บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

1. วิทยาการรหัสลับ (Cryptography)

วิทยาการรหัสลับ [1] เป็นการศึกษาเทคนิคเชิงคณิตศาสตร์ เพื่อนำมาเกี่ยวข้องกับความปลอดภัยของข้อมูล (Information Security) โดยวัตถุประสงค์หลัก เพื่อการรักษาความลับ (Confidentiality) การรักษาความสมบูรณ์ของข้อมูล (Data Integrity) การยืนยันตัวตน (Authentication) และการห้ามปฏิเสธความรับผิดชอบ (Non-Repudiation) โดยมีรายละเอียดดังต่อไปนี้

(1) Confidentiality เป็น การรับรองของข้อมูลที่จัดเก็บว่าเป็นความลับ (Secrecy) และจำกัดสิทธิเข้าใช้งานข้อมูล

(2) Data integrity เป็น การรับรองของข้อมูลว่ามีความถูกต้องสมบูรณ์มิได้ถูกทำการเปลี่ยนแปลงแก้ไขใดๆ หรือ ถูกทำลาย

(3) Authentication เป็น การรับรองของข้อมูลว่าข้อมูลที่ถูกส่งมานั้นมาจากผู้ส่งจริงและต้นฉบับจริง

(4) Non-Repudiation เป็น การรับรองว่ามีการส่งข้อมูลหาผู้รับจริง และผู้รับมิอาจปฏิเสธความเกี่ยวข้องของข้อมูลที่ได้รับมา

วิทยาการรหัสลับ มีกระบวนการที่ใช้ซ่อน ข้อความที่ต้องการส่ง (Plain text) ซึ่งถูกเรียกว่า การเข้ารหัส (Encryption) ซึ่งจะได้ ข้อความที่ถูกเข้ารหัส (Cipher text) และ เมื่อผู้รับต้องการอ่านข้อความที่ถูกเข้ารหัสนั้นต้องใช้กระบวน การถอดรหัส (Decryption) ซึ่งทั้งสองกระบวนการนั้นต้องใช้ กุญแจ (Key) ในการทำงาน โดยทำการกำหนดให้รูปแบบการสนทนาของบุคคล คือ Alice เป็นผู้ส่งสาร และ Bob เป็นผู้รับสาร และระหว่างการสนทนานั้นอาจมีผู้ดักฟังการสนทนา ในที่นี้คือ Eve ในขั้นตอนการเข้ารหัส ต้องใช้กุญแจในการเข้ารหัส (Encryption Key) และ ขั้นตอนการถอดรหัส ต้องใช้กุญแจในการถอดรหัส (Decryption Key) ซึ่งสามารถแบ่งรูปแบบการใช้กุญแจในการทำงานอยู่ 2 แบบ คือ

1.1 การเข้ารหัสโดยใช้กุญแจแบบอสมมาตร (Asymmetrical Encryption)

การเข้ารหัสแบบอสมมาตร เป็นการเข้ารหัสข้อความโดยการใช้กุญแจในการเข้ารหัสคนละดอกหรือจะเรียกโดยทั่วไปว่า “Public Key” โดยผู้ส่งจะทำการเข้ารหัสข้อความที่ส่งโดยใช้ Public Key ของผู้รับ เมื่อผู้รับได้รับข้อความที่ถูกเข้ารหัสมาแล้วนั้น ผู้รับจะใช้กุญแจอีกดอกในการถอดรหัส ซึ่งเรียกว่า “Private Key” ซึ่งกุญแจดอกนี้นั้น จะมีเฉพาะเพียงผู้รับเท่านั้นที่จะสามารถถอดรหัสได้ ยกตัวอย่างเช่น RSA , Digital signatures เป็นต้น

1.2 การเข้ารหัสโดยใช้กุญแจแบบสมมาตร (Symmetrical Encryption)

การเข้ารหัสโดยใช้กุญแจแบบสมมาตร [1] เป็นการเข้ารหัสและถอดรหัสโดยการใช้กุญแจดอกเดียวกัน (Secret-key) โดยมีตัวอย่างในการเข้ารหัสโดยใช้วิธีนี้ คือ One time pad เป็นรูปแบบการเข้ารหัส ที่ถูกพัฒนาขึ้นโดย Gilbert Vernam และ Joseph Mauborgne เมื่อปี 1918 การสุ่มกุญแจโดยใช้ เลข 0 และ 1 ซึ่งความยาวของกุญแจที่ใช้ คือ มีขนาดตามความยาวของข้อความที่ส่ง โดยที่กุญแจที่ใช้นี้จะนำมาใช้เพียงแค่ครั้งเดียวและจะไม่นำกลับมาใช้ใหม่อีกครั้ง การเข้ารหัสทำโดยการเพิ่ม key เข้าไปที่ข้อความ เลือก mod 2 , bit by bit อาจจะเรียกกระบวนการนี้ว่า exclusive or หรือสัญลักษณ์คือ XOR มีกฎ ดังนี้คือ 0+0 = 0, 0+1 = 1, 1+1=0 สำหรับการถอดรหัสข้อความเพื่ออ่านนั้นจำเป็นที่จะต้องใช้กุญแจดอกเดียวกัน(Symmetric key) 10000101 + 10101100 = 00101001

2. อัลกอริทึมที่ใช้ในการดำเนินงาน (Implemented Algorithms)

อัลกอริทึมที่เลือกมาใช้ในการดำเนินงานดำเนินงานครั้งนี้คือ Data Encryption Standard และ Advanced Encryption Standard

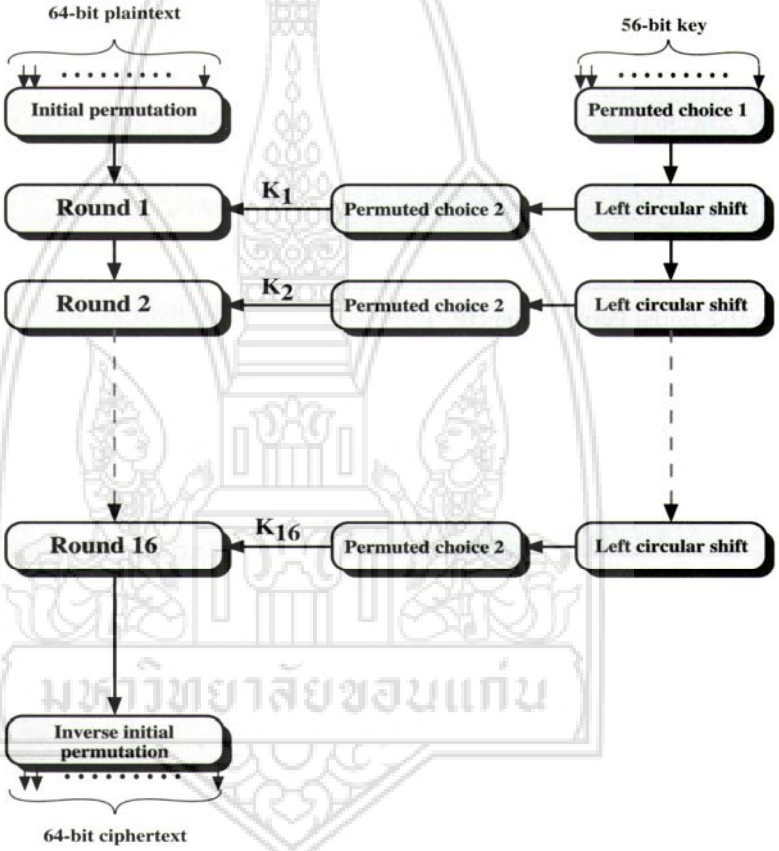
2.1 Data Encryption Standard(DES)

เป็นอัลกอริทึมที่ใช้ในการเข้าและถอดรหัสแบบ Block Cipher [2] ซึ่งจะทำการแบ่งข้อมูลออกเป็น block แล้วนำไปทำการเข้ารหัส และเป็นอัลกอริทึมกุญแจแบบสมมาตร คือใช้กุญแจในการเข้าและถอดรหัสดอกเดียวกัน (Secret key encryption)

เมื่อปี 1973 รัฐบาลสหรัฐภายใต้ชื่อ NBS (National Bureau of Standards) เริ่มมีการประกาศถึงความต้องการในการสร้างมาตรฐานในการเข้ารหัส ซึ่งปัจจุบันองค์กรนี้เป็นรู้จักกันอย่างแพร่หลายภายใต้ชื่อ NIST (National Institute of Standards and Technology) ในปีต่อมาจึงได้มีการพัฒนาและค้นหาอัลกอริทึมเพื่อนำมาใช้เป็นมาตรฐาน บริษัท IBM ได้ทำการพัฒนา Lucifer ให้มาเป็น DES และถูกตีพิมพ์ครั้งแรกในปี 1975 โดยประกาศใช้งานเมื่อปี 1977 โดยใช้ชื่อ

รหัสว่า FIPS PUB 64 (Federal Information Processing Standard 46) และในปี 1994 ได้ปรับปรุงมาตรฐานเป็น FIPS PUB 46-2 โดยอัลกอริทึมที่ใช้รู้จักกันในชื่อของ DEA (Data Encryption Algorithm) จนกระทั่งเมื่อปี 2001 อัลกอริทึม DES ได้ถูแทนที่ด้วยอัลกอริทึมแบบ AES (Advanced Encryption Standard)

หลักการทำงานพื้นฐานการทำงานของอัลกอริทึม DES แยกเป็นส่วน ดังนี้ ทำการนำเข้าชุดข้อมูลแบบบล็อกขนาด 64 บิต และใช้กุญแจขนาด 56 บิต มีจำนวนรอบในการทำงานเท่ากับ 16 รอบ เพื่อสร้างกุญแจย่อยให้มีจำนวน 16 ดอก โดยระหว่างการเข้ารหัสนั้นแต่ละรอบการทำงานจะมีกุญแจขนาด 48 บิต การทำงานดังภาพที่ 1



ภาพที่ 1 แสดงการทำงานของอัลกอริทึม DES โดย การนำเข้าข้อมูลขนาด 64 บิต และกุญแจขนาด 54 บิต [7]

ขั้นตอนการสร้างกุญแจ

ยกตัวอย่าง การนำเข้าข้อความขนาด 64 บิต

ข้อความตัวอักษร : abcdefgh

เลขฐานสองที่ได้ : 01100001 01100010 01100011 01100100 01100101 01100110 01100111 01101000

เมื่อทำการแปลงข้อมูลข้อความเป็นตัวเลขฐานสองแล้วจึงนำเลขเหล่านั้นมาทำการสลับบิตโดยใช้ค่าในตาราง IP

ตารางที่ 1 Initial Permutation (IP)

Initial Permutation (IP)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

หมายความว่าค่าบิตที่ 1 ของผลลัพธ์ ให้นำค่าบิตที่ 58 ของฐานสองที่นำเข้ามาวางที่ตำแหน่งนี้และค่าบิตผลลัพธ์ถัดมาให้นำค่าบิตที่ 50 ของฐานสองที่นำเข้ามาวางที่ตำแหน่งนี้ผลลัพธ์ที่ได้เมื่อผ่านตาราง IP คือ

11111111000000000111100001010101 00000000111111111000000001100110

กำหนดให้ 32 บิตแรกคือ L และ 32 บิตหลังคือ R จากนั้นทำการเลือกกุญแจขนาด 56 บิต โดยการนำข้อความขนาด 64 บิต มาสลับตำแหน่งบิตตามตาราง PC-1

ตารางที่ 2 Permuted Choice 1 (PC-1)

Permuted Choice 1 (PC-1)						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

หมายความว่าค่าบิตที่ 1 ของผลลัพธ์ ให้นำค่าบิตที่ 57 ของฐานสองที่นำเข้ามาวางที่ตำแหน่งนี้ และค่าบิตผลลัพธ์ถัดมาให้นำค่าบิตที่ 49 ของของฐานสองที่นำเข้ามาวางที่ตำแหน่งนี้ ผลลัพธ์ที่ได้เมื่อผ่านตาราง PC-1 คือ

00000000111111111111111110000 01100110011110001000000000

กำหนดให้ 28 บิต แรก แทนด้วย C( ) และ 28 บิตหลังคือ D( )

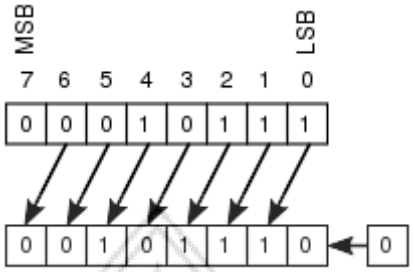
จากนั้นทำการเลื่อนบิตไปทางซ้าย (Left Shifts) ทีละ 28 บิต แรก และ ทีละ 28 บิต หลัง ตามตารางจำนวนรอบของการสร้างกุญแจดังตารางที่ 3

ตารางที่ 3 Left Shifts

รอบที่	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Left Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

เช่น การคำนวณกุญแจรอบที่ 1 ทำการเลื่อนบิตไปทางซ้าย หนึ่งครั้งดังภาพที่ 2

C( ) = 00000000111111111111111110000



ภาพที่ 2 Left Shifts

Left Shift = 00000001111111111111111100000

เมื่อทำการเลื่อนบิตแล้วนำบิตที่ได้มาทำการสลับที่บิตโดยใช้ค่าตำแหน่งตามตารางที่ 4

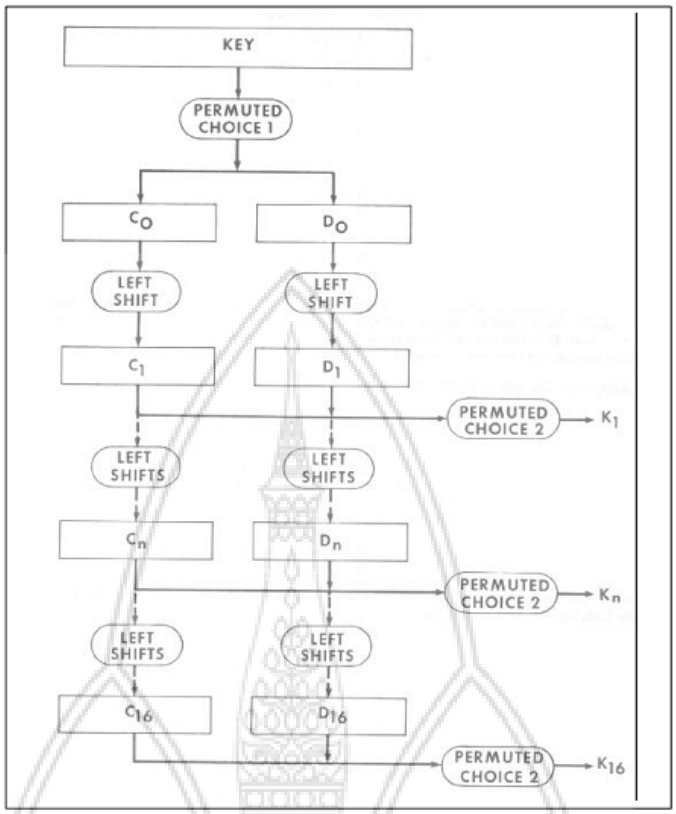
ตารางที่ 4 Permuted Choice 2 (PC-2)

Permuted Choice 2 (PC-2)					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

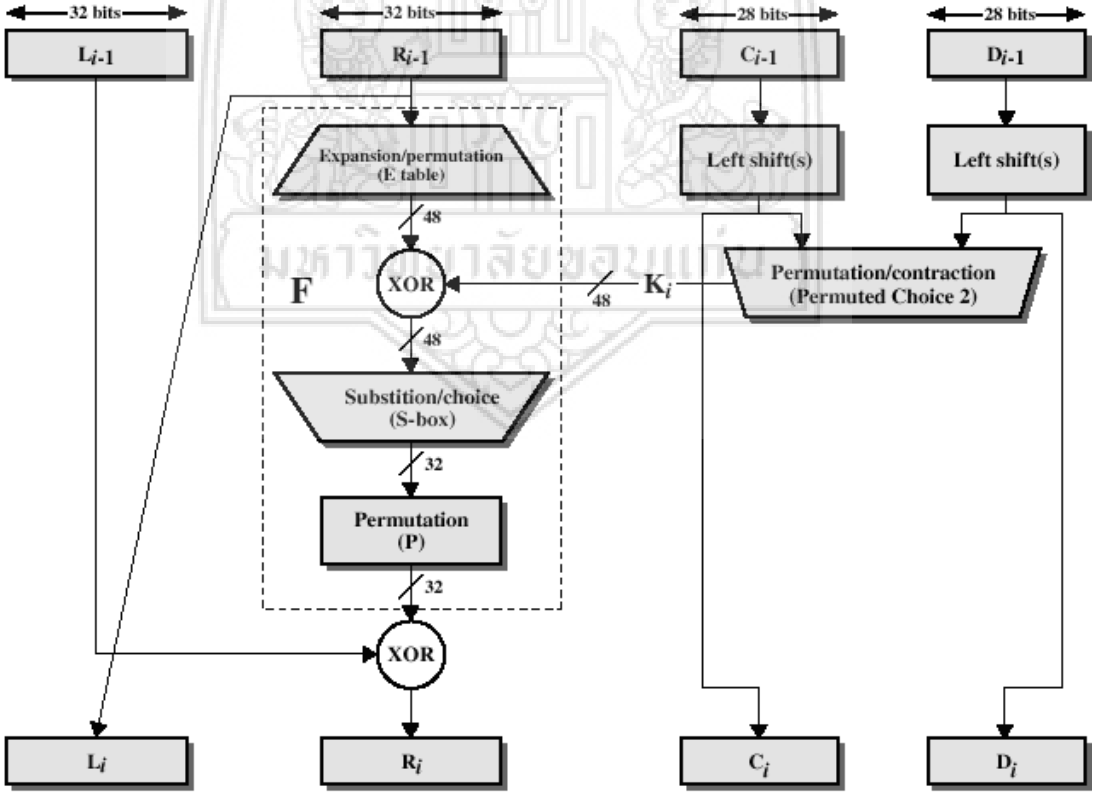
เมื่อทำการสลับบิตดังตารางทั้งจะได้ค่าของกุญแจดอกแรกออกมาดังนี้

K1 = 111000001011111001100110000100110010101010000010

จากการคำนวณกุญแจขนาด 48 บิต ที่กล่าวมาข้างต้นสามารถสรุปได้ดังภาพที่ 3



ภาพที่ 3 สรุปขั้นตอนการสร้างกุญแจย่อยจำนวน 16 ดอก [2]



ภาพที่ 4 การคำนวณรอบที่ของการทำงานของ DES algorithm [8]

เมื่อทำการสร้างกุญแจย่อยจำนวน 1 ดอก สำเร็จ ขั้นตอนต่อไปจะต้องนำกุญแจนั้นมา XOR กับ 32 บิต R ที่กำหนดดังภาพที่ 4 ซึ่งเป็นภาพรวมของการเข้ารหัสแบบ DES สำหรับรายละเอียดของการเข้ารหัสในแต่ละรอบนั้น ได้แสดงไว้ในภาพที่ 4

นำข้อมูลขนาด 32 บิต มาทำการขยายบิตเพิ่ม โดยใช้ตาราง E Bit- Selection โดยผลลัพธ์ที่ได้คือ บิตที่มีขนาด 48 บิต จากนั้นจะนำผลลัพธ์ที่ขยายเป็น 48 บิตไป XOR กับกุญแจย่อย

ตารางที่ 5 Expansion (E)

Expansion (E)					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

จากค่า R = 00000000111111111000000001100110

เมื่อทำการผ่านค่า R ขนาด 32 บิต ในตารางที่ 5 จะได้ค่าบิตที่สลับ ขยายเป็น 48 บิต ดังนี้

R0 = 000000000001011111111111100000000000001100001100

จากนั้นนำ R0 มาทำการ XOR กับ K1 ที่ได้

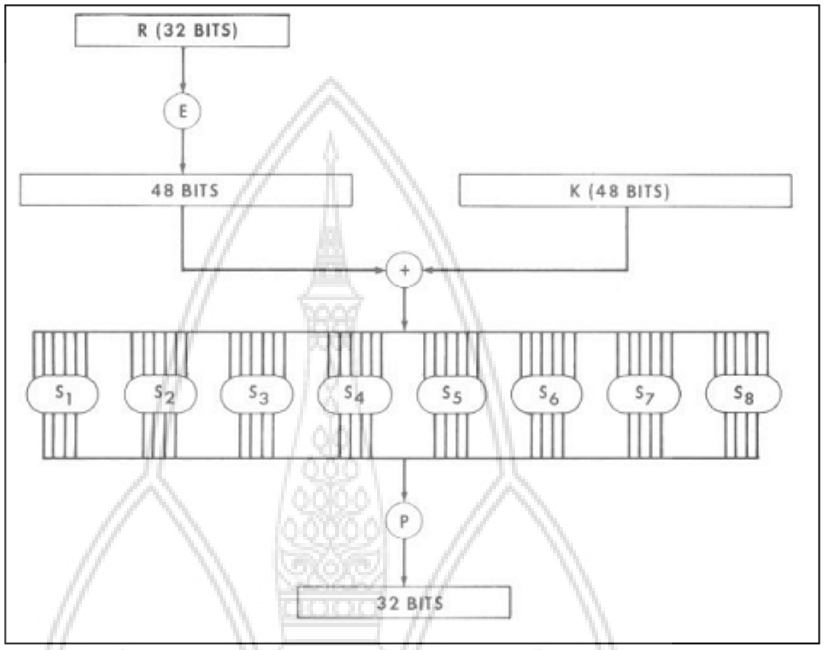
เมื่อกำหนดให้ ค่าบิตที่เหมือนกัน XOR ได้ 0 ส่วนค่าบิตที่ต่างกัน XOR กันได้ 1

R0 = 000000000001011111111111100000000000001100001100 XOR

K1 = 111000001011111001100110000100110010101010000010

F(R0,K1) = 111000001010100110011001110100110010100110001110

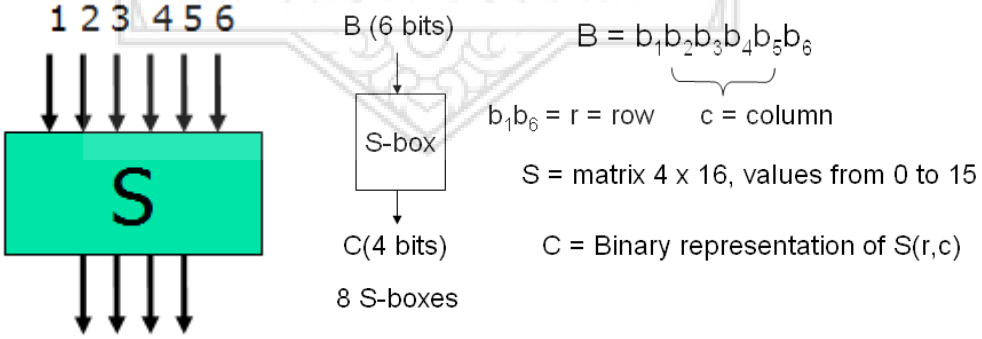
เมื่อได้ค่าออกมาแล้วจึงทำการลดขนาดบิตให้เหลือเพียง 32 บิตโดยการผ่านกระบวนการ S-Box



ภาพที่ 5 การลดขนาดบิตโดยผ่าน S-BOX ถูกลดความยาวลงเหลือ 32 บิต [2]

<u>ตัวอย่างการคำนวณ S box</u>

โดยการสลับ และแปลงข้อมูล 6 บิตเหลือ 4 บิต Bit 1,6 เลือก Row และ Bit 2,3,4,5 เลือก Column



ภาพที่ 6 แปลงข้อมูล 6 บิตเหลือ 4 บิต

S_1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

ภาพที่ 7 ตัวอย่าง การอ้างอิงตัวเลขจากตาราง s box ในการลดขนาดบิต [2]

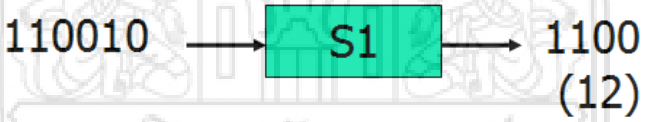
ข้อมูล 110010 เข้า S1

กำหนดให้

10 = Row เลขฐานสิบ คือ 2

1001 = Column เลขฐานสิบ คือ 9

เปิดตาราง S1 ค่าที่ได้คือ 12



ภาพที่ 8 แปลงข้อมูล 6 บิตเหลือ 4 บิต โดยผ่าน ตาราง s1

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

ภาพที่ 9 ค่าอ้างอิงตาราง S box ทั้ง 8 ตาราง ของการลดขนาดบิต [2]

ตารางที่ 6 ตัวอย่างการ XOR ของ f(R,K) ขนาด 48 บิต

ค่าบิตที่	**R0**	**K1**	**F(R0,K1)**
	48 บิต	**48 บิต**	**XOR**
1	0	1	1
2	0	1	1
3	0	1	1
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	1	1
10	0	0	0
11	0	1	1
12	1	1	0
13	0	1	1
14	1	1	0
15	1	1	0
16	1	0	1
17	1	0	1
18	1	1	0
19	1	1	0
20	1	0	1
21	1	0	1
22	1	1	0
23	1	1	0
24	1	0	1

ตารางที่ 6 ตัวอย่างการ XOR ของ f(R,K) ขนาด 48 บิต (ต่อ)

ค่าบิตที่	R0	K1	F(R0,K1)
	48 บิต	48 บิต	XOR
25	1	0	1
26	1	0	1
27	0	0	0
28	0	1	1
29	0	0	0
30	0	0	0
31	0	1	1
32	0	1	1
33	0	0	0
34	0	0	0
35	0	1	1
36	0	0	0
37	0	1	1
38	0	0	0
39	1	1	0
40	1	0	1
41	0	1	1
42	0	0	0
43	0	0	0
44	0	0	0
45	1	0	1
46	1	0	1
47	0	1	1
48	0	0	0

ตารางที่ 7 ตัวอย่างการลดขนาดบิตโดยใช้ S-Box ทั้ง 8

S-Box	**แถว/ คอลัมภ์**	**Row**	**Column**
	บิตที่	**B1B6**	**B2B3B4B5**
S_1	Binary	10_2	1100_2
	Decimal	2	12
	Output(4บิต)	$3 = 0011_2$	
S_2	Binary	00_2	0101_2
	Decimal	0	5
	Output(4บิต)	$11 = 1011_2$	
S_3	Binary	10_2	1100_2
	Decimal	2	3
	Output(4บิต)	$9 = 1001_2$	
S_4	Binary	10_2	1100_2
	Decimal	1	12
	Output(4บิต)	$1 = 0001_2$	
S_5	Binary	10_2	1100_2
	Decimal	2	10
	Output(4บิต)	$5 = 0101_2$	
S_6	Binary	10_2	1100_2
	Decimal	2	9
	Output(4บิต)	$4 = 0100_2$	
S_7	Binary	10_2	1100_2
	Decimal	2	3
	Output(4บิต)	$13 = 1101_2$	
S_8	Binary	10_2	1100_2
	Decimal	0	7
	Output(4บิต)	$1 = 0001_2$	

เมื่อทำการลดขนาดบิตโดยใช้S-box แล้วจะได้ข้อมูลขนาด 32 บิต ขั้นตอนต่อไปคือทำการสลับค่าบิต โดยใช้ตาราง Primitive function P เมื่อทำการสลับค่าบิตเป็นที่เรียบร้อยแล้วจึงทำการ XOR อีกครั้งกับ L ก็จะได้ค่า Ri ตามลำดับ

ตารางที่ 8 Primitive function P

Primitive function P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

ตารางที่ 9 ตัวอย่างการ XOR ของ f(R,K) ขนาด 32 บิต

ค่าบิตที่	**F(R0,K1)**	**L0**	**Ri= L0 $\oplus$ F(R0,K1)**
1	1	1	0
2	1	1	0
3	1	1	0
4	0	1	1
5	0	1	1
6	1	1	0
7	1	1	0
8	0	1	1
9	0	0	0
10	0	0	0

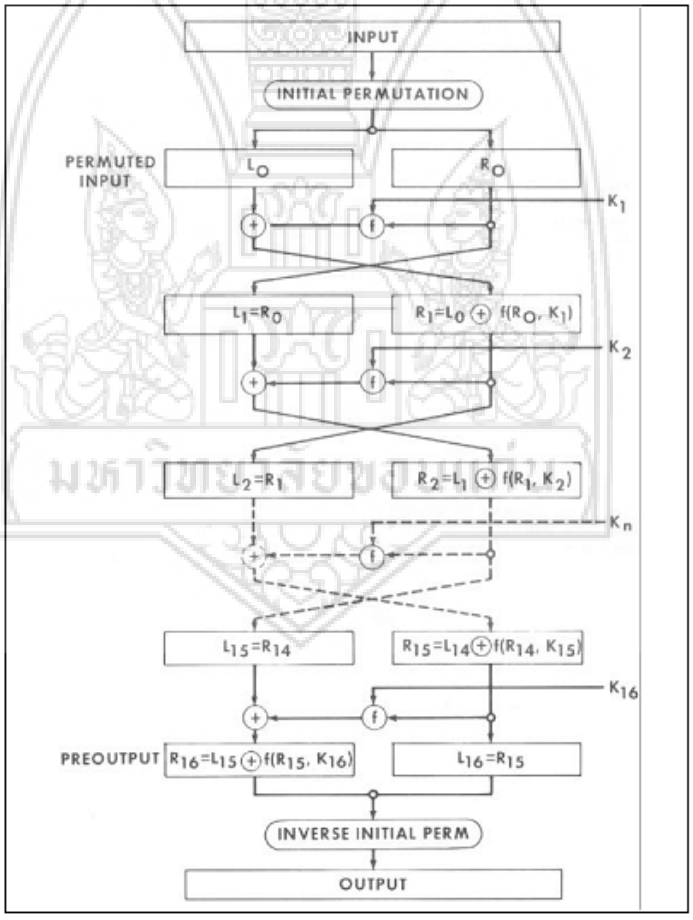
ตารางที่ 9 ตัวอย่างการ XOR ของ f(R,K) ขนาด 32 บิต (ต่อ)

ค่าบิตที่	F(R0,K1)	L0	Ri= L0 $\oplus$ F(R0,K1)
11	0	0	0
12	1	0	1
13	1	0	1
14	1	0	1
15	0	0	0
16	0	0	0
17	0	0	0
18	1	1	0
19	0	1	1
20	0	1	1
21	1	1	0
22	0	0	0
23	1	0	1
24	1	0	1
25	0	0	0
26	0	1	1
27	0	0	0
28	0	1	1
29	1	0	1
30	0	1	1
31	1	0	1
32	1	1	0

เมื่อดำเนินการสร้างกุญแจจนครบ 16 รอบการคำนวณขั้นตอนสุดท้ายคือ กลับค่าบิตจึงจะใช้ค่าในตาราง IP^{-1} เป็นขั้นตอนสุดท้ายก่อนทำการแสดงผลการคำนวณทั้งหมด จากการดำเนินการทั้งหมดสามารถสรุปเป็นขั้นตอนการคำนวณอัลกอริทึมแบบ DES ได้ดังภาพที่ 10

ตารางที่ 10 Inverse of the initial permutation

Inverse of the initial permutation (IP^{-1})							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25



ภาพที่ 10 สรุปขั้นตอนการคำนวณอัลกอริทึมแบบ DES [2]

2.2.2 Advanced Encryption Standard (AES)

เมื่อปีค.ศ. 1997 US NIST ได้มีการประกาศ ค้นหาอัลกอริทึมเพื่อคัดเลือกเป็นอัลกอริทึมใหม่สำหรับ AES ซึ่งจะนำมาแทนที่ DES ซึ่งการแข่งขันได้ถูกจัดขึ้นในปีค.ศ. 1998 และสิ้นสุดลงในปี ค.ศ. 2000 โดยมีอัลลกอรึมที่ผ่านรอบสุดท้ายและถูกเลือกมาเป็นอัลกอริทึมใหม่สำหรับ AES คือ Rijindael [3] ในเดือนตุลาคม 2000

NIST ได้มีการเปิดเผยวิธีการทำงานของอัลกอริทึม โดยที่ AES อัลกอริทึมนั้นจะทำการเพิ่มขนาดของ block ข้อมูลที่นำเข้ามาคำนวณ จาก 64 bits เป็น 128 bits และขนาดของกุญแจจาก 128 bits เป็น 256 bits จากการประเมินและวิเคราะห์ รอบการทำงาน ของ Rijndael ซึ่งถูกออกแบบโดย Rijmen & Daemen จากประเทศเบลเยี่ยม โดยอัลกอริทึม AESที่ถูกเลือกนั้นมีคุณสมบัติหลัก ดังนี้

(1) ขนาดของ blockที่ใช้ในการคำนวณ มีขนาด 128-bit

(2) ขนาดของกุญแจ ที่ใช้ในการคำนวณ มีขนาด 128, 192 หรือ 256

(3) ในการทำงานมีทางเลือกมากกว่า Feistel cipher (คล้ายกับ IDEA)

(4) มีการจัดการขนาดข้อมูล 4 กลุ่ม ของ 4 ไบท์

(5) ในการทำงานซึ่งมีรอบในการคำนวณ 9, 11 หรือ 13 โดยแต่ละรอบนั้นจะบรรจุไปด้วย

(5.1) ขั้นตอนในการแทนค่า ไบท์ (1 S-box ถูกนำมาใช้กับทุก ไบท์)

(5.2) ขั้นตอนในการเลื่อนแถว (การสับเปลี่ยนไบท์ระหว่างกลุ่ม)

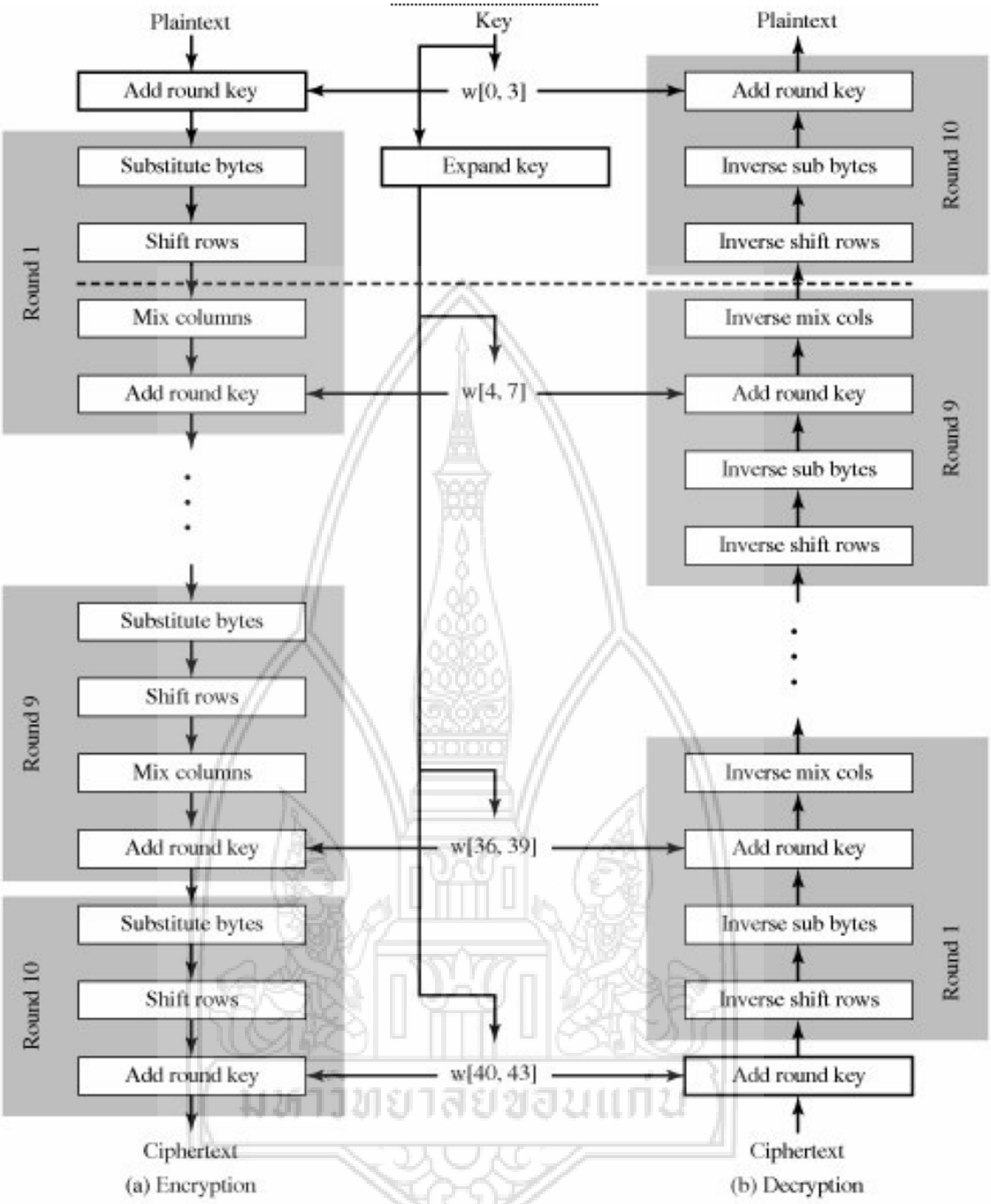
(5.3) ขั้นตอนในการผสมคอลัมภ์ (การคูณเมตริทต์ของกลุ่ม กับ ส่วนอื่นๆ)

(5.4) ขั้นตอนในการเพิ่มรอบของกุญแจ

(6) การดำเนินการทั้งหมดถูกนำมาเปรียบเทียบ xor และ ตาราง lookups ดังนั้นจึงทำให้การทำงานมีทั้งประสิทธิภาพความเร็วที่เพิ่มมากขึ้น

วิธีการทำงานของ Advance Encryption Standard (AES)

เป็นอัลกอริทึมที่ถูกคิดค้นและพัฒนาโดย Rijmen & Daemen หรือเรียกกันทั่วไปว่า Rijndael มีการใช้เทคนิคขนาดของคีย์ (Key Size) และ ขนาดของข้อมูล (Block Size) ซึ่งขนาดของคีย์สามารถเลือกได้เป็น 128 บิต ,192 บิต และ 256 บิต AES จะมีตัวแปรที่เก็บรอบการทำงานซึ่งจะขึ้นกับขนาดของ คีย์ โครงสร้างการเข้ารหัสและถอดรหัสของอัลกอริทึมเออีเอสอย่างง่ายแสดงได้แสดงในภาพที่ 11



ภาพที่ 11 โครงสร้างการเข้ารหัสและถอดรหัสของอัลกอริทึมเออีเอสอย่างง่าย

2.2.2.1 Notation and Conventions

(1) Bytes

การนำเข้าและส่งออกข้อมูลสำหรับอัลกอริทึม AES นั้นบรรจุไปด้วยอนุกรมของ 128 บิต (ค่า 1 หรือ 0) อนุกรมเหล่านี้บางครั้งอ้างอิงถึง blocks และจำนวนของบิตที่

บรรจุอยู่ในนั้น เพื่ออ้างอิงถึงขนาดของความยาว ในส่วนของ Cipher key สำหรับ AES อัลกอริทึมนั้น เป็นอนุกรมของ 128 , 192 หรือ 256 บิต หากความยาวอื่นนอกจากที่กำหนดนี้จะไม่ตรงกับมาตรฐานที่ตั้งๆไว้นี้

โดยพื้นฐานหน่วยในการประมวลผล ใน AES อัลกอริทึม คือ ไบท์ (bytes) ซึ่งเป็นชุดอาร์เรย์ที่ใช้ในการคำนวณ โดยกำหนดให้ เป็น ตัวแปร a(n)และ n คือ ค่าในช่วงต่อไปนี้

Key length = 128 bits, $0 \leq n < 16$; block length = 128 bits , $0 \leq n < 16$;

Key length = 192 bits, $0 \leq n < 24$;

Key length = 256 bits, $0 \leq n < 32$;

ค่าไบท์ทั้งหมดใน AES อัลกอริทึม จะนำเสนอในรูปของค่าบิต 1 หรือ 0 ระหว่างช่วงค่าที่เรียงลำดับ คือ{ b_7 , b_6 , b_5 , b_4 , b_3 , b_2 , b_1 , b_0} โดยที่สมาชิกแต่ละตัวสามารถเขียนอยู่ในรูป polynomial ดังนี้

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i$$

ยกตัวอย่างเช่น {01100011} เมื่อเปรียบเทียบกับสมาชิกคือ $x^6 + x^5 + x + 1$

เพื่อความสะดวกในการเขียนค่าไบท์จึงได้ใช้สัญลักษณ์เลขฐานหก ซึ่งแต่ละกลุ่มหมายถึงสี่บิตแทนตัวอักษรดัง ภาพที่ 12

Bit Pattern	Character
0000	0
0001	1
0010	2
0011	3

Bit Pattern	Character
0100	4
0101	5
0110	6
0111	7

Bit Pattern	Character
1000	8
1001	9
1010	a
1011	b

Bit Pattern	Character
1100	c
1101	d
1110	e
1111	f

ภาพที่ 12 Hexadecimal representation of bit patterns [3]

ยกตัวอย่างเช่น เลขฐานสอง{01100011} เมื่อแปลงเป็นฐานสิบหก คือ {63}

(2) Arrays of Bytes

อาร์เรย์ ของไบท์จะแสดงอยู่ในรูปแบบของ

$$a_0 a_1 a_2 \cdots\cdots a_{15}$$

ไบท์และบิตที่เรียงลำดับภายในไบท์ได้รับจากอนุกรมที่นำเข้าขนาด 128 บิต

$$input_0 input_1 input_2 \cdots\cdots input_{126} input_{127}$$

เมื่อนำมาเรียงใส่อาร์เรย์

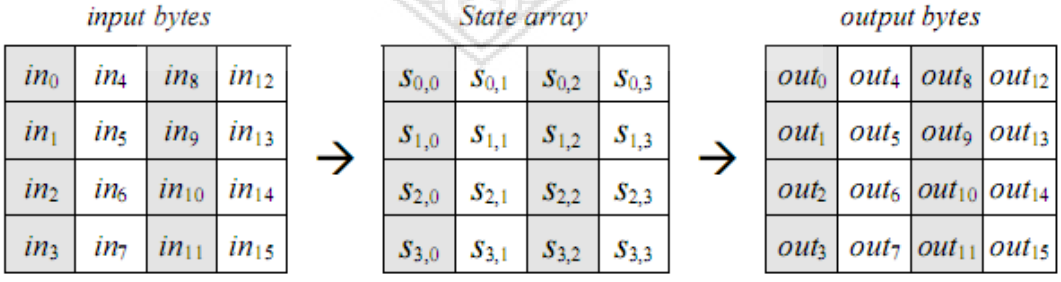
$$a_0 = \{input_0, input_1, \ldots, input_7\};$$
$$a_1 = \{input_8, input_9, \ldots, input_{15}\};$$
$$\vdots$$
$$a_{15} = \{input_{120}, input_{121}, \ldots, input_{127}\};$$

รูปแบบการวางสามารถถูกขยายให้เป็นชุดอนุกรมที่ใช้มีค่ายาวกว่านี้ เช่น กุญแจขนาด 192 บิต และ 256 บิต ดังนั้นสามารถเขียนในรูปทั่วไปได้ดังนี้

$$a_n = \{input_{8n}, input_{8n+1}, \ldots, input_{8n+7}\};$$

(3) The State

ภายในการดำเนินงานของ AES อัลกอริทึม อยู่ในรูป อาร์เรย์ 2 มิติ ของไบท์ ซึ่งถูกเรียกว่า state โดยประกอบไปด้วย สี่แถวของไบท์ แต่ละแถวบรรจุไปด้วย Nb ไบท์ ซึ่ง Nb คือ ความยาวของบล็อกที่ถูกหารด้วย 32 State array ถูกแสดงโดยสัญลักษณ์ s โดยแต่ละอันของไบท์มีสองตัวบ่งชี้ (index) กับจำนวนแถว r ซึ่งอยู่ในช่วง $0 \leq r < 4$ และ จำนวนคอลัมน์ซึ่งอยู่ในช่วง $0 \leq c < Nb$ ไบท์แต่ละตัวของ State ถูกอ้างอิงโดย $S_{r,c}$ หรือ S[r,c] สำหรับมาตรฐานนี้ Nb = 4 และ $0 \leq c < 4$ ลักษณะการนำเข้าและส่งออก อาร์เรย์ของไบท์ เป็นดังภาพด้านล่าง



ภาพที่ 13 State array input and output [3]

ดังนั้น การเริ่มต้นของ Cipher หรือ Inverse Cipher การนำเข้าอาร์เรย์ in หมายถึงการคัดลอก มายัง Stateอาร์เรย์ เป็นไปตามโครงสร้างดังนี้

$$s[r,c] = in[r+4c] \quad \text{สำหรับ } 0 \le r < 4 \text{ และ } 0 \le c < Nb,$$

และตอนสุดท้ายของ Cipher หรือ Inverse Cipher นั้น State ทำการคัดลอกส่งออก อาร์เรย์ out ดังนี้

$$out[r+4c] = s[r,c] \quad \text{สำหรับ } 0 \le r < 4 \text{ และ } 0 \le c < Nb,$$

(4) The State as an Array of Columns

สี่ไบท์ในแต่ละคอลัมน์ ของ State อาร์เรย์ ในรูปคำ 32 บิต ซึ่งจำนวนแถว (r) ถูกจัดไว้บ่งชี้ถึงสำหรับสี่ไบท์ภายในแต่ละคำ ดังนั้นการแปล state ให้สามารถเขียนให้อยู่ในรูป อาร์เรย์ขนาด หนึ่งมิติของ 32 บิต คำ(columns) , $w_0 \ldots\ldots w_3$, ซึ่ง จำนวนคอลัมน์ (c) ถูกจัดเตรียมไว้ เป็นตัวบ่ชี้ถึงอาร์เรย์เหล่านี้ ยกตัวอย่าง State ที่สามารถเขียนให้อยู่ในรูปอาร์เรย์ สี่คำ ดังภาพด้านล่าง

$$w_0 = s_{0,0}\, s_{1,0}\, s_{2,0}\, s_{3,0} \qquad w_2 = s_{0,2}\, s_{1,2}\, s_{2,2}\, s_{3,2}$$
$$w_1 = s_{0,1}\, s_{1,1}\, s_{2,1}\, s_{3,1} \qquad w_3 = s_{0,3}\, s_{1,3}\, s_{2,3}\, s_{3,3}.$$

ภาพที่ 14 แสดงตัวอย่างอาร์เรย์ของสี่คำ [3]

2.2.2.2 Mathematical Preliminaries

จำนวนไบท์ทั้งหมดใน AES อัลกอริทึม ถูกแปลความหมายเช่นเดียวกับสมาชิกฟิลด์จำกัด (Finite Field Elements) ซึ่งมีความสามารถในการบวก การคูณ แต่ในการดำเนินการแตกต่างจากการใช้สำหรับจำนวน (Number) โดยที่แนวความคิดทางคณิตศาสตร์ที่นำมาใช้มีดังนี้

(1) การบวก (Addition)

การบวกของสองสมาชิกใน ฟิลด์จำกัด (Finite Field) หาค่าโดย การบวกของสัมประสิทธิ์ของเลขชี้กำลังตัวเดียวกันของ พหุนาม (Polynomials) สำหรับสมาชิก 2 ตัว การบวกยังหมายถึงการดำเนินการแบบ XOR (สัญลักษณ์ที่แสดงถึงคือ $\oplus$) เช่น modulo 2 หมายถึง $1 \oplus 1 = 0$, $1 \oplus 0 = 1$ และ $0 \oplus 0 = 0$

การบวกของ สมาชิกฟิลด์จำกัด (finite field elements) สามารถอธิบายได้เช่นเดียวกับการบวกแบบ modulo 2 ของการแสดงถึงบิตในไบท์ สำหรับสองไบท์ $\{a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0\}$ และ $\{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\}$ รวมกันได้คือ $\{c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0\}$ ซึ่งแต่ละ $c_i = a_i \oplus b_i$ (เช่น $c_7 = a_7 \oplus b_7$, $c_6 = a_6 \oplus b_6 \ldots c_0 = a_0 \oplus b_0$)

ยกตัวอย่าง การแสดงถึงการเท่ากันของค่าที่สามารถเขียนให้อยู่ในรูปแบบอื่นได้ ดังนี้

$$\left(x^6 + x^4 + x^2 + x + 1\right) + \left(x^7 + x + 1\right) \quad = \quad x^7 + x^6 + x^4 + x^2 \quad \text{(รูปแบบ พหุนาม)}$$

$\{01010111\} \oplus \{10000011\} = \{11010100\}$ (รูปแบบ เลขฐานสอง)

$\{57\} \oplus \{83\} = \{d4\}$ (รูปแบบเลขฐานสิบหก)

(2) การคูณ (Multiplication)

ในการเขียนในรูปแบบพหุนาม การคูณในGF(2^8) (สัญลักษณ์ ●) สอดคล้องกับการคูณของพหุนามmodulo พหุนามลดทอนไม่ได้ (irreducible polynomial) ของดีกรี 8 พหุนามรูปแบบนี้หมายถึง ตัวหารที่นำมาหารได้มีเพียงแค่หนึ่งหรือตัวมันเองเท่านั้น สำหรับ AES อัลกอริทึม irreducible polynomial คือ

$$m(x) \quad = \quad x^8 + x^4 + x^3 + x + 1$$ หรือเขียนในรูปแบบเลขฐานสิบหก คือ {01}{1b}

สำหรับ Rijndael เลขยกกำลังของ x ถึงเลข 8 นั้นไม่สามารถแสดงได้ เพียงไบท์เดียว ซึ่งเมื่อเขียนจะได้ว่า 1{00011011} หรือ 1{1b}

ยกตัวอย่างเช่น {57} ● {83} = {c1}

วิธีทำ {57} = {0101 0111} = $\left(x^6 + x^4 + x^2 + x + 1\right)$

{83} = {1000 0011} = $\left(x^7 + x + 1\right)$

{57} ● {83} $= \left(x^6 + x^4 + x^2 + x + 1\right) \bullet \left(x^7 + x + 1\right)$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

เนื่องจาก ดีกรีของผลคูณมากกว่า 8 จึงต้องทำการลดดีกรีของผลคูณโดยใช้ m(x)

$$m(x) \bullet x^5 = (x^8 + x^4 + x^3 + x + 1) \bullet x^5 = x^{13} + x^9 + x^8 + x^6 + x^5$$

{57} ● {83} - $m(x) \bullet x^5$ $= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 -$
$x^{13} + x^9 + x^8 + x^6 + x^5$

$= x^{11} + x^4 + x^3$ // ดีกรียังคงมากกว่า 8 จึงต้องลดดีกรีอีก

$$m(x) \bullet x^3 = (x^8 + x^4 + x^3 + x + 1) \bullet x^3 = x^{11} + x^7 + x^6 + x^4 + x^3$$

{57} ● {83} - $m(x) \bullet x^5$ - $m(x) \bullet x^3$ $= x^{11} + x^4 + x^3 -$
$x^{11} + x^7 + x^6 + x^4 + x^3$

$= x^7 + x^6 + 1$

เมื่อผลลัพธ์ที่ได้มีดีกรีน้อยกว่า 8 ก็จะเป็นคำตอบของผลคูณ คือ {1100 0001} หรือ {c1}

2.2.2.3 Algorithm Specification

สำหรับ AES อัลกอริทึม ความยาวของ block ที่นำเข้า ส่งออก และ state นั้นมีค่า 128 บิต นั้นหมายถึง Nb มีค่าเท่ากับ 4 ซึ่งเป็นต่อจำนวน 32 บิตคำ (จำนวนของคอลัมน์) ใน stateในการคำนวณจำนวนรอบในการทำงานเป็นไปตามมาตรฐานดังภาพด้านล่างนี้

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

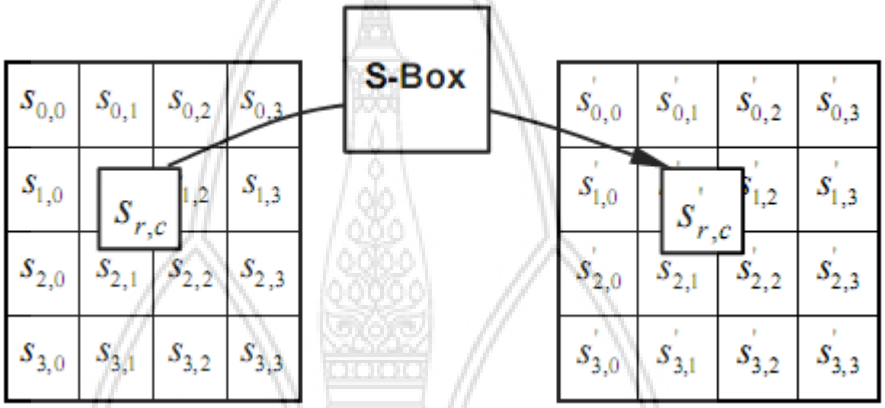
ภาพที่ 15 Key-Block-Round Combinations [3]

(1) Cipher

การเริ่มต้นของ Cipher นั้น เริ่มจากนำข้อมูลเข้ามาใน อาร์เรย์ หลังจากนั้นก็เพิ่มกุญแจเริ่มต้น (initial round key) เข้าไปในการทำงาน โดยการทำงานของฟังก์ชันรอบนั้น คือ 10 , 12 หรือ 14 ครั้ง (ขึ้นอยู่กับความยาวของกุญแจ) ซึ่งกระบวนการทำงานประกอบไปด้วย SubBytes ( ), ShiftRows ( ), MixColumns ( ) และ AddRoundKey( )

(1.1) SubBytes ( ) Transformation

เป็นการแทนค่าใน State ด้วยตาราง S-box



ภาพที่ 16 SubBytes ( ) ที่ประยุกต์ใช้ S-box ในแต่ละ ไบท์ ของ State [3]

ยกตัวอย่างเช่น ถ้า $s_{1,1} = \{53\}$ เมื่อต้องการแทนที่ค่านี้ โดยเทียบค่าเลขฐานสิบหกจากตาราง S-box คือ แถว x ที่ 5 และ คอลัมน์ y ที่ 3 ซึ่งจะได้ค่าใหม่ที่จะนำมาแทนที่คือ $s'_{1,1} = \{ed\}$

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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
	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

ภาพที่ 17 S-box สำหรับการแทนที่ค่า byte xy (ในรูปแบบเลขฐานสิบหก) [3]

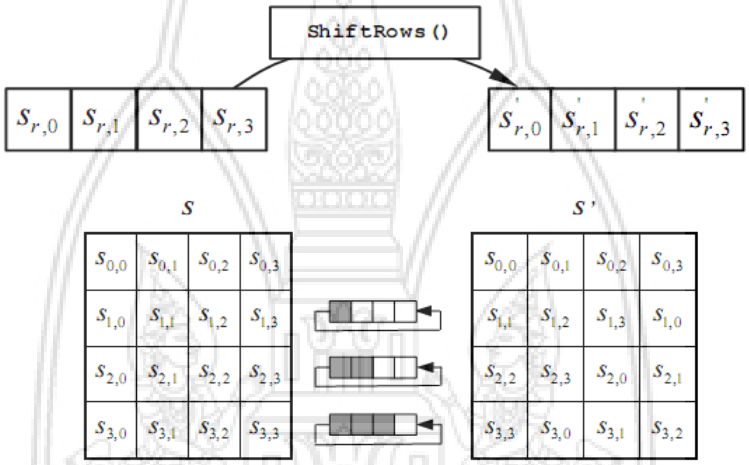
(1.2) ShiftRows ( ) Transformation

ไบท์ในสามแถวสุดท้ายจะทำการเลื่อนด้วยจำนวนที่แตกต่างกัน (offsets) โดยที่แถวแรกไม่ต้องเลื่อน หมายถึง r = 0 ซึ่งเป็นไปตามสมการด้านล่าง

$$S'_{r,c} = S_{r,(c+shift(r,Nb))\bmod Nb} \quad \text{สำหรับ } 0 \le r < 4 \text{ และ } 0 \le c < Nb,$$

ซึ่งค่าที่ต้องเลื่อน shift(r,Nb)นั้นขึ้นอยู่กับจำนวนแถว (r) เมื่อกำหนดให้ Nb = 4 จะได้ค่าดังนี้

$$shift(1,4) = 1; \quad shift(2,4) = 2; \quad shift(3,4) = 3.$$



ภาพที่ 18 ShiftRow ( ) การเลื่อน 3 แถว [3]

(1.3) MixColumns ( ) Transformation

เป็นการคูณคอลัมน์ต่อคอลัมภ์ $s'(x) = a(x) \otimes s(x)$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad \text{สำหรับ } 0 \le c < Nb,$$

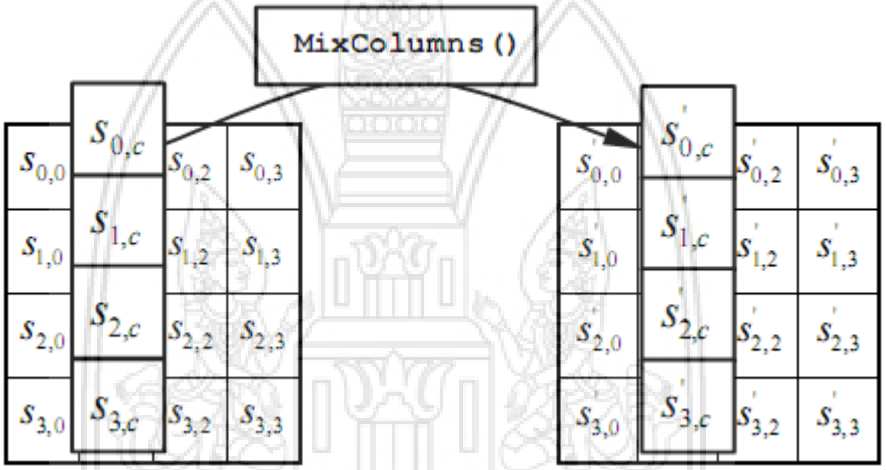
ซึ่งผลคูณที่ได้จะถูกนำไปแทนที่ในการทำงานครั้งต่อไป

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$



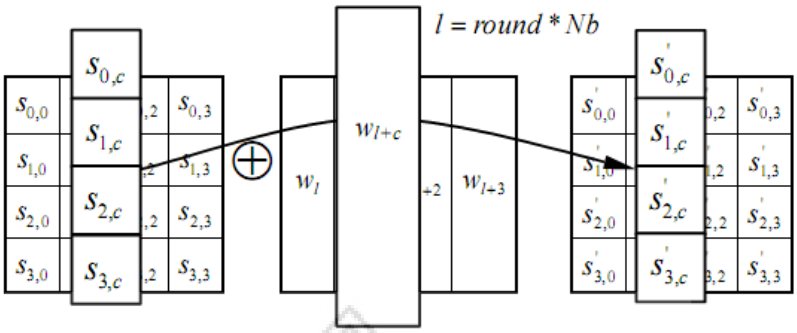
ภาพที่ 19 MixColumns ( ) ทำงานแบบ คอลัมน์ต่อคอลัมภ์ [3]

(1.4) AddRoundKey ( ) Transformation

เป็นขั้นตอนของการเพิ่มรอบกุญแจ (round key) เข้าไปในการทำงานแบบ bitwise XOR ดังนี้

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [w_{round*Nb+c}]$$

สำหรับ $0 \leq c < Nb$,



ภาพที่ 20 AddRoundKey ( ) XORs แต่ละคอลัมน์ของ State กับ คำ จาก ตารางกุญแจ [3]

(2) การขยายกุญแจ (Key Expansion)

อัลกอริทึม AES นั้นใช้ Cipher key (K) และมีการสร้างกุญแจเป็นรอบๆ การทำงาน โดยที่ผลลัพธ์ของตารางกุญแจนั้นบรรจุด้วย อาร์เรย์เชิงเส้นของ 4 ไบท์คำ ซึ่งแทนด้วย $[w_i]$ โดยที่ i อยู่ในช่วง $0 \leq i < Nb(Nr+1)$.

SubWord ( ) เป็น ฟังก์ชันที่นำเข้าสี่ไบท์คำ และ ประยุกต์ใช้กับ S-box ในแต่ละสี่ไบท์คำได้สร้างคำออกมา และRotWord ( ) ใช้คำ $[a_0, a_1, a_2, a_3]$ นำเข้ามา ดำเนินการแบบ cyclic permutation และส่งค่ากลับเป็น $[a_1, a_2, a_3, a_0]$

Rcon[j] คือ รอบคงที่ของอาร์เรย์คำ ซึ่งบรรจุไปด้วย $[x^{i-1}, \{00\}, \{00\}, \{00\}]$

โดยสามารถอธิบายได้ดังภาพด้านล่างนี้

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented;
they are all included in the conditional statement above for
conciseness.   Specific implementation requirements for the
Cipher Key are presented in Sec. 6.1.
```

ภาพที่ 21 Pseudo code สำหรับ Key Expansion [3]

ตัวอย่างการคำนวณ AES อัลกอริทึม

Input = 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Encryption Process

นำเลขฐานสิบหกเข้าสู่ State

input			
32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

⊕

Key			
2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

=

Result XOR			
19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

ใช้ S-box

After SubBytes			
d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

After ShiftRows			
d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

เลื่อนมา 1 ช่อง
เลื่อนมา 2 ช่อง
เลื่อนมา 3 ช่อง

ตัวอย่างการคำนวณ XOR จำนวน 1 คอลัมน์									
Hexadecimal	Binary Code								Result
32	0	0	1	1	0	0	1	0	
2b	0	0	1	0	1	0	1	1	
32⊕ 2b	0	0	0	1	1	0	0	1	19
43	0	1	0	0	0	0	1	1	
7e	0	1	1	1	1	1	1	0	
43 ⊕ 7e	0	0	1	1	1	1	0	1	3d
f6	1	1	1	1	0	1	1	0	
15	0	0	0	1	0	1	0	1	
f6 ⊕ 15	1	1	1	0	0	0	1	1	e3
a8	1	0	1	0	1	0	0	0	
16	0	0	0	1	0	1	1	0	
a8 ⊕ 16	1	0	1	1	1	1	1	0	be

ในการ MixColumns นั้นจากตาราง ShiftRows ขั้นตอนต่อไปจะนำค่าในคอลัมน์มาทำการ MixColumns โดยกำหนดให้

$$S_{0,c} = d4, \qquad S_{1,c} = bf, \qquad S_{2,c} = 5d, \qquad S_{3,c} = 30$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

จากสูตรการ MixColumns

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

แทนค่าในสูตร

$$S'_{0,c} = (\{02\} \bullet d4) \oplus (\{03\} \bullet bf) \oplus 5d \oplus 30 \qquad \text{สูตรที่ 1}$$

$$S'_{1,c} = d4 \oplus (\{02\} \bullet bf) \oplus (\{03\} \bullet 5d) \oplus 30 \qquad \text{สูตรที่ 2}$$

$$S'_{2,c} = d4 \oplus bf \oplus (\{02\} \bullet 5d) \oplus (\{03\} \bullet 30) \qquad \text{สูตรที่ 3}$$

$$S'_{3,c} = (\{03\} \bullet d4) \oplus bf \oplus 5d \oplus (\{02\} \bullet 30) \qquad \text{สูตรที่ 4}$$

ยกตัวอย่างการ MixColumns

โดยที่ จากสูตรที่ 1

$$S'_{0,c} = (\{02\} \bullet d4) \oplus (\{03\} \bullet bf) \oplus 5d \oplus 30$$

ขั้นตอนการหา {02}●d4

02 = 00000010, d4 = 11010100 , สัญลักษณ์ ● คือ Finite field multiplication
โดยที่ ค่าเลขชี้กำลังพหุนามคือ ค่าตำแหน่งบิตที่มีค่าเป็น 1

$$02 \bullet d4 = x\left(x^7 + x^6 + x^4 + x^2\right) = x^8 + x^7 + x^5 + x^3$$

หากผลการคูณมีเลขชี้กำลังมากกว่า 7 จะต้องทำการ modulo กับ $x^8 + x^4 + x^3 + x + 1$
เสียก่อน
ดังนั้น

$$= x^8 + x^7 + x^5 + x^3 \bmod x^8 + x^4 + x^3 + x + 1$$
$$= x^7 + x^5 + x^4 + x + 1$$

เมื่อนำ $x^7 + x^5 + x^4 + x + 1$ มาเทียบเป็นเลขฐานสองจะมีค่าเท่ากับ 10110011

ขั้นตอนการหา {03}●bf

03 = 00000011, bf = 10111111 , สัญลักษณ์ ● คือ Finite field multiplication
โดยที่ ค่าเลขชี้กำลังพหุนามคือ ค่าตำแหน่งบิตที่มีค่าเป็น 1

$$03 \bullet bf = (x+1)\left(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1\right)$$
$$= x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$$
$$= x^8 + x^7 + x^6 + 1$$

เนื่องจากเลขชี้กำลังมากกว่า 7 จึงต้องทำการ modulo กับ $x^8 + x^4 + x^3 + x + 1$
ดังนั้น

$$= x^8 + x^7 + x^6 + 1 \bmod x^8 + x^4 + x^3 + x + 1$$
$$= x^7 + x^6 + x^4 + x^3 + x$$

เมื่อนำ $x^7 + x^6 + x^4 + x^3 + x$ มาเทียบเป็นเลขฐานสองจะมีค่าเท่ากับ 11011010

Operation	Binary code
{02}●d4	10110011
{03}●bf	11011010
({02}●d4) ⊕ ({03}●bf)	01101001
5d	01011101
({02}●d4) ⊕ ({03}●bf) ⊕ 5d	00110100
30	00110000
({02}●d4) ⊕ ({03}●bf) ⊕ 5d⊕30	00000100
Hexadecimal is	04

ดังนั้นคำตอบของ $S'_{0,c} = (\{02 \bullet d4\}) \oplus (\{03\} \bullet bf) \oplus 5d \oplus 30$ คือ 04

จากสูตรที่ 2

$$S'_{1,c} = d4 \oplus (\{02\} \bullet bf) \oplus (\{03\} \bullet 5d) \oplus 30$$

ขั้นตอนการหา {02}●bf

02 = 00000010, bf = 10111111 , สัญลักษณ์ ● คือ Finite field multiplication

โดยที่ ค่าเลขชี้กำลังพหุนามคือ ค่าตำแหน่งบิตที่มีค่าเป็น 1

$$02 \bullet bf = x(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) = x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$$

หากผลการคูณมีเลขชี้กำลังมากกว่า 7 จะต้องทำการ modulo กับ $x^8 + x^4 + x^3 + x + 1$ เสียก่อน

ดังนั้น

$$= x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x \bmod x^8 + x^4 + x^3 + x + 1$$

$$= x^6 + x^5 + x^2 + 1$$

เมื่อนำ $x^6 + x^5 + x^2 + 1$ มาเทียบเป็นเลขฐานสองจะมีค่าเท่ากับ 01100101

ขั้นตอนการหา {03}●5d

03 = 00000011, 5d = 01011101, , สัญลักษณ์ ● คือ Finite field multiplication

โดยที่ ค่าเลขชี้กำลังพหุนามคือ ค่าตำแหน่งบิตที่มีค่าเป็น 1

$$\begin{aligned} 03 \bullet 5d &= (x+1)\left(x^6 + x^4 + x^3 + x^2 + 1\right) \\ &= x^7 + x^5 + x^4 + x^3 + x + x^6 + x^4 + x^3 + x^2 + x + 1 \\ &= x^7 + x^6 + x^5 + x^2 + x + 1 \end{aligned}$$

เนื่องจากผลการคูณเลขชี้กำลังไม่เกินกว่า 7 จึงไม่ต้องทำการ modulo กับ $x^8 + x^4 + x^3 + x + 1$ ดังนั้น จึงนำ $= x^7 + x^6 + x^5 + x^2 + x + 1$ มาเทียบเป็นเลขฐานสองจะมีค่าเท่ากับ 11100111

Operation	Binary code
d4	11010100
{02}●bf	01100101
d4 ⊕({02}●bf)	10110001
{03}●5d	11100111
d4 ⊕({02}●bf) ⊕({03}● 5d)	01010110
30	00110000
d4 ⊕({02}●bf) ⊕({03}● 5d) ⊕30	01100110
Hexadecimal is	66

ดังนั้นคำตอบของ $S'_{1,c} = d4 \oplus (\{02\} \bullet bf) \oplus (\{03\} \bullet 5d) \oplus 30$ คือ 66

AddRoundKey เป็นขั้นตอนหลังจากดำเนินการ MixColumns เป็นที่เรียบร้อย แล้วจึงดำเนินการ XOR ค่าที่ได้กับกุญแจ ทีละคอลัมน์ ดังนี้

After MixColumns			
04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

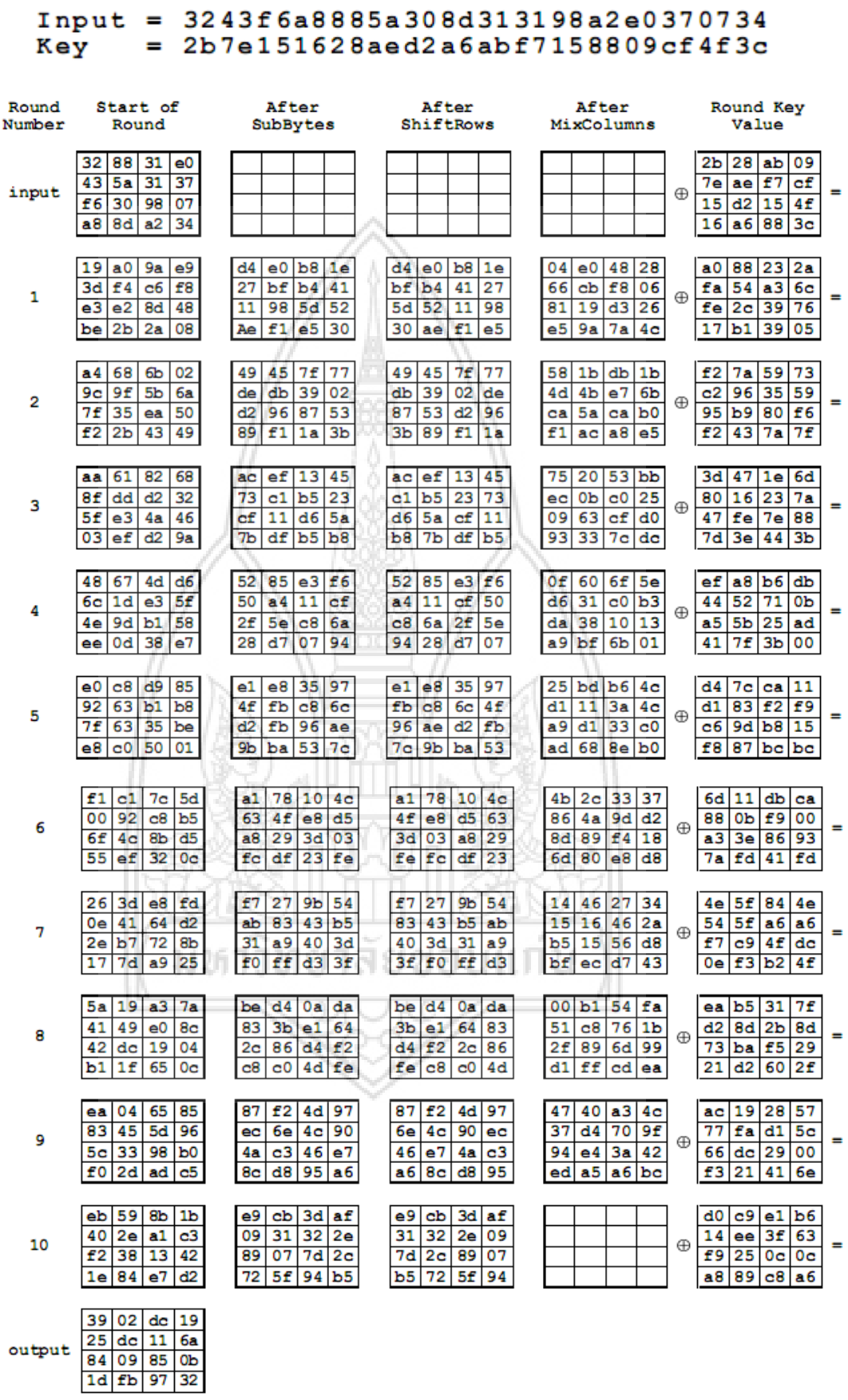
⊕

Round Key 1			
a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	B1	39	05

=

Result			
a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

ตัวอย่างการคำนวณ XOR จำนวน 1 คอลัมน์									
Hexadecimal	Binary Code								Result
04	0	0	0	0	0	1	0	0	
a0	1	0	1	0	0	0	0	0	
04⊕ a0	1	0	1	0	0	1	0	0	a4
66	0	1	1	0	0	1	1	0	
fa	1	1	1	1	1	0	1	0	
66 ⊕ fa	1	0	0	1	1	1	0	0	9c
81	1	0	0	0	0	0	0	1	
fe	1	1	1	1	1	1	1	0	
81 ⊕ fe	0	1	1	1	1	1	1	1	7f
e5	1	1	1	0	0	1	0	1	
17	0	0	0	1	0	1	1	1	
e5 ⊕ 17	1	1	1	1	0	0	1	0	f2



ภาพที่ 22 สรุปการคำนวณ AES อัลกอริทึม ขนาด 128 บิต ซึ่ง มีค่าในรูปเลขฐานสิบหก [3]

เป็นกระบวนการสร้างกุญแจสำหรับการเข้ารหัสในแต่ละรอบการคำนวณนั้นค่าที่ (Round constant) แทนด้วยคำย่อ Rcon และ j คือ รอบที่คำนวณ ซึ่งจะให้ค่าต่างกันและมีวิธีการนำไปใช้ดังนี้

กำหนดให้

$$Rcon[j] = [RC[j], 0, 0, 0]$$

และ $$RC[1] = 1,$$

$$RC[j] = 2 \bullet RC[j-1]$$

โดยที่ค่าการคำนวณของ RC อยู่ในรูปของเลขฐานสิบหก ดังตารางด้านล่าง

ตารางที่ 11 Key Schedule

J	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1b	36

ยกตัวอย่างในการคำนวณ

รอบที่ 2 ค่า j=2 แทนค่า

RC[2]	=	2●RC[2-1]
	=	2●RC[1]
	=	2
binary		0000 0010
hex		02

รอบที่ 3 ค่า j=3 แทนค่า

RC[3]	=	2●RC[3-1]
	=	2●RC[2]
	=	4
binary		0000 0100
hex		04

รอบที่ 4 ค่า j=4 แทนค่า

RC[4]	=	2●RC[4-1]
	=	2●RC[3]
	=	8
binary		0000 1000
hex		08

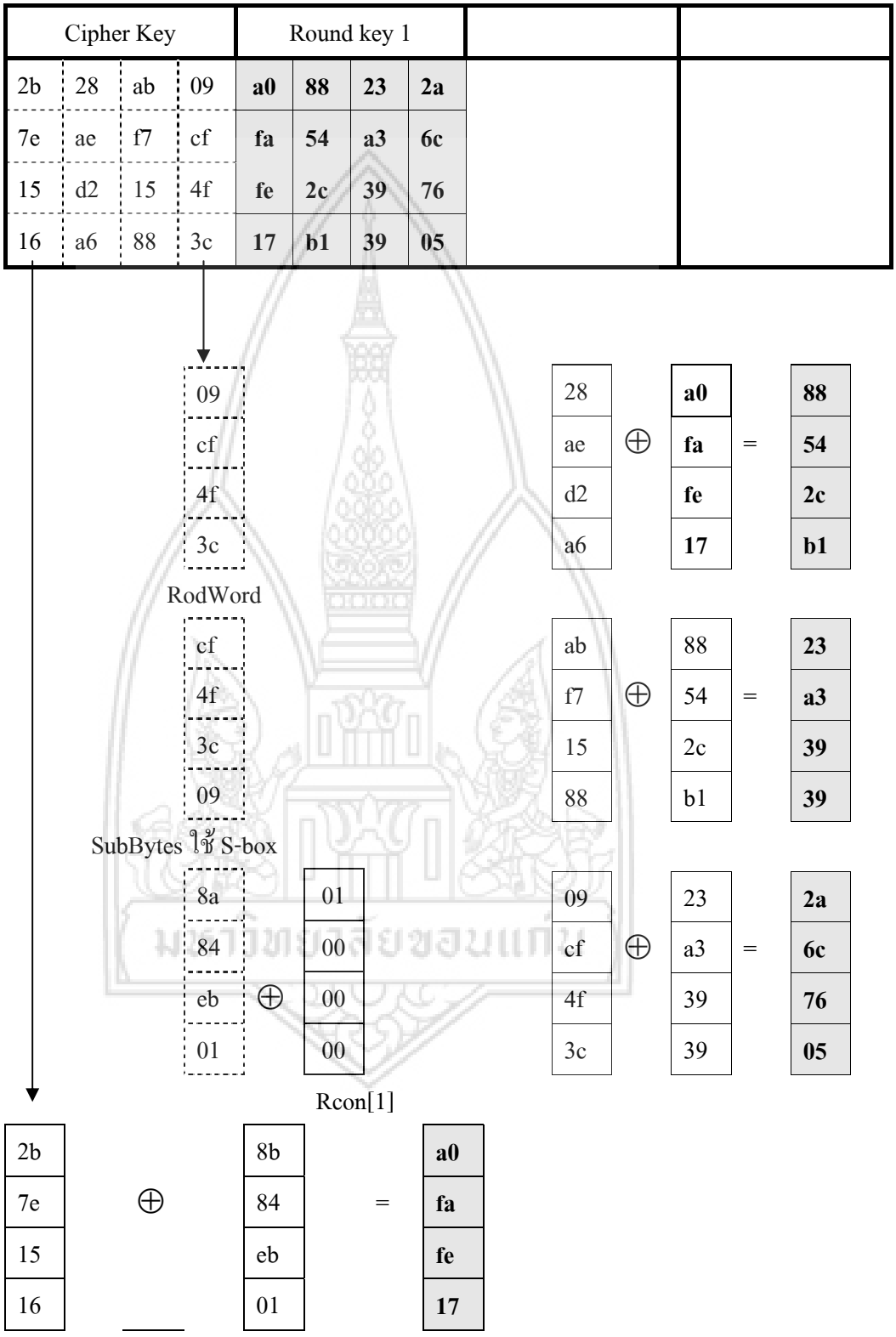
รอบที่ 5 ค่า j=5 แทนค่า

RC[5]	=	2●RC[5-1]
	=	2●RC[4]
	=	16
binary		0001 0000
hex		10

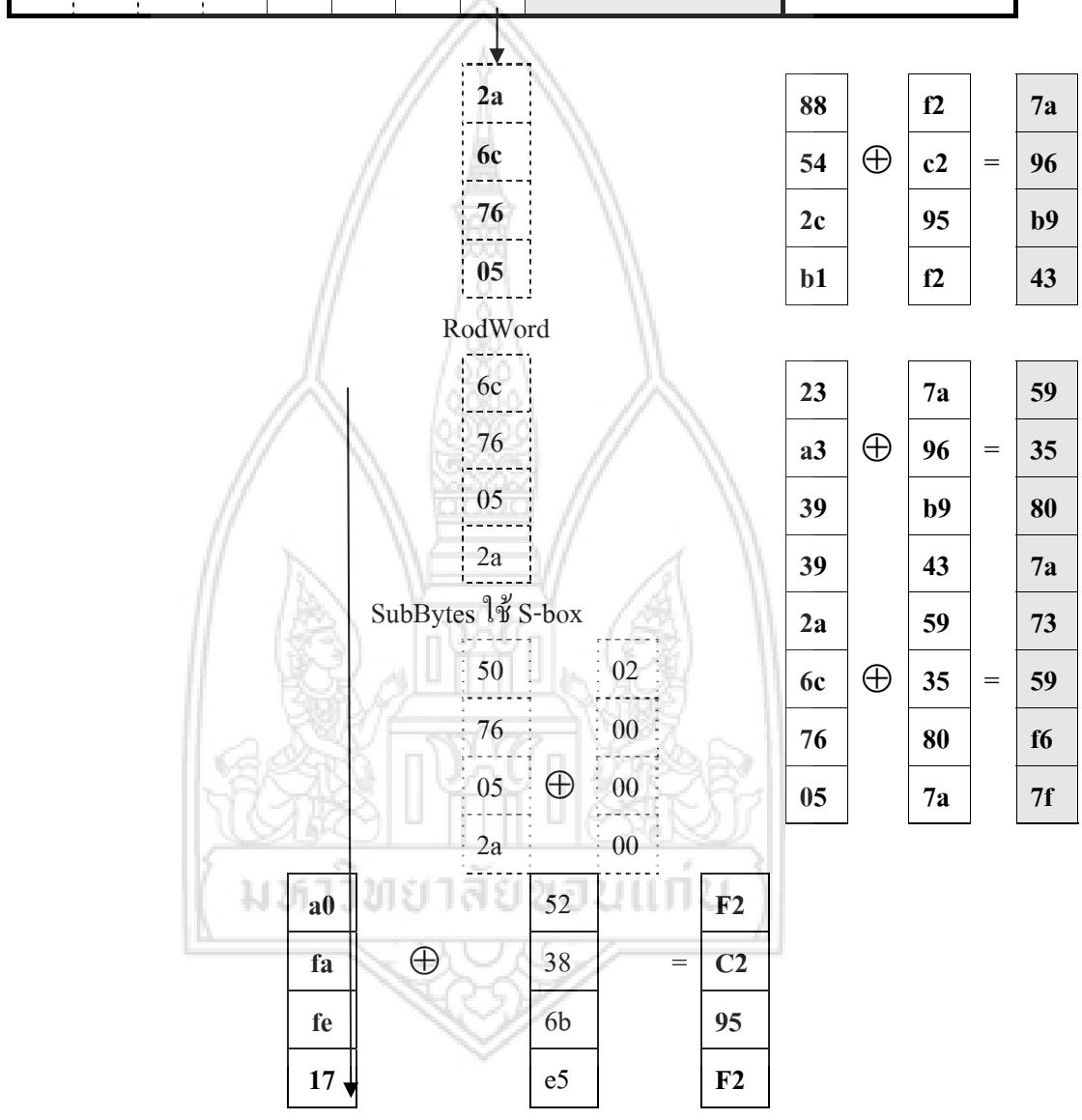
จากคำนวณครบ 10 รอบ จะได้ ตาราง Rcon ดังนี้

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

เริ่มต้นการคำนวณตารางกุญแจ



Cipher Key				Round key 1				Round key 2				
2b	28	ab	09	a0	88	23	2a	**f2**	**7a**	**59**	**73**	
7e	ae	f7	Cf	fa	54	a3	6c	**c2**	**96**	**35**	**59**	
15	d2	15	4f	fe	2c	39	76	**95**	**b9**	**80**	**f6**	
16	a6	88	3c	17	b1	39	05	**f2**	**43**	**7a**	**7f**	



Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

เมื่อ Nk = 4

w_0 = 2b7e1516 w_1 = 28aed2a6 w_2 = abf71588 w_3 = 09cf4f3c

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR

3. ยูนิโค้ด (Unicode)

ปัจจุบันมีลักษณะมีตัวอักษรให้เลือกใช้อยู่หลายแบบ ซึ่งมี Character set อยู่มากมาย เช่น ตัวอักษรภาษาไทย, ตัวอักษรภาษาญี่ปุ่น ฯลฯ ในอดีตการ Encode ภาษา หากเป็น ASCII จะมีขนาดในการจัดเก็บเป็น 1 byte ซึ่งมีขนาด 8 bit [4] โดยที่ ASCII code นั้นมีลักษณะการจัดเก็บตัวอักษรภาษาอังกฤษตัวเล็ก, ตัวใหญ่, ตัวเลข, เครื่องหมายมากกว่า, น้อยกว่า, ไม่เท่ากับ, full stop, # เป็นต้น ดังนั้นการใช้พื้นที่เพียง 7 bit ก็สามารถ Encode ข้อมูลได้ครบถ้วน ในส่วน บิตที่ 8 นั้นจะทำการเติมค่า 0 ลงไปในบิต

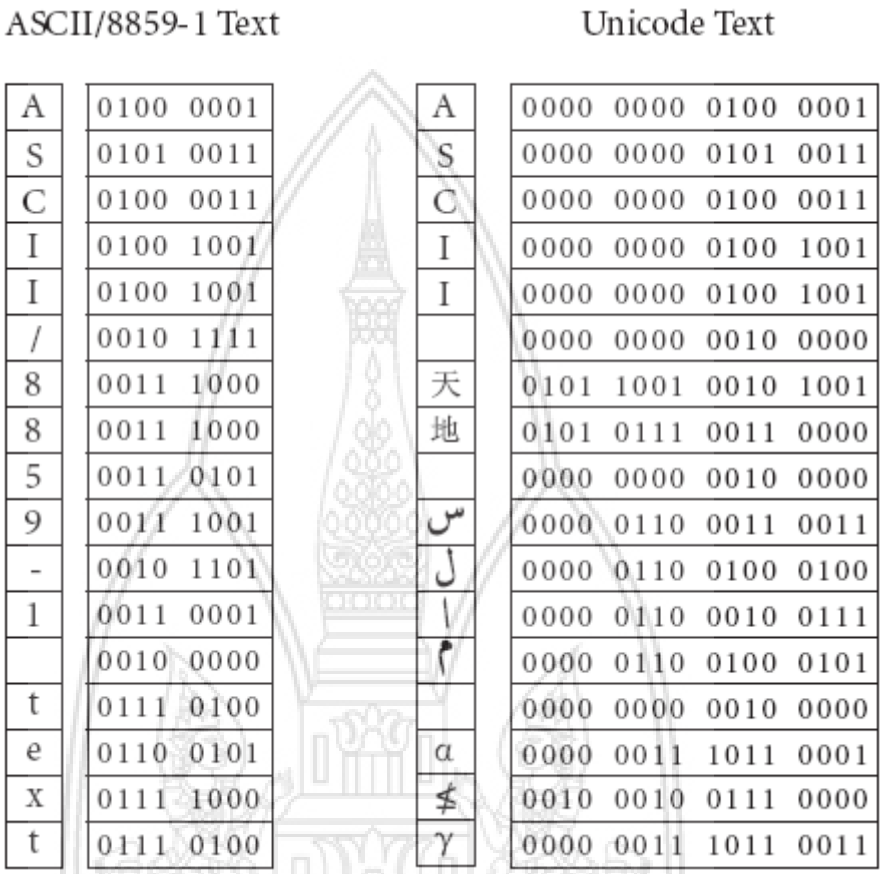
หากแต่ในแต่ละประเทศนั้นมิได้ใช้ตัวอักษรภาษาอังกฤษในการเขียนเท่านั้น หากแต่ มีความต้องการมีรหัสในการเขียนตัวอักษรเป็นของประเทศของตัวเองเช่นกัน ดังนั้นจึงได้ทำการแก้ปัญหาก็คือ จะนำเอาบิต ที่ 8 มาใช้ เช่น โดยทำการตั้ง บิต ที่ 8 ให้มีค่าเป็น 1 แล้วจึง Encode ด้วยรหัสของตัวเอง หากแต่ต้องเป็นรหัส ASCII code ที่ยังไม่ได้ถูกนำไปใช้งาน เช่น 1000011 เป็น ‘ก’, 1000012 เป็น ‘ข’, 1000013 เป็น ‘ค’ เป็นต้น

จากวิธีการดังกล่าวจะทำให้สามารถจัดเก็บได้ 2 ภาษา แต่การทำลักษณะแบบนี้ไม่ใช่วิธีสากลทั่วไปที่ถูกนำไปใช้งาน ยกตัวอย่างเช่นหากภาษาจีนหรือภาษาญี่ปุ่นมีการใช้ลักษณะแบบเดียวกัน คือนำเอา bit ที่ 8 มาใช้ ด้วยเหตุนี้ ตัวอักษร ‘ก’ ของภาษาไทยก็จะมีรหัสเหมือนกับตัวอักษรบางอย่างในภาษาจีน ดังนั้นจึงได้มีการแก้ปัญหาเพื่อทำให้เป็นมาตรฐานสากลทั่วโลก จึงควรมีวิธีการสร้างรหัสเป็นลักษณะเฉพาะของสำหรับอักษรตัวเดียว และในแต่ละภาษา ด้วยเหตุนี้จึงได้มีการสร้าง Unicode ขึ้นมาใช้งาน โดยที่ Unicode นั้นมีความแตกต่างจาก ASCII ตรงส่วนที่ ASCII นั้น เก็บ byte เพียง byte เดียว หากแต่ Unicode นั้นเก็บ 2 byte ซึ่งข้อมูลขนาด 2 byte นั้นสามารถเก็บข้อมูลได้มากมายมหาศาล ซึ่งสามารถจัดเก็บข้อมูลได้มากมายหลายภาษาในโลก เช่นเดียวกับภาษาไทยที่มีการจัดเก็บมาตรฐานแบบ Unicode นี้เช่นกัน ดังนั้นหากนำรหัสภาษาไทยไปเปิดใช้ในภาษาจีน ก็ยังคงแสดงเป็นตัวอักษรภาษาไทยอยู่ ไม่แสดงตัวอักษรเป็นภาษาจีน เพราะว่ามีการจัดแบ่ง code ไว้ให้แต่ละภาษาชัดเจน หมายความว่า code นี้จองพื้นที่ไว้สำหรับจัดเก็บตัวอักษรภาษาไทย หมายความว่าแต่ละประเทศจะมี code ใช้ที่ไม่ซ้ำกัน เป็นต้น

3.1 หลักการทำงานของ Unicode

รหัสแบบที่กำหนด 1 byte หรือ 8 บิตสำหรับ 1 ตัวอักษรนั้นมีความเหมาะสมและเพียงพอสำหรับภาษาที่ใช้ตัวอักขระแต่ละตัวแทนเสียงในภาษา (alphabetical language) แต่สำหรับภาษาที่เขียนโดยใช้ตัวอักขระแทนพยางค์หรือหน่วยคำ เช่น ภาษาญี่ปุ่น ภาษาจีน จะมีจำนวนอักขระที่เขียนเป็นจำนวนมากเกินกว่าที่จะแทนด้วย 1 byte ได้ (ซึ่งแทนได้เพียง 256 แบบของตัวอักขระ) อีกทั้งเมื่อมีการใช้รหัสอักขระที่แตกต่างกัน ก็จะมีผลต่อการย้ายข้อมูลข้ามภาษา เช่น รหัสที่แทนตัวอักขระ ก ที่ใช้ภาษาไทยจะไปตรงกับรหัสตัวอักขระ ¡ ในอีกตารางหนึ่ง เป็นต้น จึงมีความ

พยายามแก้ปัญหาให้มีรหัสเดียวที่ใช้ได้กับอักขระทุกภาษา ซึ่งเป็นที่มาของการพัฒนารหัส Unicode ขึ้นมาตั้งแต่ปีค.ศ.1991 (Unicode 1.0) โดยที่รหัสตัวอักขระ 256 ตัวแรกนั้นจะเหมือนกับรหัสของ ISO-8859 ปัจจุบันพัฒนามาถึง Unicode 5.1



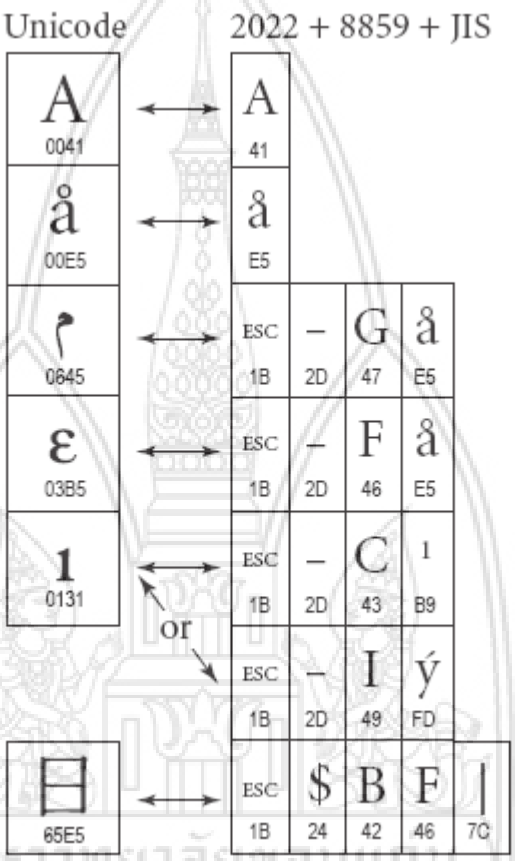
ASCII/8859-1 Text	
A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

Unicode Text	
A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
س	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0010 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
≤	0010 0010 0111 0000
γ	0000 0011 1011 0011

ภาพที่ 24 ตัวอย่างการเข้ารหัสตัวอักษร ASCII และ Unicode [4]

หลักการของ Unicode คือแต่ละตัวอักขระที่ใช้ในภาษาต่างๆ จะมีรหัสเฉพาะของตน แต่รูปแบบของการเข้ารหัสแบบ Unicode นั้นมีได้หลายแบบ เป็นการแปลงรหัส Unicode ที่กำหนด หรือที่เรียกว่า Unicode transformation format (UTF) เป็นชุดของ byte เฉพาะสำหรับแต่ละอักขระ (unique byte sequence) ในแบบที่ง่ายสุดคือใช้จำนวนบิตคงที่(เหมือนการแทนรหัสแบบอื่นที่กล่าวมา) จะใช้จำนวน 32 บิตสำหรับแต่ละอักขระ (4 bytes/character) (เรียกว่า UTF-32) แต่จะเสียเนื้อที่เก็บข้อมูลมากไม่ประหยัด และเพราะตัวอักษรต่างๆที่ใช้บ่อยๆ ก็มักเป็นเหมือนรหัส ISO-8859 ซึ่งใช้เพียง 8 บิตสำหรับแต่ละตัวอักษรก็พอ จึงมีการคิดวิธีการเข้ารหัสที่ประหยัดที่เก็บมากขึ้นโดยจำนวนบิตของแต่ละอักขระไม่จำเป็นต้องคงที่การเข้ารหัสแบบ UTF-16 กำหนดให้ตัวอักขระที่ใช้

บ่อยๆ เก็บ 16 บิต ส่วนตัวอักขระที่ใช้ไม่บ่อยจะเก็บเป็น 32 บิต (2-4bytes/character) ส่วนการเข้ารหัสแบบ UTF-8 จะยิ่งประหยัดมากขึ้น เพราะตัวอักขระที่ใช้บ่อยจะเก็บ 8 บิตส่วนตัวอักขระอื่นๆ อาจเก็บ 16 หรือ 32 บิต (1-4 bytes/character) จึงทำให้ประหยัดที่เก็บข้อมูลมากขึ้นรหัสแบบ Unicode นี้เป็นที่คาดการณ์ว่าจะเป็นมาตรฐานสำหรับการเก็บข้อมูลภาษาในคอมพิวเตอร์ในอนาคต



ภาพที่ 25 ตัวอย่างรูปแบบการเข้ารหัสที่ออกแบบให้ประหยัดที่เก็บข้อมูลมากขึ้น [4]

3.2 รูปแบบการเข้ารหัสตัวอักษรที่เป็น Unicode

การเข้ารหัสตัวอักษรในระบบ Unicode มีมากมายหลายวิธีซึ่งวิธีที่ใช้กันเป็นที่ทั่วไปคือ UTF-16 ใช้กันมากในระบบปฏิบัติการ Windows และ UTF-8 ถูกใช้ในระบบปฏิบัติการ Linux

(3.3) UTF-16

16-bit Unicode Transformation Format เป็นวิธีการเข้ารหัสของ Unicode ในรูปแบบ 16 bits ต่อ 1 ตัวอักษร ซึ่งเรียกว่า code units ซึ่งการเข้ารหัสจะใช้ระบบ Basic Multilingual Plane (BMP) มากำหนดลำดับตัวอักษร โดยตำแหน่ง code ที่มีตัวอักษรจะอยู่ในช่วง U+0000 ถึง U+10FFFF ยกเว้น code ที่อยู่ระหว่าง U+D800 ถึง U+DFFF (ซึ่งจะไม่มีตัวอักษรใดๆ เลย) จะเป็นส่วนที่กำหนดไว้รองรับตัวอักษรที่จะถูกกำหนดในอนาคต

4. งานวิจัยที่เกี่ยวข้อง

4.1 ขั้นตอนวิธีในการสร้างรหัสภาษาไทย (Algorithms in Thai Encryption) [5]

งานวิจัยชิ้นนี้ ได้กล่าวถึงขั้นตอนวิธีการสร้างรหัสลับที่ได้จากการสำรวจและปรับปรุงวิธีการที่ใช้ในภาษาอังกฤษทั้ง 3 แบบ และนำมาปรับปรุงขั้นตอนวิธี เพื่อใช้กับภาษาไทยและพัฒนาไปสู่การสร้างระบบความลับสำหรับภาษาไทย โดยทำการแทนตัวอักษรภาษาไทยเลือกตัวอักษรพยัญชนะไทย 43 ตัว สระไทย 17 ตัว และ วรรณยุกต์ 4 ตัว แทนตัวอักษรเหล่านี้ด้วยตัวเลข 0 ถึง 63 ดังภาพที่ 19

ก	ข	ค	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ	ฐ	ฑ	ฒ
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ	ภ	ม	ย
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

ร	ฤ	ล	ว	ศ	ษ	ส	ห	ฬ	อ	ฮ	_ะ	_ั	า	_ำ	_ิ
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

_็	_่	_้	_๊	_๋	เ	แ	โ	ใ	ไ	_็	_	_	_	_	_
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

ภาพที่ 26 การแทนตัวอักษรภาษาไทยด้วยตัวเลข [5]

อุปกรณ์ที่ใช้นั้นเป็นเครื่องไมโครคอมพิวเตอร์ 486DX ขึ้นไป และชุดคำสั่งระบบจัดการภาษาไทยพร้อมกับตัวแปลโปรแกรมภาษา Pascal

วิธีการเรียงลำดับดังนี้

1. เลือกตัวอักษรภาษาไทยและจำนวนตัวอักษรที่เหมาะสม
2. ทดลองแทนตัวอักษรด้วยตัวเลข
3. ทดลองและปรับขั้นตอนการเข้ารหัสและถอดรหัสคำขนาดสั้น
4. เขียนโปรแกรมคอมพิวเตอร์จากขั้นตอนวิธีในข้อ 3
5. ทดสอบโปรแกรมคอมพิวเตอร์

สรุป จากผลการทดลองจะเห็นว่าขั้นตอนวิธีที่ปรับปรุงขึ้นนี้ใช้กับอักษรไทยได้ แต่ยังมีข้อจำกัดด้านโปรแกรมคอมพิวเตอร์ในเรื่องหน่วยความจำของเครื่องคอมพิวเตอร์และประเภทของตัวแปรภาษาในโปรแกรม แปลภาษาซึ่งสามารถเก็บค่าจำนวนเต็มได้จำกัด ทำให้ไม่สามารถทดลองใช้กับตัวเลขจำนวนใหญ่ๆ ได้

4.2 เปรียบเทียบประสิทธิภาพการทำงานของอัลกอริทึมในการเข้ารหัสข้อมูล (A Performance Comparison of Data Encryption Algorithms) [6]

งานวิจัยครั้งนี้ ได้นำอัลกอริทึมในการเข้ารหัสที่เป็นที่นิยมใช้ เช่น DES, 3DES, AES(Rijadael) และ Blowfish การนำอัลกอริทึมในการเข้ารหัสเหล่านี้มาเปรียบเทียบประสิทธิภาพการทำงาน โดยการนำเข้าไฟล์ข้อมูลที่มีขนาดแตกต่างกัน บนลักษณะฮาร์แวร์ที่แตกต่างกัน รูปแบบภาษาที่เลือกใช้นั้นเน้นที่เปรียบเทียบความเร็วในการทำงาน ผลการดำเนินงานครั้งนี้อยู่บนพื้นฐานของการทดลองที่สรุปได้ว่า Blowfish นั้น เป็นอัลกอริทึมในการเข้ารหัสที่ดีที่สุดจากการทดลองการเปรียบเทียบครั้งนี้ อัลกอริทึมที่เลือกมาใช้ในการดำเนินงานดำเนินงานครั้งนี้ คือ

(1) DES
(2) Triple DES
(3) AES(Rijndael)
(4) Blowfish

เมื่อเทียบกับ black cipher ประสิทธิภาพการทำงานของ stream cipher นั้น ขึ้นอยู่กับขนาดของ block และขนาดของกุญแจ แต่ผลของขนาดของ block ที่ใหญ่กว่า จะตรงกันข้ามอัลกอริทึมนั้นจะทำการคำนวณได้ช้ากว่า เพราะว่า block ที่มีขนาดใหญ่กว่านั้นจะ ต้องประมวลการทำงานที่นานกว่าสำหรับปริมาณข้อมูลที่มีขนาดเท่ากัน (bit or byte) ในการคำนวณแต่ละรอบ เมื่อเทียบกับ block ที่มีขนาดเล็กกว่า มีขนาดนำเข้าข้อมูลเท่ากัน จะมีประสิทธิภาพในการทำงานที่มากกว่าด้วยเหตุนี้เวลาที่ใช้ในการเข้ารหัสจึงลดลงเมื่อเทียบกับส่วนอื่นๆ ที่เท่ากัน

ผลลัพธ์ของกุญแจที่ใหญ่กว่า ใน stream cipher เท่ากันกับ แบบ block cipher คือ การเข้ารหัสจะช้าลงเพราะโดยทั่วไปทุกบิตของกุญแจที่เกี่ยวข้องนั้นจะนำไปคำนวณรอบการทำงานของอัลกอริทึมนั้น และกรณีกุญแจที่มีขนาดเล็กกว่านั้น จำนวนบิตของกุญแจก็จะลดลงทำให้รอบในการคำนวณลดลงในแต่ละครั้ง ส่วนอื่นๆ ก็มีค่าเท่ากัน

Measuring Execution Times

ความถูกต้องแม่นยำในการคำนวณของเวลาในการปฏิบัติงานนั้น ต้องพิสูจน์ด้วยการวัด ซึ่งมีความซับซ้อน ในการทดลองครั้งนี่ได้ทำการเลือก เครื่อง แบบ Pentium-II 266 MHz. (ทำงานบนระบบปฏิบัติการ ไมโครซอฟต์วินโดว์) และอีกเครื่องคือ Pentium -4 2.4 GHz. (ทำงานบนระบบปฏิบัติการ ไมโครซอฟต์วินโดว์ XP) ใช้เกณฑ์การวัดในเรื่องของเวลาเพราะวัถตุประสงค์หลัก ต้องการวัดเวลาในการเข้ารหัส ของอัลกอริทึม เวลาในการเริ่มสร้างกุญแจ รวมไปถึงการเปรียบเทียบ ทั่วไปเวลาในการถอดรหัสและเวลาในการเข้ารหัสนั้นเท่านั้น เกือบจะเท่ากันในทุกอัลกอริทึม ด้วยเหตุนี้เราจึงวัดแต่เวลาในการเข้ารหัสเท่านั้น

Performance Results for Block Ciphers

ผลลัพธ์ในการทำงานของ อัลกอริทึมกุญแจรหัสลับ ในโหมดของ ECB (Electronic Codebook) ถูกนำเสนอเป็นอันดับแรก โดยผลลัพธ์ที่ได้เป็นดังตารางที่ 12 และตารางที่ 13 สำหรับเครื่องแบบ Pentium-II 266 MHz. และอีกเครื่องคือ Pentium -4 2.4 GHz. เรียงตามลำดับ

ตารางที่ 12 เปรียบเทียบเวลาการทำงาน (หน่วย วินาที) ของ อัลกอริทึมกุญแจรหัสลับในโหมดของ ECB บนเครื่องแบบ Pentium-II 266 MHz.

Input size (bytes)	DES	3DES	AES	BF
20,527	24	72	39	19
36,002	48	123	74	35
45,911	57	158	94	46
59,852	74	202	126	58
69,545	83	243	143	67
137,325	160	461	285	135
158,959	190	543	324	158
156,364	198	559	355	162
191,383	227	655	378	176
232,398	276	799	460	219
Average Time	134	383	288	108
Bytes/sec	835	202	491	1,036

ตารางที่ 13 เปรียบเทียบเวลาการทำงาน (หน่วย วินาที) ของ อัลกอริทึมกุญแจรหัสลับในโหมดของ ECB บนเครื่องแบบ Pentium-4 2.4 GHz.

Input size (bytes)	**DES**	**3DES**	**AES**	**BF**
20,527	2	7	4	2
36,002	4	13	5	3
45,911	5	17	8	4
59,852	7	23	11	6
69,545	9	25	13	7
137,325	17	51	26	14
158,959	20	60	30	16
156,364	21	62	31	17
191,383	24	72	36	19
232,398	30	87	44	24
Average Time	14	42	21	11
Bytes/sec	7,988	2,883	5,320	10,187