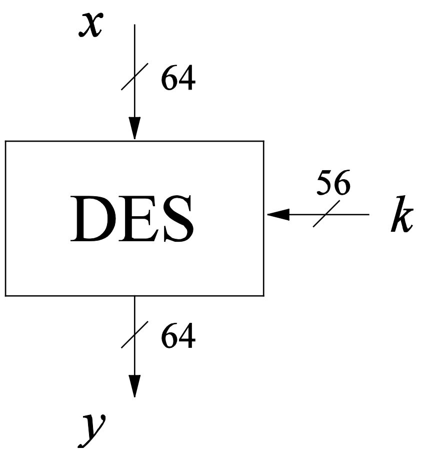
- DES developed in 1970's
- Based on IBM Lucifer cipher
- U.S. government standard
- DES development was controversial
  - NSA was secretly involved
  - Design process not open
  - Key length was reduced
  - Subtle changes to Lucifer algorithm

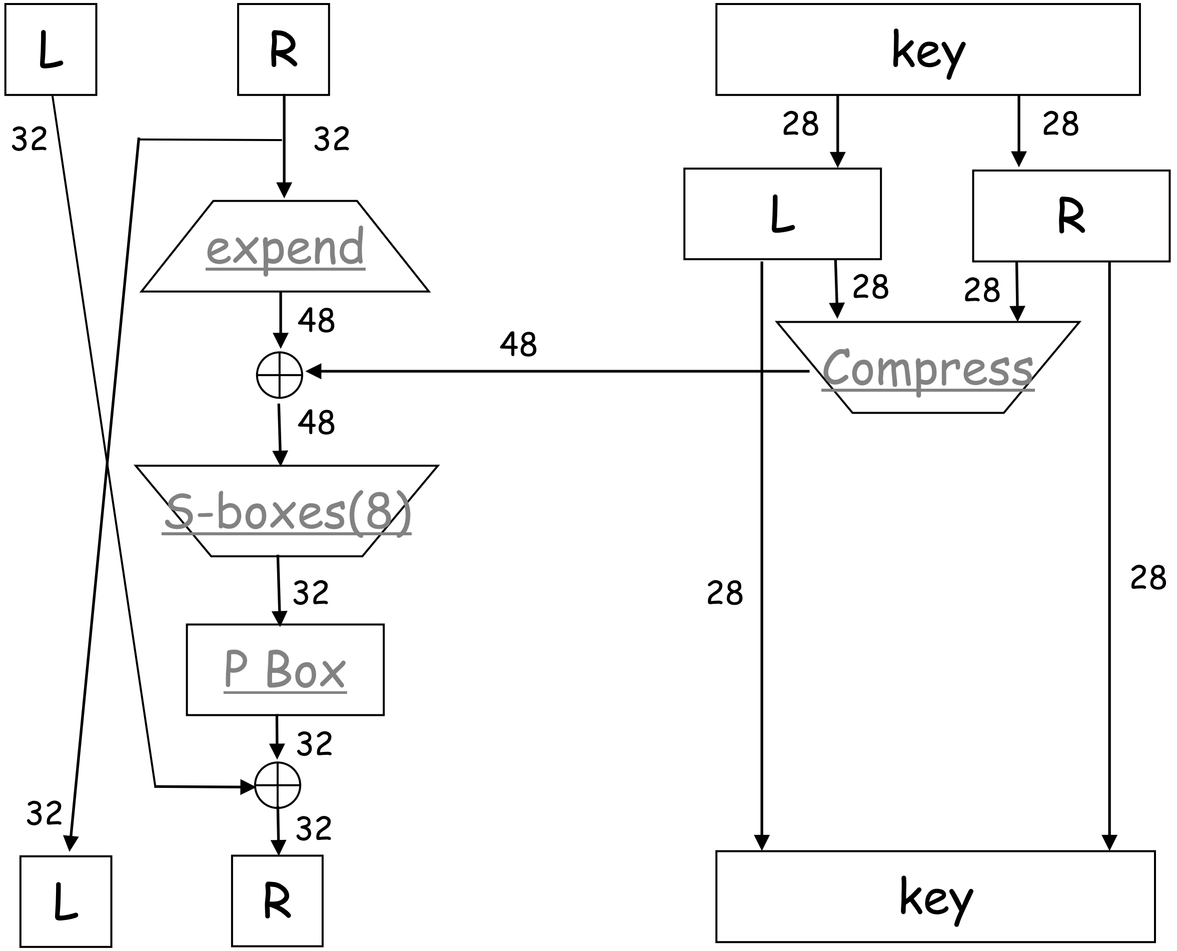
DES NUMEROLOGY

- 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

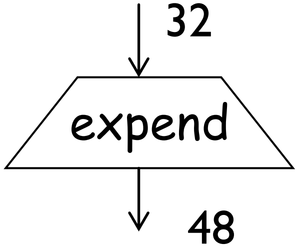


ONE ROUND OF DES





DES EXPANSION PERMUTATION



- Input 32 bits

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

- Output 48 bits

index	0	1	2	3	4	5	6	7	8	9	10	11
index	12	13	14	15	16	17	18	19	20	21	22	23
index	24	25	26	27	28	29	30	31	32	33	34	35
index	36	37	38	39	40	41	42	43	44	45	46	47

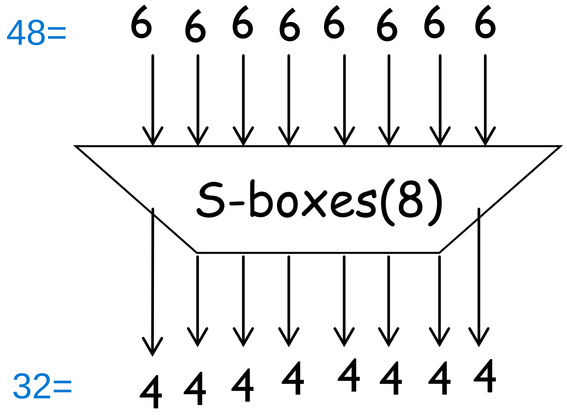
Input 32 bits

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

- Output 48 bits

index	0	1	2	3	4	5	6	7	8	9	10	11
	31	0	1	2	3	4	3	4	5	6	7	8
index	12	13	14	15	16	17	18	19	20	21	22	23
	7	8	9	10	11	12	11	12	13	14	15	16
index	24	25	26	27	28	29	30	31	32	33	34	35
	15	16	17	18	19	20	19	20	21	22	23	24
index	36	37	38	39	40	41	42	43	44	45	46	47
	23	24	25	26	27	28	27	28	29	30	31	0

DES S-BOX



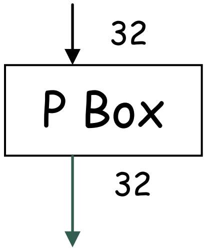
- 8 “substitution boxes” or S-boxes
- Each S-box maps 6 bits to 4 bits
- S-box number 1

input bits (0,5)

input bits (1,2,3,4)

	00 00	00 01	00 10	00 11	01 00	01 01	01 10	01 11	10 00	10 01	10 10	10 11	11 00	11 01	11 10	11 11
0 0	11 10	01 00	11 01	00 01	00 10	11 11	10 11	10 00	00 11	10 10	01 10	11 00	01 01	10 01	00 00	01 11
0 1	00 00	11 11	01 11	01 00	11 10	00 10	11 01	00 01	10 10	01 10	11 00	10 11	10 01	01 01	00 11	10 00
1 0	01 00	11 01	11 10	10 00	11 01	01 10	00 10	10 11	11 11	11 00	10 01	01 11	00 11	10 10	01 01	00 00
1 1	11 11	11 00	10 00	00 10	01 00	10 01	00 01	01 11	01 11	10 11	00 11	11 10	10 10	00 00	01 10	11 01

DES P-BOX



- Input 32 bits

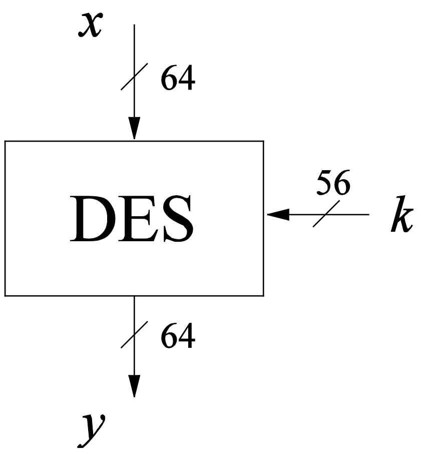
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

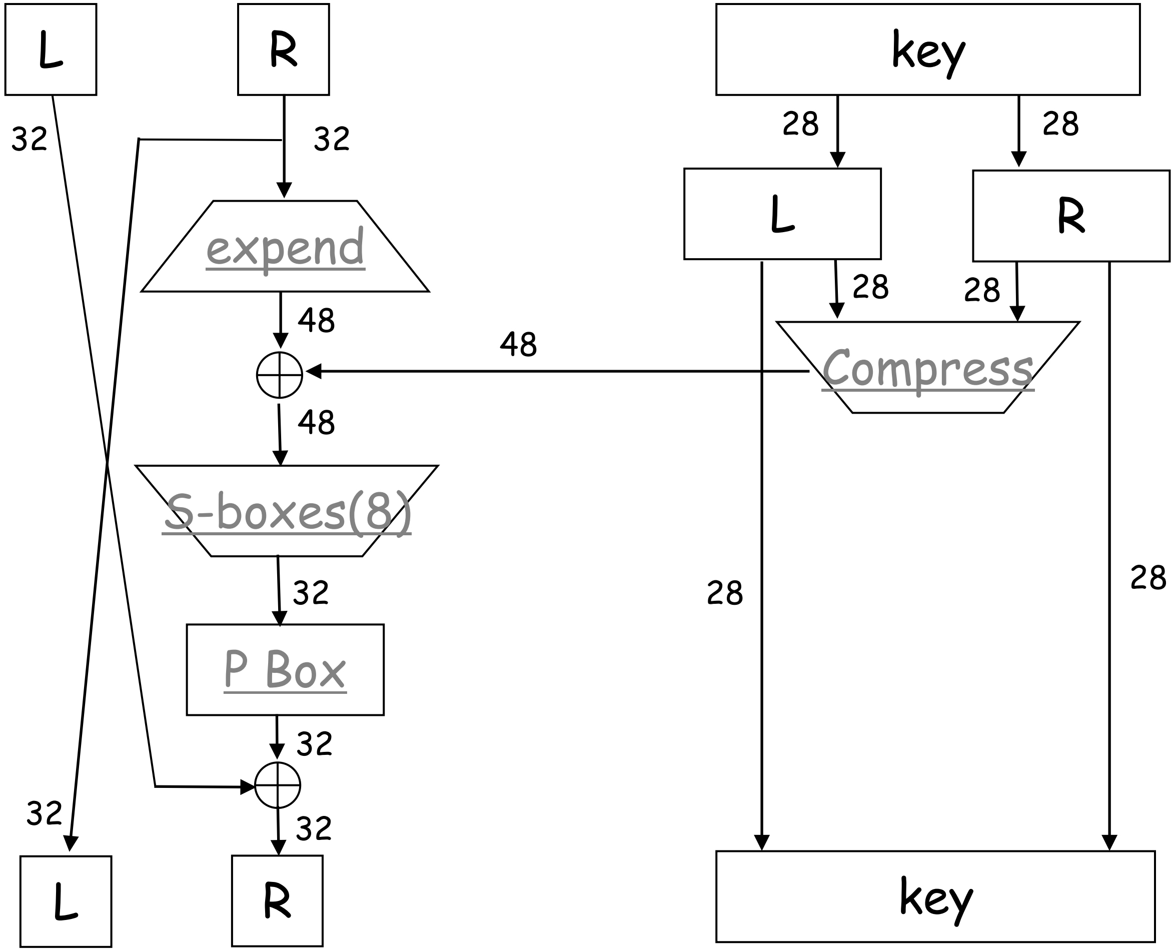
- Output 32 bits

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

BACK

DES SUBKEY


x
64
DES
56
k
64
y


L
R
key
32
32
28
28
L
R
expend
48
28
28
48
Compress
48
S-boxes(8)
32
P Box
28
28
32
32
32
L
R
key

DES SUBKEY

- 56 bit DES key, numbered 0,1,2,…,55
  - Left half key bits, 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, 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

DES SUBKEY

- 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 of LK bits

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

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

DES SUBKEY

- 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

BACK

DES LAST WORD (ALMOST)

- An initial perm 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

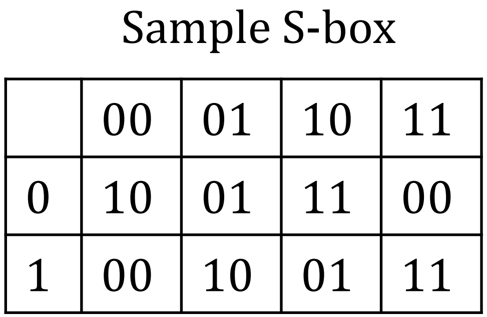
- Security of DES depends a lot on S-boxes
  - Everything else in DES is linear
- Thirty years of intense analysis has revealed no "back door"
- Attacks today use exhaustive key search
- Inescapable conclusions
  - Designers of DES knew what they were doing
  - Designers of DES were ahead of their time

HISTORY OF ATTACKS ON DES

Year	Proposed/ implemented DES Attack
1977	Diffie & Hellman, (under-)estimate the costs of a key search machine
1990	Biham & Shamir propose differential cryptanalysis (2^{47} chosen ciphertexts)
1993	Mike Wiener proposes design of a very efficient key search machine: Average search requires 36h. Costs: $1.000.000
1993	Matsui proposes linear cryptanalysis (2^{43} chosen ciphertexts)
Jun. 1997	DES Challenge I broken, 4.5 months of distributed search
Feb. 1998	DES Challenge II--1 broken, 39 days (distributed search)
Jul. 1998	DES Challenge II--2 broken, key search machine *Deep Crack* built by the Electronic Frontier Foundation (EFF): 1800 ASICs with 24 search engines each, Costs: $250 000, 15 days average search time (required 56h for the Challenge)
Jan. 1999	DES Challenge III broken in 22h 15min (distributed search assisted by *Deep Crack*)
2006-2008	Reconfigurable key search machine *COPACOBANA* developed at the Universities in Bochum and Kiel (Germany), uses 120 FPGAs to break DES in 6.4 days (avg.) at a cost of $10 000.

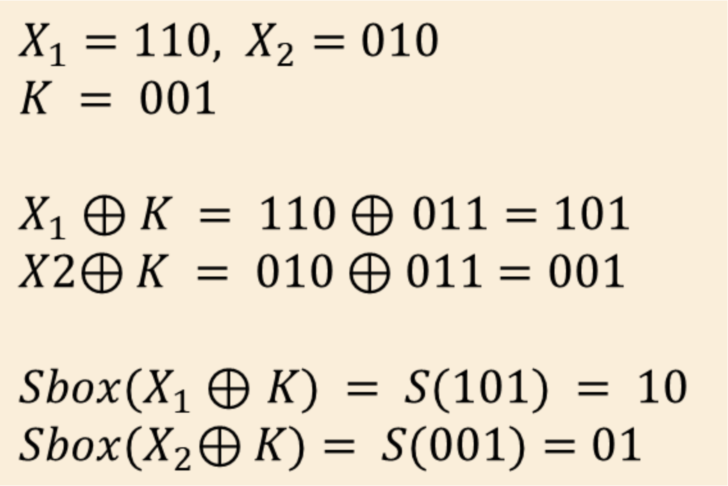
Thank you

BREAKING S-BOX



Sample S-box

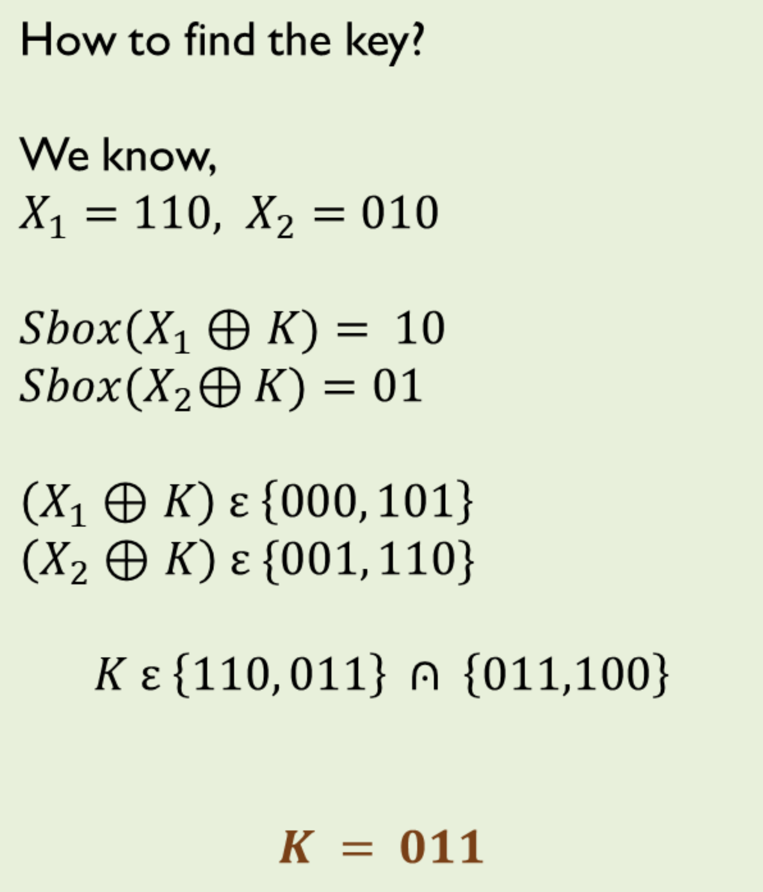
	00	01	10	11
0	10	01	11	00
1	00	10	01	11



$$X_1 = 110,\ X_2 = 010$$
$$K = 001$$

$$X_1 \oplus K = 110 \oplus 011 = 101$$
$$X2 \oplus K = 010 \oplus 011 = 001$$

$$Sbox(X_1 \oplus K) = S(101) = 10$$
$$Sbox(X_2 \oplus K) = S(001) = 01$$



How to find the key?

We know,

$$X_1 = 110,\ X_2 = 010$$

$$Sbox(X_1 \oplus K) = 10$$
$$Sbox(X_2 \oplus K) = 01$$

$$(X_1 \oplus K)\ \varepsilon \{000, 101\}$$
$$(X_2 \oplus K)\ \varepsilon \{001, 110\}$$

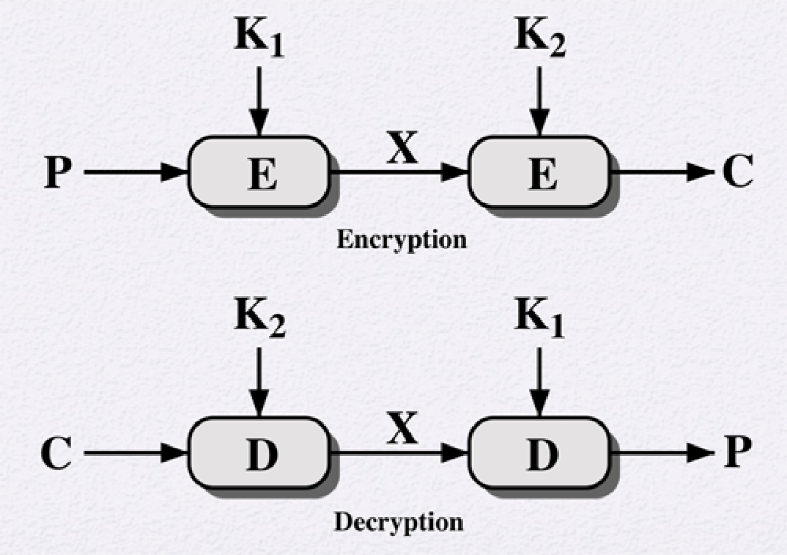
$$K\ \varepsilon \{110, 011\} \cap \{011, 100\}$$

$$\mathbf{K = 011}$$

CRYPTANALYSIS

Check Lecture 2-1 first.

DOUBLE DES AND MEET IN THE MIDDLE ATTACK



For DES, Key length is 56.
Hacker needs to check 2^{56} combination in brute force attack.
In 2DES, Key length 56 + 56 = 112
It only works when there is a known plaintext/ciphertext pair.

1. Encrypt the plaintext with all 2^{56} possible keys and write down the results
2. Decrypt the ciphertext with all 2^{56} possible keys and write down the results
3. Check where the results are the same. That is your key.

Note that all you had to do to recover the key was using DES 2×2^{56} times, which makes 2^{57}.

Block Cipher: AES

ADVANCED ENCRYPTION STANDARD (AES) HISTORY

- Needs for replacement for DES
  - DES had outlived its usefulness
    - Attacked by exhaustive key search: Special purpose DES crackers and distributed attack at internet
  - 3DES is very resistant to crypto analysis but
    - No efficient software code
    - Too slow: 3 times as many rounds as DES
    - 3DES use 64-bit block size: for reasons of both efficient and security, a larger blk sixe desirable
    - So, 3DES is not solution for long-term use
- In 1997, NIST made a formal call for advanced encryption standard algorithms

AES HISTORY

- GOAL: replace DES for both government and private sector encryption.
- Requirement of AES
  - Unclassified, publicly disclosed encryption algorithm, available royalty-free, worldwide.
  - The algorithm must implement symmetric key
  - Cryptography as a block cipher and (at a minimum) support block sizes of 128-bits and key sizes of 128- , 192-, and 256-bits.
- In 1998, NIST announced a group of 15 AES candidate algorithms.

AES HISTORY

- Criteria for selecting AES:
  - Security, Robustness, Speed
- In 1999, out of 15, the selection was narrowed to 5 candidates:
  - MARS, RC6, Rijndael, Serpent, and Twofish.
- All the five protocols were thought to be secure
- On October 2, 2000, NIST has selected Rijndael to propose for the AES.
  - Pronounced like "Rain Doll" or "Rhine Doll"
  - Invented by Joan Daemen and Vincent Rijmen

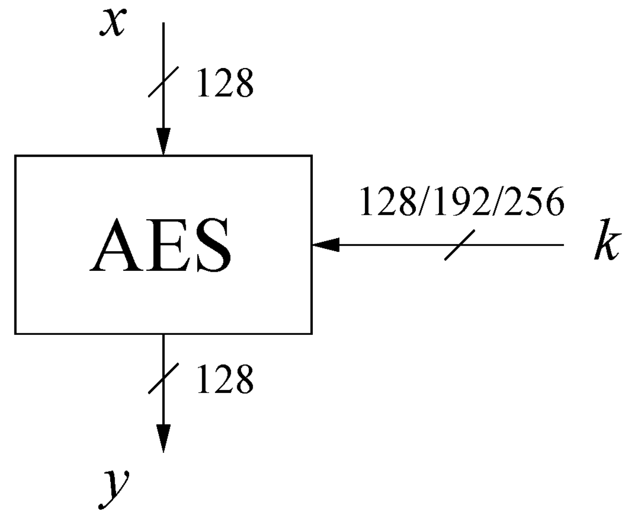
AES FEATURES

- Designed to be efficient in both hardware and software across a variety of platforms.
- Not a Feistel Network
  - Iterated block cipher (like DES)
  - Not a Feistel cipher (unlike DES)
- "Secure forever" – Shamir

AES OVERVIEW

- **Block size:** 128 bits (others in Rijndael)
- **Key length:** 128, 192 or 256 bits (independent of block size in Rijndael)
- 10 to 14 rounds (depends on key length)
- Each round uses 4 functions (3 “layers”)
  - ByteSub (nonlinear layer)
  - ShiftRow (linear mixing layer)
  - MixColumn (nonlinear layer)
  - AddRoundKey (key addition layer)

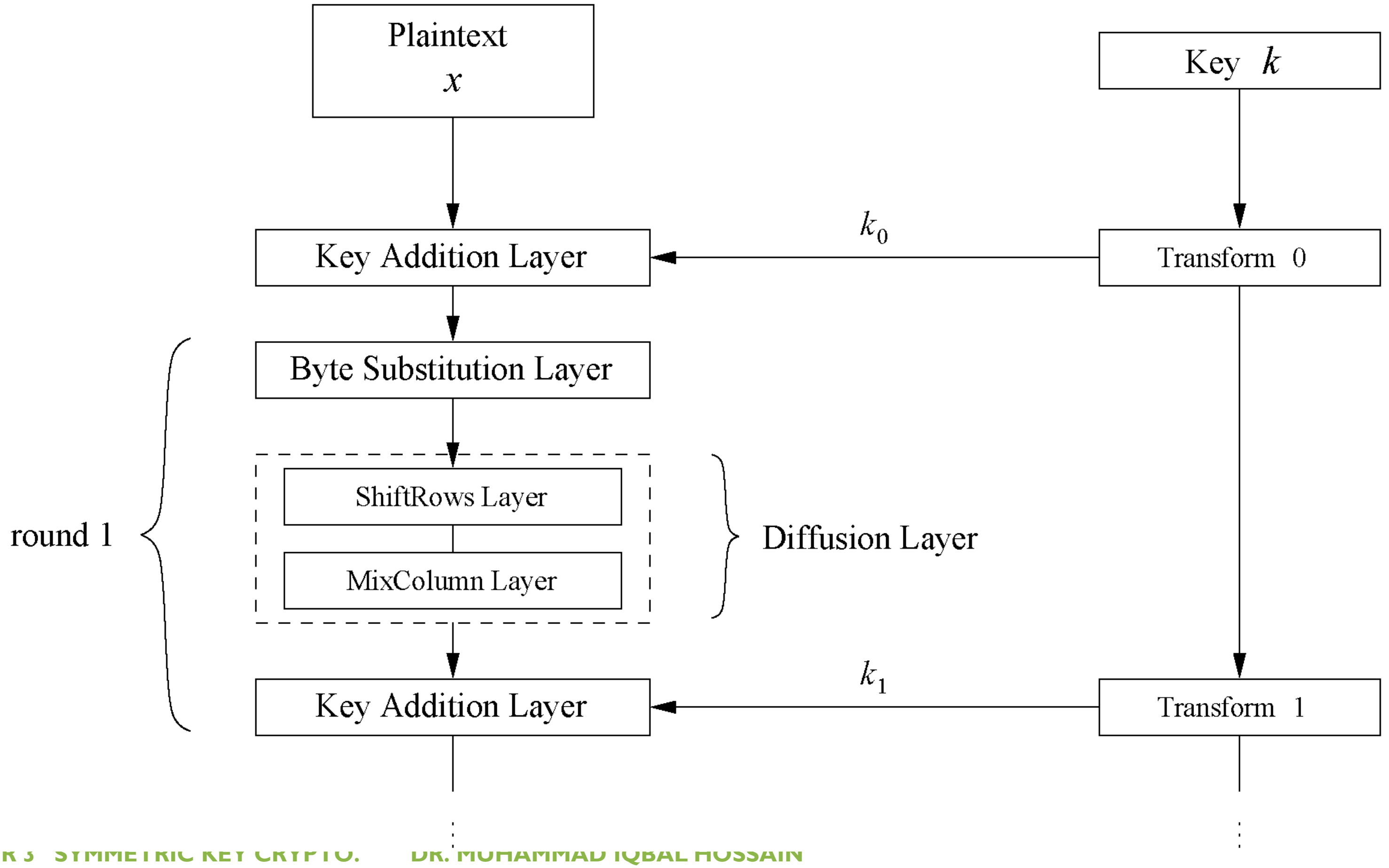
AES: OVERVIEW

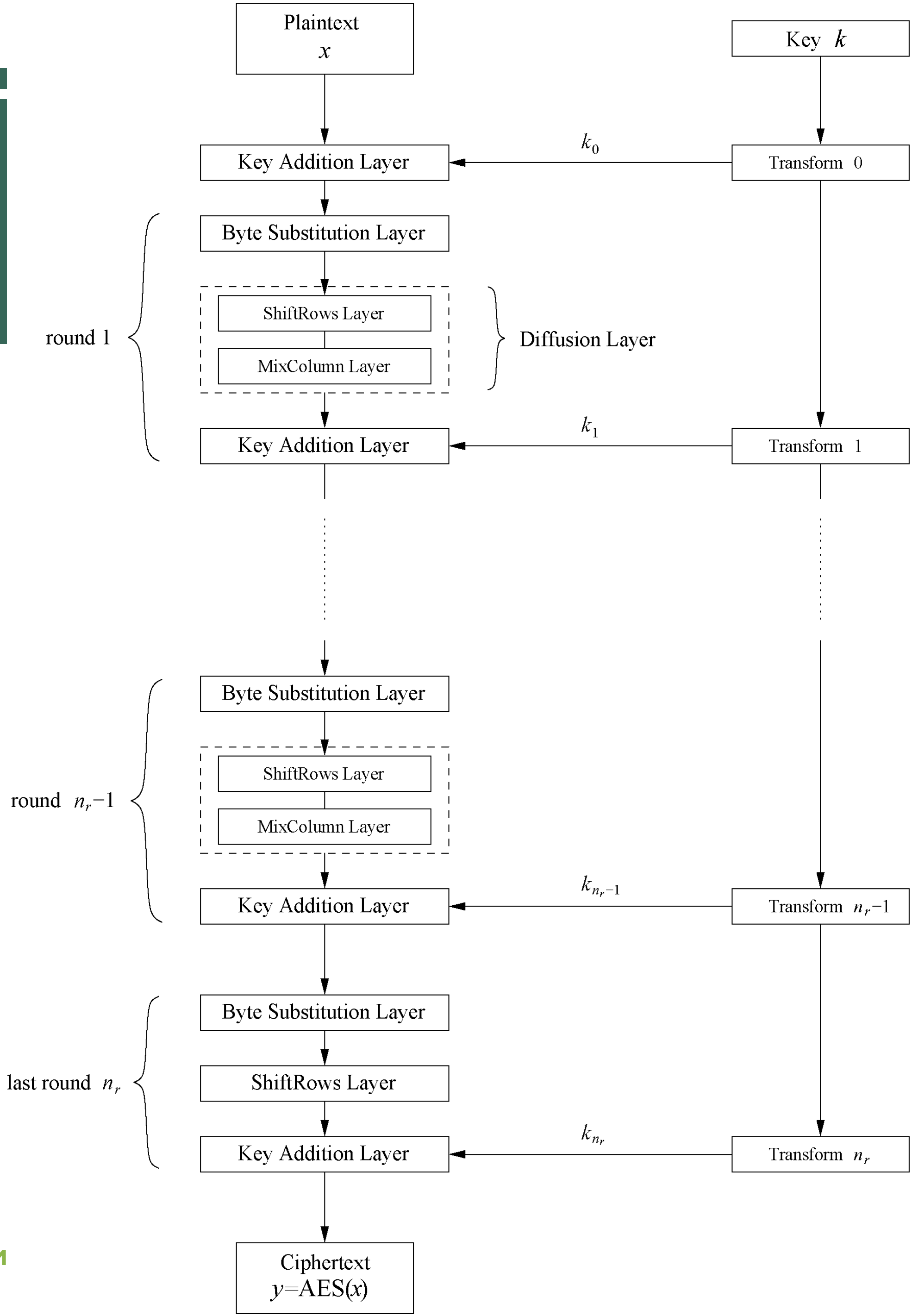


The number of rounds depends on the chosen key length:

Key length (bits)	Number of rounds
128	10
192	12
256	14

AES: OVERVIEW




Plaintext
x
Key k
Key Addition Layer
k_0
Transform 0
round 1
Byte Substitution Layer
ShiftRows Layer
MixColumn Layer
Diffusion Layer
Key Addition Layer
k_1
Transform 1
round n_r-1
Byte Substitution Layer
ShiftRows Layer
MixColumn Layer
Key Addition Layer
k_{n_r-1}
Transform n_r-1
last round n_r
Byte Substitution Layer
ShiftRows Layer
Key Addition Layer
k_{n_r}
Transform n_r
Ciphertext
y=AES(x)

INTERNAL STRUCTURE OF AES

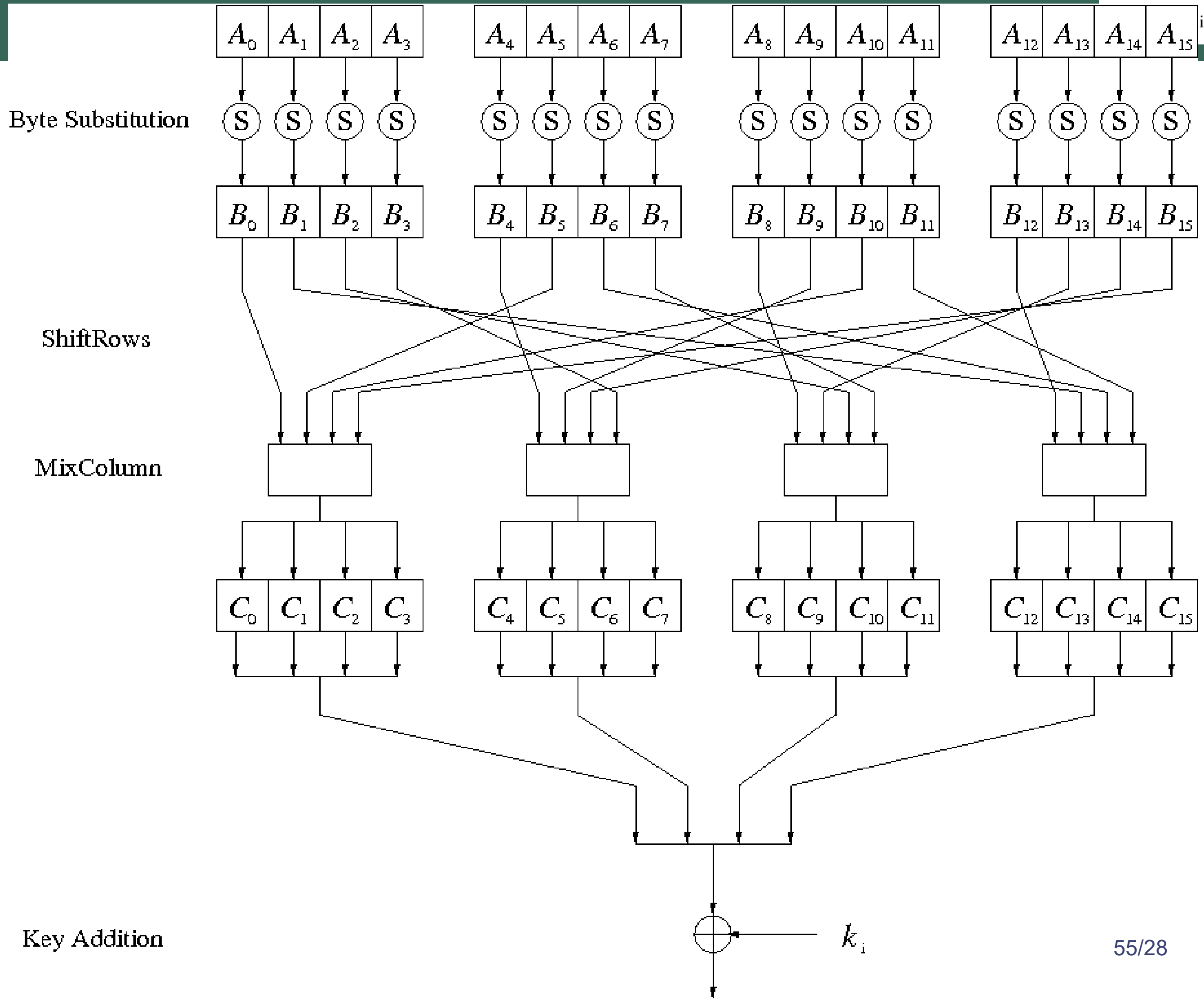
- AES is a byte-oriented cipher
- The state *A* (i.e., the 128-bit data path) can be arranged in a 4x4 matrix:

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

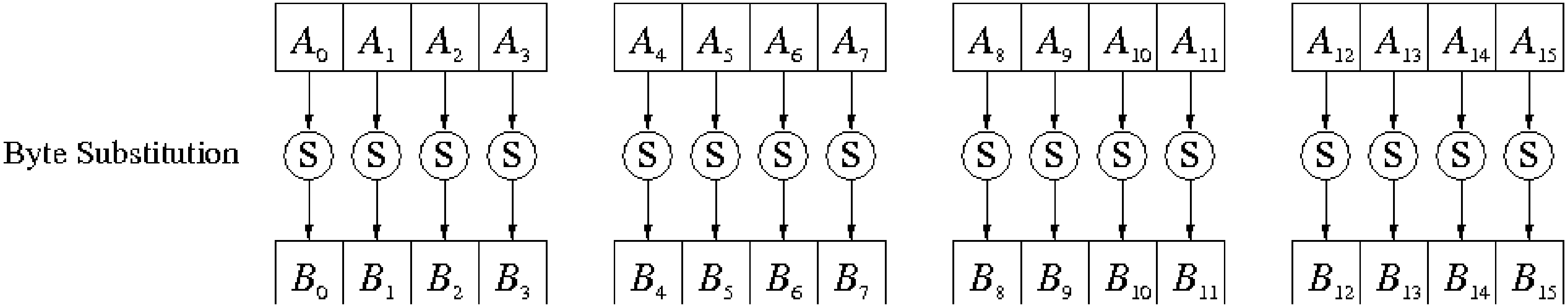
$A_0,\ldots,A_{15}$ Contain HEX number.
For example: $A_0=C2, A_2=EA$ ...

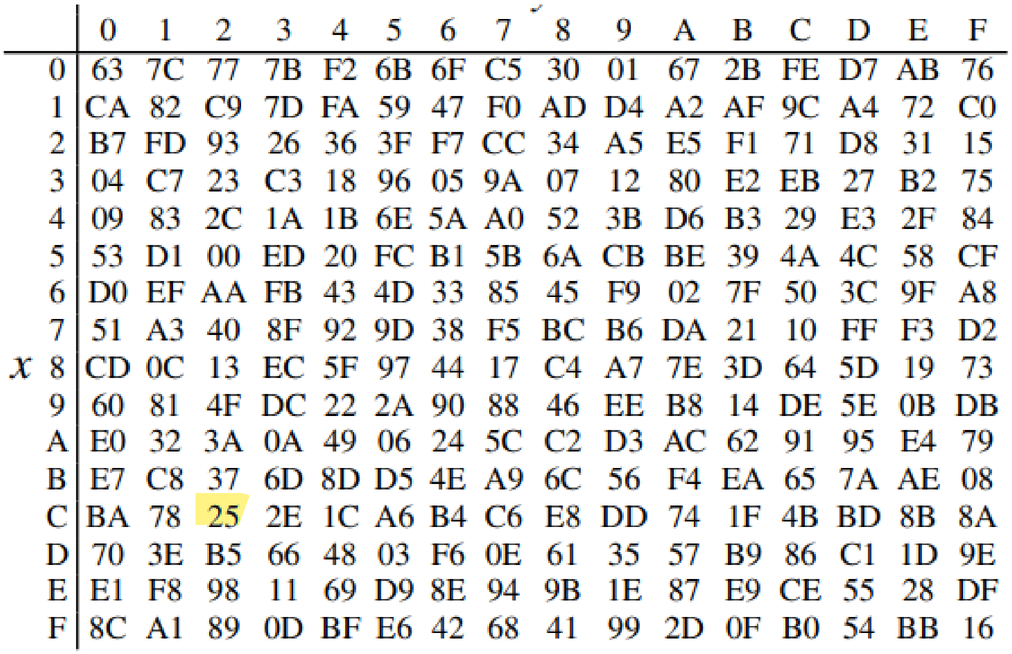
with $A_0,\ldots,A_{15}$ denoting the 16-byte input of AES

INTERNAL STRUCTURE OF AES


A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13} A_{14} A_{15}
Byte Substitution
S
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9 B_{10} B_{11} B_{12} B_{13} B_{14} B_{15}
ShiftRows
MixColumn
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10} C_{11} C_{12} C_{13} C_{14} C_{15}
Key Addition
k_i

BYTE SUBSTITUTION

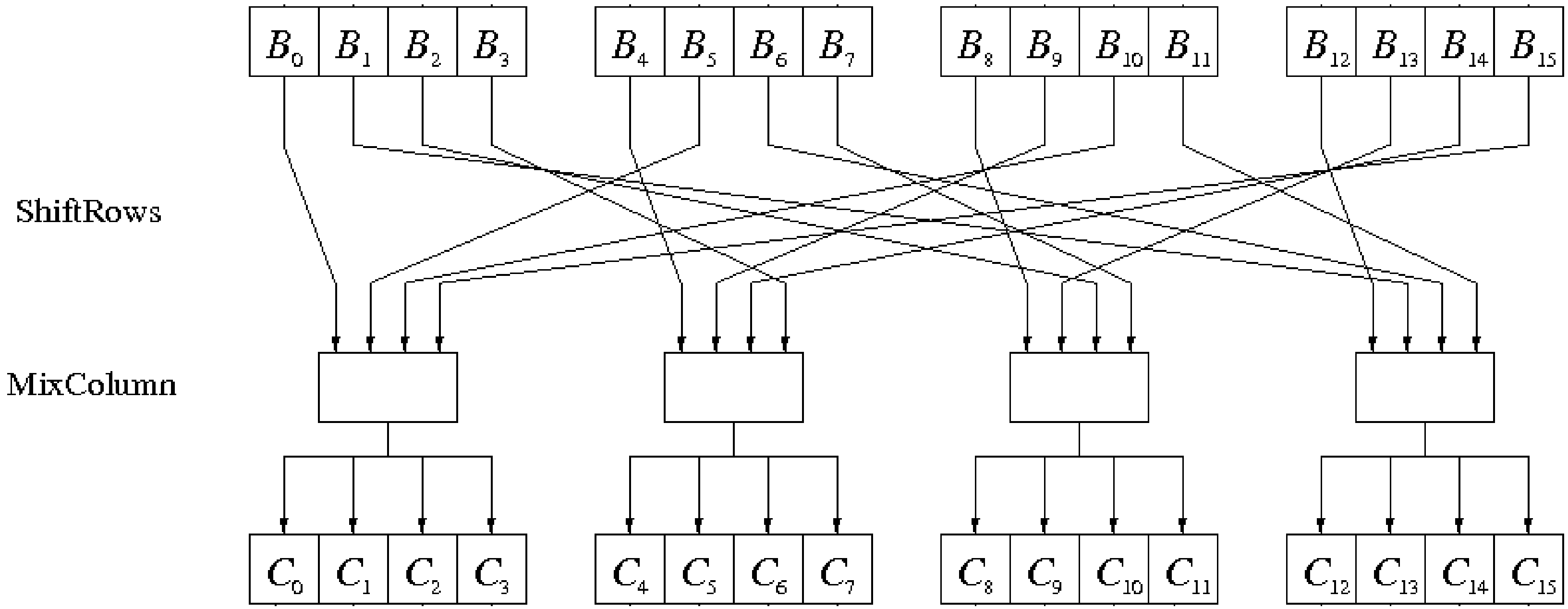

Byte Substitution
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} A_{11} A_{12} A_{13} A_{14} A_{15}
S
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9 B_{10} B_{11} B_{12} B_{13} B_{14} B_{15}



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x 8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Let’s assume the input byte to the S-Box is Ai = (C2)hex, then the substituted value is S((C2)hex) = (25)hex.

SHIFTROWS AND MIXCOLUMN


B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9 B_{10} B_{11} B_{12} B_{13} B_{14} B_{15}
ShiftRows
MixColumn
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10} C_{11} C_{12} C_{13} C_{14} C_{15}

SHIFTROWS SUBLAYER

- Rows of the state matrix are shifted cyclically:

Input matrix

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Output matrix

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	← one position left shift
B_{10}	B_{14}	B_2	B_6	← two positions left shift
B_{15}	B_3	B_7	B_{11}	← three positions left shift

MIXCOLUMN SUBLAYER

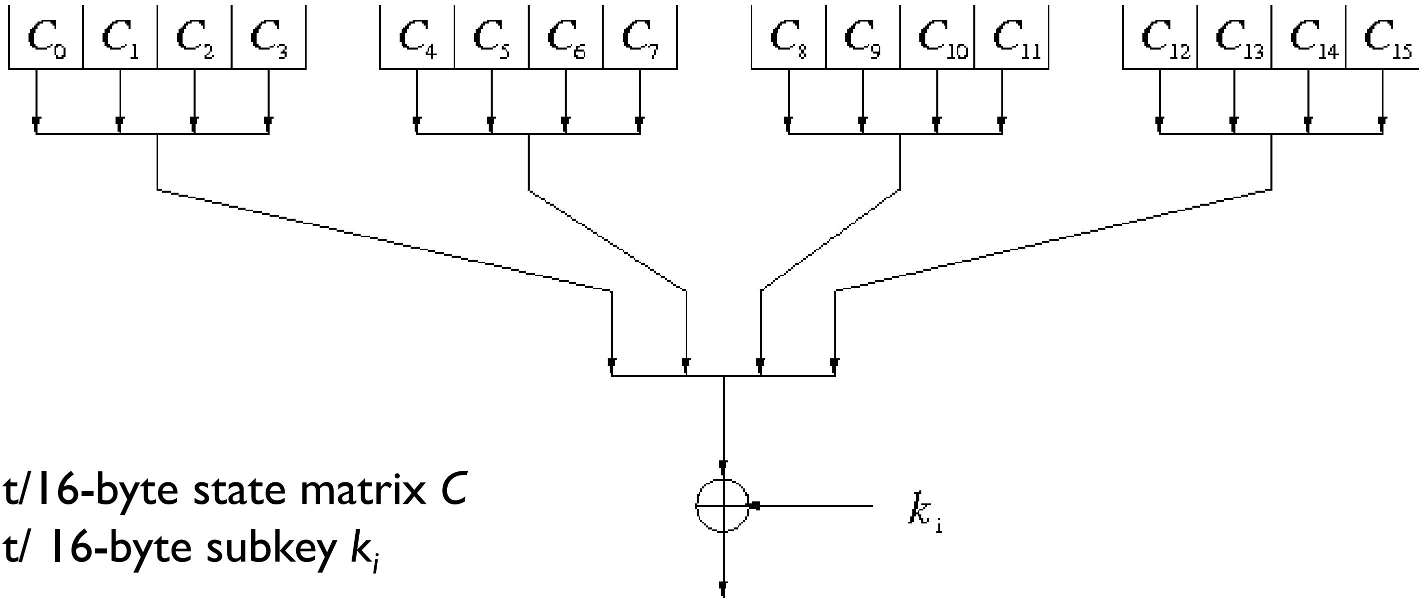
Each 4-byte column is considered as a vector and multiplied by a fixed 4x4 matrix, e.g.,

B_0	B_4	B_8	B_{12}
B_5	B_9	B_{13}	B_1
B_{10}	B_{14}	B_2	B_6
B_{15}	B_3	B_7	B_{11}

Output matrix

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

KEY ADDITION LAYER



Inputs:

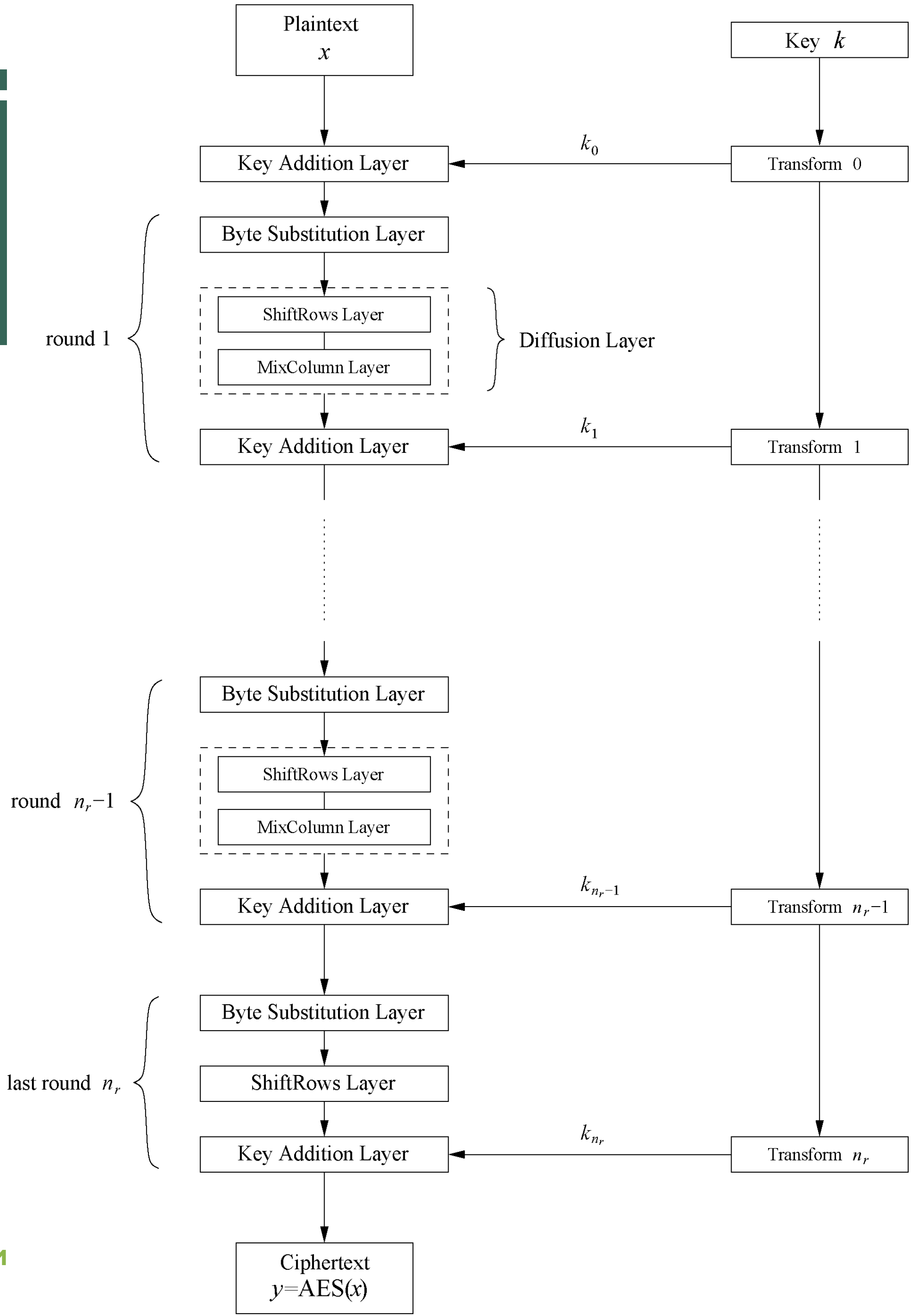
128bit/16-byte state matrix C

128bit/ 16-byte subkey k_i

Output: $C \oplus k_i$

The subkeys are generated in the key schedule

Key Schedule



Plaintext x

Key k

Key Addition Layer

k_0

Transform 0

round 1

Byte Substitution Layer

ShiftRows Layer

MixColumn Layer

Diffusion Layer

Key Addition Layer

k_1

Transform 1

round n_r-1

Byte Substitution Layer

ShiftRows Layer

MixColumn Layer

Key Addition Layer

k_{n_r-1}

Transform n_r-1

last round n_r

Byte Substitution Layer

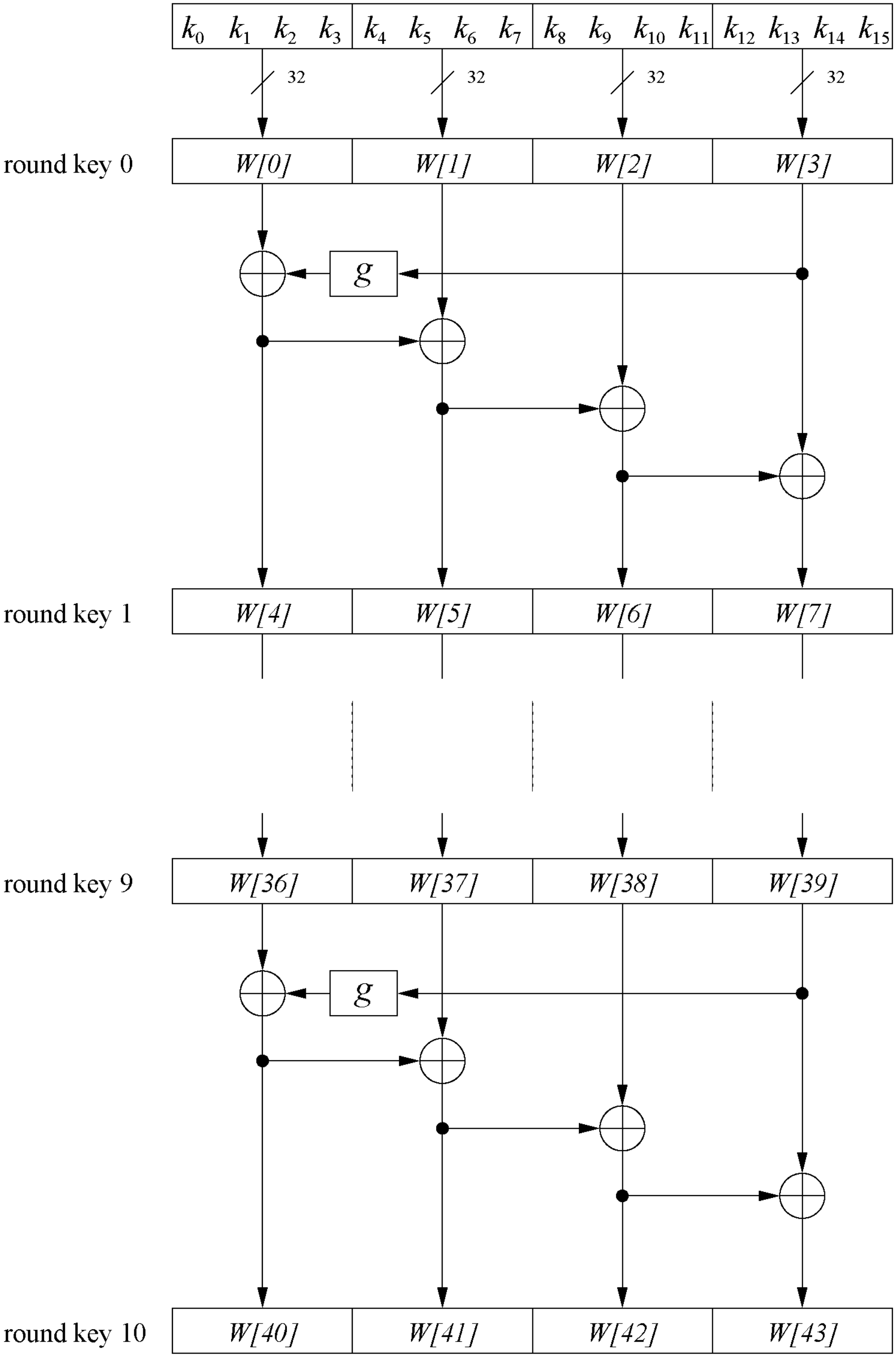
ShiftRows Layer

Key Addition Layer

k_{n_r}

Transform n_r

Ciphertext y=AES(x)

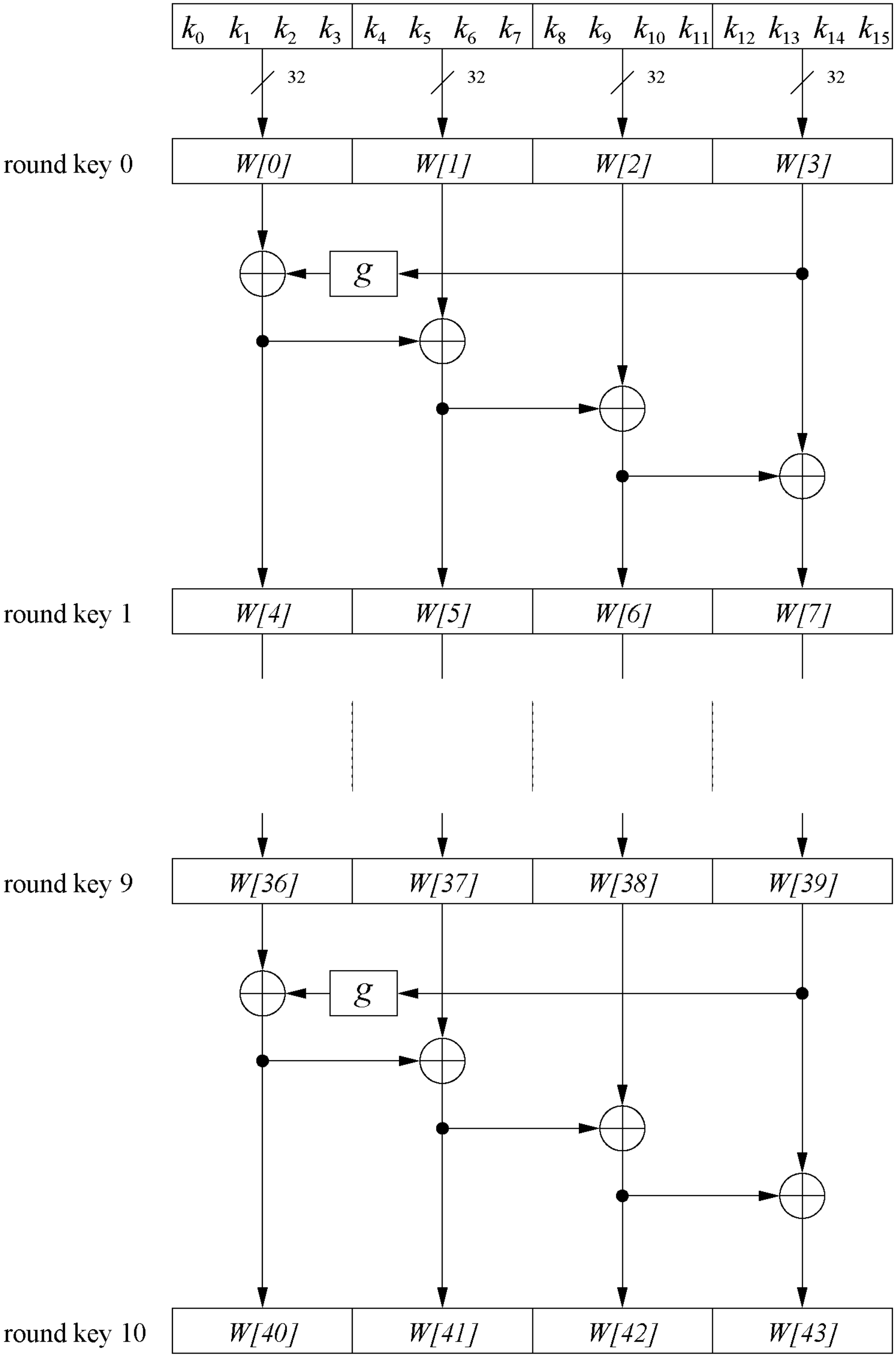


- Word-oriented: 1 word = 32 bits

- 11 subkeys are stored in *W[0]…W[3]*, *W[4]…W[7]*, … , *W[40]…W[43]*

- First subkey *W[0]…W[3]* is the original AES key

EXAMPLE: KEY SCHEDULE FOR 128-BIT KEY AES



- **G-Function**

- **RotWord** is quite simple. It takes a 4-byte word $[a_0, a_1, a_2, a_3]$ and returns $[a_1, a_2, a_3, a_0]$
- **SubWord** is a little bit more complex. It takes a 4-byte word $[a_1, a_2, a_3, a_0]$ and applies the AES S-Box to each of the bytes to produce a new 4-byte word $[b_0, b_1, b_2, b_3]$.
- The result of steps 1 and 2 is XORed with a round constant, **Rcon**[j].

This question is about the Key Schedule of AES algorithm.
Let's assume the initial key is: **(W[0] W[1] W[2] W[3])** = (**4A C6 9E 45)** and the Round constant is **00 01 10 11.**
You need to use the S-box of AES.

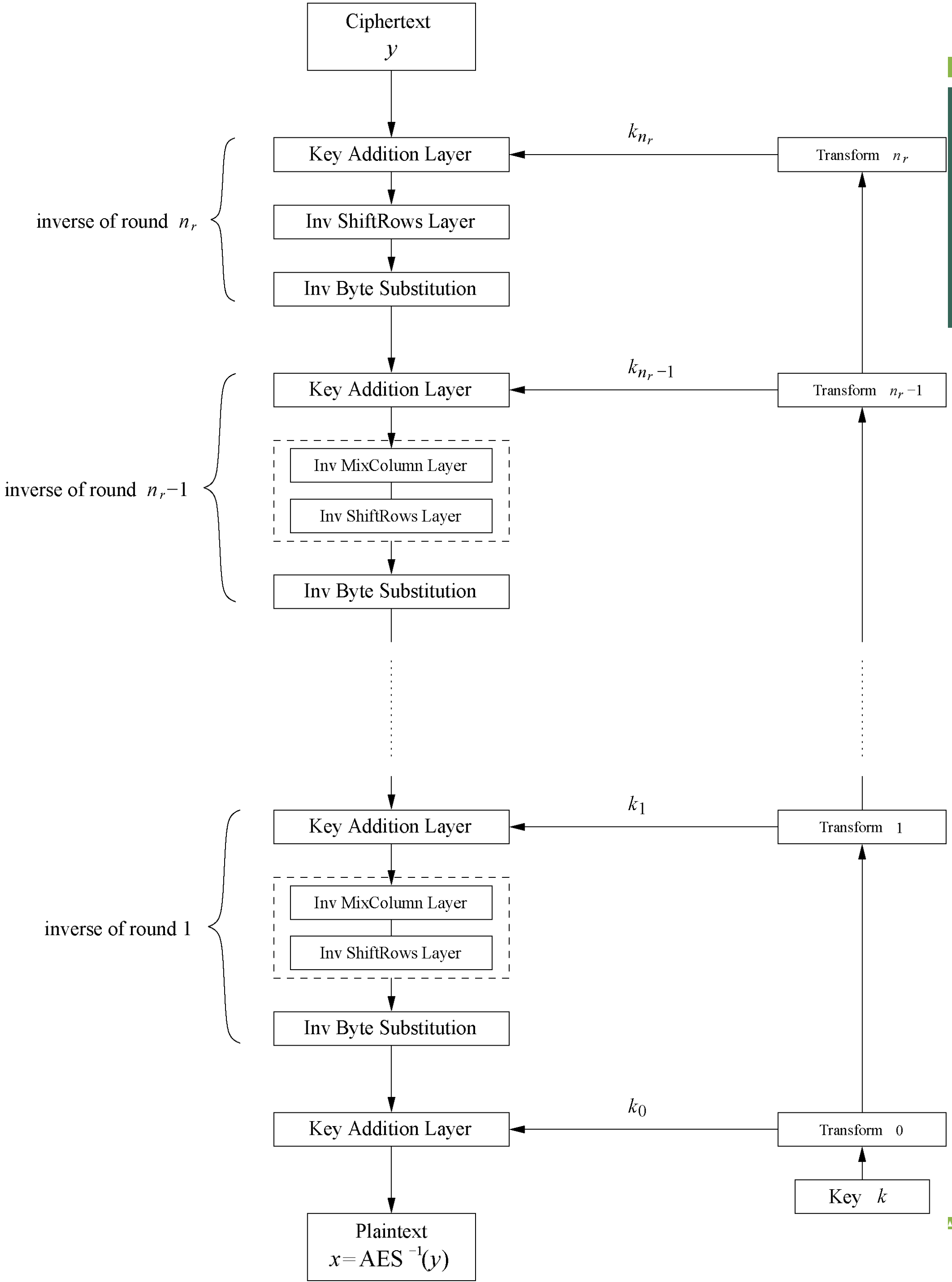
W[4] = W[0] ⊕ g(W[3])
W[5] = W[1] ⊕ W[4]
W[6] = W[2] ⊕ W[5]
W[7] = W[3] ⊕ W[6]

W[4] = 4A ⊕ g(W[3]) = 01001010 ⊕ 00111011
=01110001 = 71
W[5] = C6 ⊕ 71 = 11000110 ⊕ 01110001
=10110111 = B7
And so on…

g(W[3]) = g(45)
Step 1:
Rotate W[3] i.e 45 becomes 54
Step 2:
S-box(54) = 20
Step 3:
20 ⊕ RC
= 00100000 ⊕ 00011011
=**00111011** = g(W[3])

AES DECRYPTION

- To decrypt, process must be invertible
- Inverse of MixAddRoundKey is easy, since “⊕” is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)



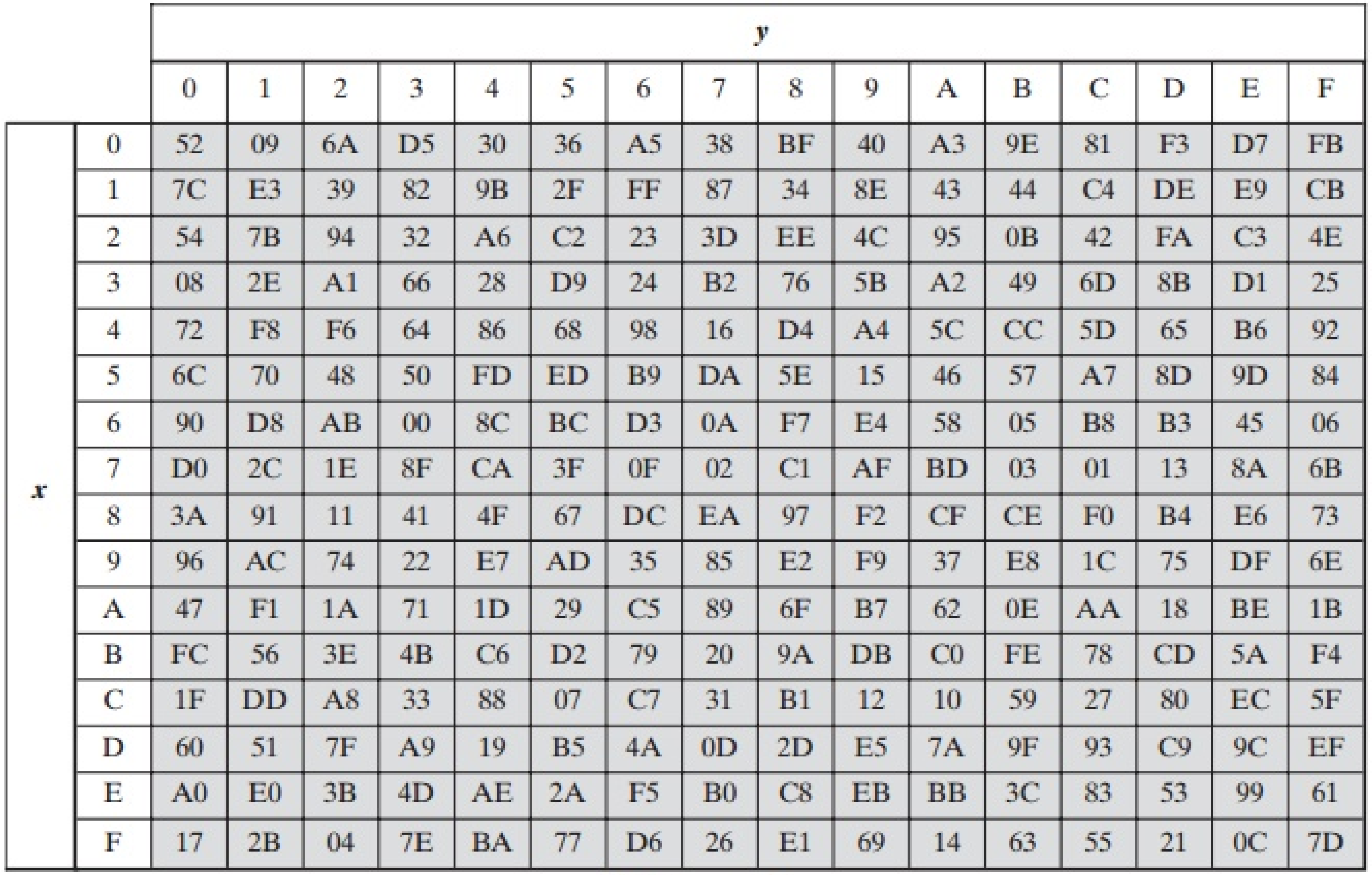
AES DECRYPTION

BRAC UNIVERSITY
Inspiring Excellence

- AES is not based on a Feistel network

⇒ All layers must be inverted for decryption:

- MixColumn layer → **Inv MixColumn layer**
- ShiftRows layer→ **Inv ShiftRows layer**
- Byte Substitution layer → **Inv Byte Substitution layer**
- Key Addition layer is its own inverse



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

A FEW OTHER BLOCK CIPHERS

- Briefly…
  - IDEA
  - Blowfish
  - RC6
- More detailed…
  - TEA

End of segment

MODES OF OPERATION

- Many modes — we discuss 3 most popular
- Electronic Codebook (**ECB**) mode
  - Encrypt each block independently
  - Most obvious approach, but a bad idea
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
  - More secure than ECB, virtually no extra work
- Counter Mode (**CTR**) mode
  - Block ciphers acts like a stream cipher
  - Popular for random access

DATA INTEGRITY

- **Integrity** — detect unauthorized writing (i.e., detect unauthorized mod of data)
- Example: Inter-bank fund transfers
  - Confidentiality may be nice, integrity is ***critical***
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc., etc.

MAC

- Message Authentication Code (MAC)
  - Used for data **integrity**
  - Integrity **not** the same as confidentiality
- MAC is computed as **CBC residue**
  - That is, compute CBC encryption, saving only final ciphertext block, the MAC
  - The MAC serves as a cryptographic checksum for data

MAC COMPUTATION

- MAC computation (assuming N blocks)

$$C_0 = E(IV \oplus P_0, K),$$

$$C_1 = E(C_0 \oplus P_1, K),$$

$$C_2 = E(C_1 \oplus P_2, K),\ldots$$

$$C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = MAC$$

- Send IV, P_0, P_1, …, P_{N-1} and MAC
- Receiver does same computation and verifies that result agrees with MAC
- Both sender and receiver must know K

DOES A MAC WORK?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = \text{MAC}$
- Alice sends IV, P_0, P_1, P_2, P_3 and MAC to Bob
- Suppose Trudy changes P_1 to X
- Bob computes

  $C_0 = E(IV \oplus P_0, K)$, $\mathit{C_1} = E(C_0 \oplus X, K)$,

  $\mathit{C_2} = E(\mathit{C_1} \oplus P_2, K)$, $\mathit{C_3} = E(\mathit{C_2} \oplus P_3, K) = \mathit{MAC} \neq \text{MAC}$
- It works since error propagates into MAC
- Trudy can't make *MAC* == MAC without K

CONFIDENTIALITY AND INTEGRITY

- Encrypt with one key, MAC with another key
- Why not use the same key?
  - Send last encrypted block (MAC) twice?
  - This cannot add any security!
- Using different keys to encrypt and compute MAC works, even if keys are related
  - But, twice as much work as encryption alone
  - Can do a little better — about 1.5 "encryptions"
- Confidentiality *and* integrity with same work as one encryption is a research topic

USES FOR SYMMETRIC CRYPTO

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later…)
- Anything you can do with a hash function (upcoming chapter…)