วงจรเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้

นาย เจนโชติ ศรีพรประเสริฐ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2549

A DESIGN OF A RECONFIGURABLE AES ENCRYPTION CIRCUIT

Mr. Jenchote Sripornprasert

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

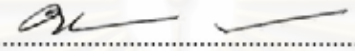
Faculty of Engineering

Chulalongkorn University

Academic Year 2006

หัวข้อวิทยานิพนธ์	วงจรเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้
โดย	นายเจน โชติ ศรีพรประเสริฐ
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	รองศาสตราจารย์. ดร. ประภาส จงสถิตย์วัฒนา

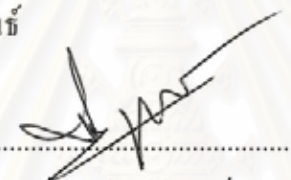
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต



..... คณบดีคณะวิศวกรรมศาสตร์

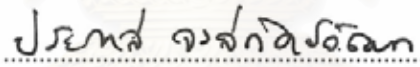
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์



..... ประธานกรรมการ

(อาจารย์ ดร. ฐิต ศิริบูรณ์)

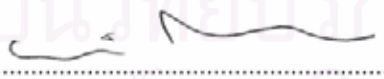


..... อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา)

..... กรรมการ

(อาจารย์ ดร.เศรษฐา ปานงาม)



..... กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. เขมะฑัต วิภาตะวนิช)

เจนโชติ ศรีพรประเสริฐ : วงจรเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้. (A DESIGN OF A RECONFIGURABLE AES ENCRYPTION CIRCUIT) อ. ที่ปรึกษา : รศ. ดร. ประภาส จงสถิตย์วัฒนา, 75 หน้า.

วิทยานิพนธ์นี้กล่าวถึงวงจรเข้ารหัสเออีเอสที่เปลี่ยนโครงแบบได้ วงจรเข้ารหัสดังกล่าวอาศัยข้อได้เปรียบจากการเปลี่ยนโครงแบบได้เพื่อลดขนาดโดยรวมของวงจรลง วงจรที่ได้จึงมีขนาดเล็กเหมาะสำหรับไปใช้ในงานที่ต้องคำนึงถึงความประหยัดเป็นอันดับแรก วงจรเข้ารหัสปกติใช้ 40,621 เกตสมมูลในขณะที่วงจรแบบเปลี่ยนโครงแบบได้ใช้เพียง 10,830 เกตสมมูล วงจรธรรมดาไม่ได้ใช้ทรัพยากรที่มีอยู่อย่างสูงสุดเนื่องจากต้องมีบางส่วนของวงจรที่ไม่ได้ทำงานอะไรเลยในบางเวลาแต่วงจรแบบเปลี่ยนโครงแบบนั้นมีลักษณะการทำงานคือ แตกงานใหญ่ให้เป็นงานย่อยๆแล้วทำงานย่อยให้เสร็จแล้วเปลี่ยนตัวเองไปทำงานย่อยอันต่อไป ด้วยพฤติกรรมที่ทำงานเล็กๆทีละอันนี้ จึงช่วยลดการสิ้นเปลืองส่วนที่ไม่ได้ถูกใช้งานไปได้มาก

งานวิจัยชิ้นนี้ได้ออกแบบวงจรประกอบด้วยสามส่วนหลักคือ หน่วยประมวลผลเปลี่ยนโครงแบบได้ หน่วยความจำรีจิสเตอร์แบงก์ และ หน่วยควบคุม หน่วยประมวลผลเปลี่ยนโครงแบบได้จะเป็นส่วนสำคัญที่ใช้ในการเปลี่ยนโครงแบบของวงจรให้เป็นวงจรย่อยที่ต้องการในขณะนั้น รีจิสเตอร์แบงก์มีหน้าที่เก็บข้อมูลจากโครงแบบก่อนหน้าเพื่อส่งต่อให้โครงแบบต่อไปทำงาน หน่วยควบคุมทำหน้าที่ควบคุมการไหลของสัญญาณให้เป็นไปอย่างถูกต้อง ขนาดของวงจรจะเท่ากับขนาดของวงจรย่อยที่ใหญ่ที่สุดรวมกับรีจิสเตอร์แบงก์และหน่วยควบคุม

ภาควิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต...เจนโชติ ศรีพรประเสริฐ.....

สาขาวิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่ออาจารย์ที่ปรึกษา...ประภาส จงสถิตย์วัฒนา

ปีการศึกษา.....2549.....

497 02660 21 : MAJOR COMPUTER ENGINEERING

KEY WORD: AES ENCRYPTION / RECONFIGURABLE EMBEDDED SYSTEM / A DYNAMICALLY RECONFIGURABLE PROCESSOR

JENCHOTE SRIPORNPRASERT : A DESIGN OF A RECONFIGURABLE AES ENCRYPTION CIRCUIT. THESIS ADVISOR : ASSOC. PROF. PRABHAS CHONGSTITVATANA, Ph.D., 75 pp.

In this thesis, we focused on AES encryption circuit with dynamic reconfiguration. This circuit has the reconfiguration ability to reduce overall circuit size. It is suitable for low budget application. The regular AES circuit consumes approximately 40,621 equivalent gates, but the reconfigurable circuit consumes only 10,830 equivalent gates. A key concept to reduce size is to divide the whole task into many subtasks. The reconfigurable circuit will execute one subtask then change itself for the next subtask, therefore it uses the resource more efficiently.

The reconfigurable circuit consists of three main components, a Reconfigurable Processing Unit, a Register Bank and a Control Unit. The Reconfigurable Processing Unit is a component which can be configured to any desired sub circuit. The Register Bank is a place to keep intermediate results between subtasks. The Control Unit controls overall data flow. The total circuit size is the sum of the largest circuit of subtasks, the Register Bank and the Control Unit.

Department .....Computer Engineering... Student's signature .....

Field of study.....Computer Engineering... Advisor's signature .....

Academic year.....2006 .....

กิตติกรรมประกาศ

ขอขอบคุณอาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ. ดร. ประภาส จงสถิตวัฒนา ที่เป็นผู้ที่จุดประกายแนะแนวทางการทำวิทยานิพนธ์ รวมทั้งยังเสียสละเวลาอันมีค่าให้คำปรึกษาที่เป็นประโยชน์ต่องานวิจัยชิ้นนี้มากมาย ขอขอบคุณคณะกรรมการสอบวิทยานิพนธ์ ซึ่งได้แก่ อ. ดร. ฐิตศิริบูรณ์ อ. ดร. เศรษฐา ปานงาม และ ผศ. ดร. เขมะฑัต วิภาตะวนิช เนื่องด้วยว่าวิทยานิพนธ์นี้จะเสร็จสมบูรณ์ไม่ได้หากไม่ได้รับคำแนะนำและชี้ให้ข้อบกพร่องที่ควรแก้ไขจากคณาจารย์เหล่านี้

ขอขอบคุณสมาชิกในห้องปฏิบัติการ ISL (Intelligent System Laboratory) และเพื่อนๆในระดับปริญญามหาบัณฑิตทุกคนที่คอยเอาใจใส่และช่วยสร้างบรรยากาศที่ดีในการทำงานตลอดมา

ขอขอบคุณจุฬาลงกรณ์มหาวิทยาลัยที่เป็นสถาบันที่ให้ความรู้ที่มีประโยชน์ยิ่งรวมไปถึงโรงเรียนกรุงเทพคริสเตียนวิทยาลัยที่ช่วยบ่มเพาะความรู้

สุดท้ายนี้ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ที่เป็นกำลังและแรงบันดาลใจสำคัญในการผลักดันให้เกิดผลสำเร็จอย่างเช่นทุกวันนี้

งานวิจัยนี้ได้รับเงินทุกสนับสนุนจากโครงการจัดการศึกษาสาขาวิชาวิศวกรรมศาสตร์ เพื่อเพิ่มศักยภาพทางด้านวิทยาศาสตร์ เทคโนโลยีและอุตสาหกรรม หมวดเงินอุดหนุนการศึกษาประจำปีการศึกษา 2549

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ระบบฝังตัวหมายถึงหน่วยประมวลผลขนาดเล็กหรือ ไมโครโปรเซสเซอร์ (Microprocessor) ที่ไปทำงานอยู่บนที่ที่ไม่ใช่คอมพิวเตอร์ทั่วไป เช่น โทรทัศน์, วิทยุ, รถยนต์, โทรศัพท์มือถือ เป็นต้น คอยทำการประมวลผลเฉพาะทางที่เกี่ยวกับระบบที่มันไปฝังตัวอยู่ เช่น ชิปประมวลผลในโทรทัศน์ กล่องควบคุมในรถยนต์

ในอดีตระบบฝังตัวที่มีหน่วยประมวลผลดังกล่าวมีขนาดเล็กและทำงานได้ดีอย่างไม่ขาดตกบกพร่อง แต่เนื่องจากเทคโนโลยีที่ก้าวหน้าขึ้นระบบฝังตัวก็มีการพัฒนาขึ้นทำงานได้หลากหลายและซับซ้อนยิ่งขึ้น หน่วยประมวลของระบบฝังตัวก็ถูกพัฒนาไปตามเทคโนโลยีที่ซับซ้อนขึ้นเช่นกัน และเนื่องจากการทำงานที่ซับซ้อนขึ้นมีความจำเป็นอย่างยิ่งที่ต้องเพิ่มขนาดของวงจรรวมทั้งพลังงานที่จะใช้ในการประมวลผล ความสิ้นเปลืองที่เกิดจากความซับซ้อนของเทคโนโลยีจึงมากขึ้นไปเงาตามตัว อย่างไรก็ตามระบบฝังตัวหนึ่งๆมีข้อจำกัดอย่างมากในด้านของทรัพยากรและพลังงานที่ใช้ ข้อจำกัดดังกล่าวสวนทางกันกับความต้องการการทำงานที่ซับซ้อนของวงจรในปัจจุบัน

ระบบฝังตัวสามารถทำงานได้โดยพึ่งสิ่งที่เรียกว่าโครงแบบหรือ Configuration เพื่อบ่งบอกลักษณะของวงจรว่ามีหน้าตาและพฤติกรรมเป็นอย่างไร ด้วยแนวคิดที่เรียกว่าการเปลี่ยนโครงแบบได้หรือ Reconfiguration ทำให้พฤติกรรมของวงจรเปลี่ยนไปอย่างสิ้นเชิง แนวคิดดังกล่าวเป็นแนวทางเพื่อสร้างวงจรที่มีความซับซ้อนได้มากขึ้นในขณะที่ขนาดของวงจรไม่จำเป็นต้องมีขนาดใหญ่ขึ้น

การประยุกต์แนวคิดการเปลี่ยนโครงแบบได้กับการเข้ารหัสก็เป็นอีกแนวทางที่ทำกันอย่างแพร่หลาย เนื่องจากวงจรการเข้ารหัสเป็นวงจรที่มีความซับซ้อนและมีขนาดใหญ่พอสมควร ดังนั้นการประยุกต์แนวคิดนี้เข้ากับการเข้ารหัสก็เพื่อลดขนาดของวงจรลงซึ่งได้มีการตีพิมพ์ออกมาเป็นผลงานทางวิชาการ [1-8] งานเหล่านี้ใช้แนวคิดการเปลี่ยนโครงแบบเข้าไปมีส่วนร่วมกับการทำงานแต่งานวิจัยชิ้นนี้มีแนวคิดที่ต่างออกไป

โครงงานวิจัยชิ้นนี้นำเสนอการเข้ารหัสแบบเออีเอสโดยอาศัยการเปลี่ยนโครงแบบได้เป็นแกนหลักสำหรับเข้ารหัส วงจรที่ออกมาจึงประหยัดกว่าวงจรเข้ารหัสในลักษณะทั่วไปในขณะที่ยังเข้ารหัสได้อย่างถูกต้องเหมือนวงจรปกติทุกประการ

1.2 วัตถุประสงค์ของการวิจัย

เพื่อพัฒนาวงจรเข้ารหัสแบบเออีเอส 128 บิต บนพื้นฐานของวงจรที่เปลี่ยนโครงแบบได้ที่สามารถเปลี่ยนแปลงตัวเองได้ในเวลารันไทม์โดยใช้อุปกรณ์รวมส่งสัญญาณ (Multiplexer) ในการจำลองการเปลี่ยนตัวเอง

1.3 ขอบเขตของการวิจัย

1. วงจรเข้ารหัสเออีเอสด้วยวิธีเปลี่ยนโครงแบบได้จะใช้การเปลี่ยนตัวเองด้วย อุปกรณ์รวมส่งสัญญาณเท่านั้น
2. วงจรเข้ารหัสเออีเอสด้วยวิธีเปลี่ยนโครงแบบได้ประมวลผลบนเครื่องมือจำลองการทำงานได้อย่างถูกต้อง
3. วงจรเข้ารหัสเออีเอสด้วยวิธีเปลี่ยนโครงแบบได้จะทำงานโดยไม่ติดต่อกับ หน่วยความจำหลัก
4. วงจรเข้ารหัสเออีเอสด้วยวิธีเปลี่ยนโครงแบบได้เข้ารหัสเออีเอสแบบ 128 บิตได้อย่างถูกต้อง

1.4 ประโยชน์ที่คาดว่าจะได้รับ

สร้างวงจรเข้ารหัสแบบเออีเอส 128 บิตบนพื้นฐานของวงจรที่เปลี่ยนโครงแบบได้ที่สามารถเปลี่ยนตัวเองตอนเวลารันไทม์ได้

1.5 วิธีดำเนินการวิจัย

1. ศึกษารายละเอียดเกี่ยวกับการทำงานโดยเปลี่ยนโครงแบบในเวลารันไทม์และการเข้ารหัสแบบเออีเอส
2. ออกแบบวงจรเข้ารหัสเออีเอสด้วยวิธีเปลี่ยนโครงแบบได้
3. สร้างวงจรเออีเอสให้นำมาใช้กับแนวคิดของวงจรเปลี่ยนโครงแบบได้
4. ทดลองและปรับปรุง
5. พัฒนาประสิทธิภาพ
6. สรุปผลและเรียบเรียงวิทยานิพนธ์

1.6 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตอบรับให้ตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง “The AES Encryption Circuit on a Reconfigurable Hardware” โดย เจนโชติ ศรีพรประเสริฐ และ รศ. ดร. ประภาส จงสถิตย์วัฒนา ในงานประชุมวิชาการ “Conf. of Electrical/Electronics, Computer, Telecommunications, and Information Technology (ECTI) 2007” ณ. มหาวิทยาลัยแม่ฟ้าหลวง จังหวัดเชียงราย ในระหว่างวันที่ 9 -11 พฤษภาคม 2550

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะขอกล่าวถึงทฤษฎีและงานวิจัยที่ใกล้เคียงกับงานวิจัยการเข้ารหัสด้วยวงจรเปลี่ยนโครงแบบได้ เพื่อที่จะนำความรู้ดังกล่าวมาประยุกต์เข้ากับงานวิจัยชิ้นนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1รินเดลล์บล็อคไซเฟอร์ (The Rijndael Block Cipher)

ไซเฟอร์ (Cipher) หมายถึงอัลกอริทึ่มสำหรับการเข้ารหัส (Encryption) และถอดรหัส (Decryption) ข้อมูลที่ต้องการส่งเป็นความลับ (Plain text) จะถูกไซเฟอร์เปลี่ยนรูปแบบ (Format) ให้อยู่ในรูปแบบที่ไม่สามารถเข้าใจได้ด้วยมนุษย์หรือคอมพิวเตอร์ถ้าปราศจากการถอดรหัส

เราแบ่งหมวดของไซเฟอร์ได้หลายวิธี ถ้าจัดหมวดหมู่โดยอาศัยขนาดของข้อมูลแล้วสามารถแบ่งไซเฟอร์ได้เป็นบล็อคไซเฟอร์ (Block Cipher) ที่ทำงานกับข้อมูลที่มีขนาดคงที่และสตรีมไซเฟอร์ (Stream Cipher) สำหรับกรณีที่ข้อมูลต่อเนื่องกันไปเรื่อยๆ นอกจากนี้ถ้าแบ่งหมวดตามคีย์ที่ใช้เข้ารหัสหรือถอดรหัสแล้วสามารถแบ่งได้เป็น อัลกอริทึ่มที่มีคีย์สมมาตร (Symmetric Key Algorithm) และอัลกอริทึ่มที่คีย์ไม่สมมาตร (Asymmetric Key Algorithm) ซึ่งในกรณีของการเข้ารหัสแบบเออีเอสถูกจัดอยู่เป็นอัลกอริทึ่มแบบบล็อคไซเฟอร์และทำงานแบบคีย์สมมาตร

รินเดลล์บล็อคไซเฟอร์เป็นทฤษฎีรากฐานของการเข้ารหัสแบบเออีเอส (AES) [9] ที่มีความยาวของคีย์และข้อความเป็น 128, 192 หรือ 256 ขั้นตอนการทำงานประกอบด้วย

1. ขั้นตอนการขยายคีย์ (KeyExpand)
2. ขั้นตอนการแทนที่ไบต์ (ByteSub)
3. ขั้นตอนการเลื่อนแถว (ShiftRow)
4. ขั้นตอนการผสมหลัก (MixColumn)
5. ขั้นตอนการบวกคีย์แต่ละรอบ (AddRoundKey)

ขั้นตอนการขยายคีย์ (KeyExpand) เป็นกระบวนการเพื่อหาคีย์ที่ใช้เพื่อเปลี่ยนสถานะ (State: สถานะของข้อความ เมื่อข้อความมีการเปลี่ยนแปลงโดยกระบวนการใดๆก็ตาม สถานะก็จะเปลี่ยนไป) ของข้อความที่เข้ามาจนได้เป็นผลลัพธ์สุดท้ายออกไป ซึ่งจะหาคีย์ทั้งหมด 10 รอบด้วยกันโดยระเบียบวิธีเป็น

```
Nk = 4:
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
       for(i = 0; i < Nk; i++)
              W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
       for(i = Nk; i < Nb * (Nr + 1); i++)
       {
              temp = W[i - 1];
```

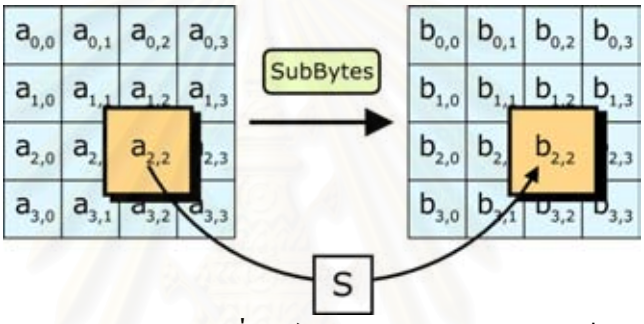
```
            if (i % Nk == 0)
            temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
            W[i] = W[i - Nk] ^ temp;
        }
}
```

*Nk – ความยาวของคีย์หารด้วย 32 , Nb – ความยาวของ block (ข้อความ) หารด้วย 32

*SubByte เป็นฟังก์ชันที่สร้างโดยตาราง look-up ซึ่งจะคืนค่าคงที่กลับมา

*Rcon เป็นค่าคงที่ที่ใช้ในสร้างคีย์ในแต่ละรอบ

ขั้นตอนการแทนที่ไบต์ (ByteSub) เป็นการแทนค่าเดิมด้วยวิธีการทำคูณแบบผกผัน (Multiplicative Inverse) บนฟังก์ชัน GF(2^8) หรือวิธีใช้ตารางแทนค่าลงไปตรงๆเลยก็ได้ โดยการนำอินพุตต์เข้าไปยังมอดูลที่ชื่อว่าเอสบ็อกซ์ (S- box) ดังรูปที่ 2.1 กระบวนนี้แทนด้วย ByteSub(State)



รูปที่ 2.1 ระเบียบวิธีการแทนที่ไบต์ (ByteSub) โดยอาศัยเอสบ็อกซ์ (S-box)

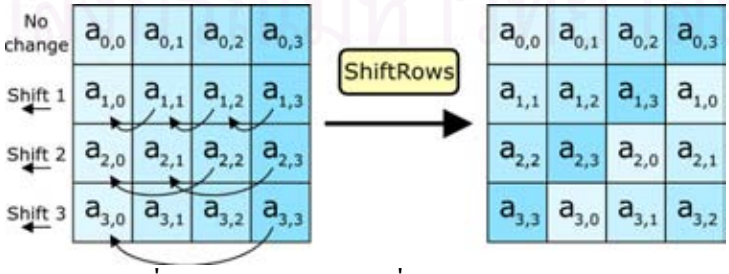
ขั้นตอนการเลื่อนแถว (ShiftRow) เป็นสถานะที่ทำการเลื่อนซ้ายแบบเป็นวงหรือ cyclic left shift โดยจำนวนครั้งที่เลื่อนขึ้นอยู่กับแถวนั้นๆ

แถวที่0 – ไม่มีการเลื่อน

แถวที่1 – เลื่อนซ้ายวนหนึ่งครั้ง

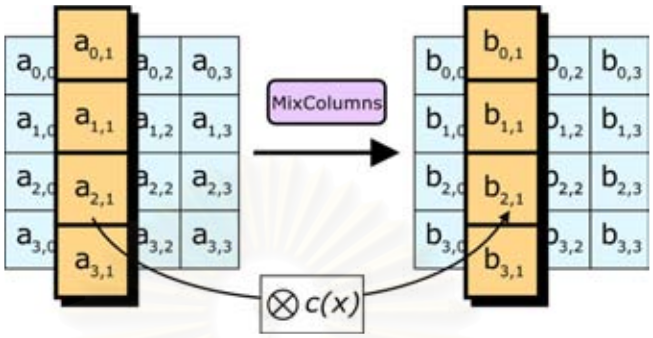
แถวที่2 – เลื่อนซ้ายวนสองครั้ง

แถวที่3 – เลื่อนซ้ายวนสามครั้ง ดังแสดงในรูปที่ 2.2 แทนด้วย ShiftRow(State)



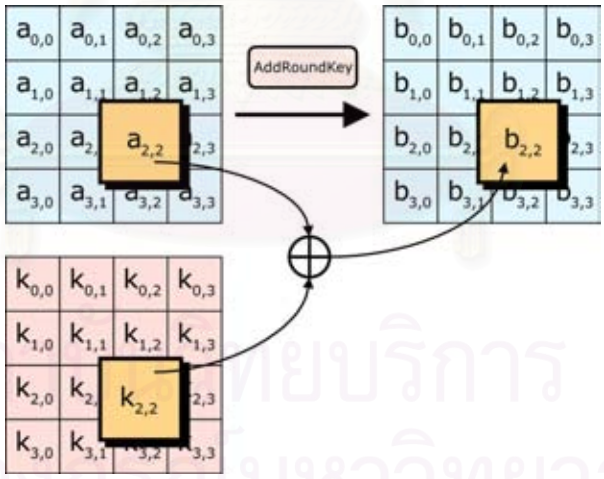
รูปที่ 2.2 ระเบียบวิธีการเลื่อนแถว (ShiftRow)

ขั้นตอนการผสมหลัก (MixColumn) คือการนำหลักแต่ละหลักของอินพุตต์มาพิจารณาแบบ GF(2^8) แล้วนำไปคูณมอดูโล (Multiply modulo) ด้วย $x^4 + 1$ กับพหุนาม ‘03’x^3 + ‘01’x^2 + ‘01’x + ‘02’ ดังรูปที่ 2.3 ซึ่งแทนด้วย MixColumn(State)



รูปที่ 2.3 ระเบียบวิธีการผสมหลัก (MixColumn)

ขั้นตอนการบวกคีย์แต่ละรอบ (AddRoundKey) เป็นกระบวนการเปลี่ยนสถานะของข้อความโดยการทำเอ็กซ์ออร์ (xor) กับคีย์ในรอบนั้นๆดังที่แสดงในรูปที่ 2.3
กระบวนการนี้แทนด้วย AddRoundKey(State, RoundKey)
โดย State คือข้อความ และ RoundKey คือ คีย์ในรอบนั้นๆ



รูปที่ 2.4 ระเบียบวิธีการบวกคีย์แต่ละรอบ

จากรูปที่ 2.4 ตัวแปรที่แสดงในรูประเบียบวิธีการบวกคีย์แต่ละรอบ (AddRoundKey) จะมีความหมายดังนี้
a – คือข้อความที่เข้ามาเพื่อจะเข้ากระบวนการบวกคีย์แต่ละรอบ (AddRoundKey)
k – คือคีย์ที่ใช้ในรอบนั้นๆเพื่อใช้ในการบวกคีย์แต่ละรอบ (AddRoundKey)

b – คือผลลัพธ์ที่ได้จากการบวกคีย์แต่ละรอบ (AddRoundKey)

ลำดับของการทำงานจะเป็นดังนี้

```
Round(State,RoundKey)
{
	ByteSub(State);
	ShiftRow(State);
	MixColumn(State);
	AddRoundKey(State,RoundKey);
}
```

การทำงานในรอบที่ 1 – 9 จะใช้ RoundKey(State, RoundKey) เมื่อสิ้นสุดแต่ละรอบก็จะใช้ KeyExpand(RoundKey, Rcon) ในการหา RoundKey ของรอบถัดไป

```
FinalRound(State,RoundKey)
{
	ByteSub(State) ;
	ShiftRow(State) ;
	AddRoundKey(State,RoundKey);
}
```

การทำงานรอบสุดท้ายจะละการทำ MixColumn(State) แล้วจบการทำงานด้วย AddRoundKey(State)

การถอดรหัสมีขั้นตอนกระบวนการที่ไม่แตกต่างจากการเข้ารหัสเท่าไรนักเพียงแค่เปลี่ยนลำดับการทำงานของแต่ละรอบเท่านั้นดังนี้

```
InvRound(State,RoundKey)
{
	AddRoundKey(State,RoundKey);
	InvMixColumn(State);
	InvShiftRow(State);
	InvByteSub(State);
}
```

เช่นเดียวกับการเข้ารหัส การถอดรหัสในรอบสุดท้ายก็มีความแตกต่างเพียงเล็กน้อย

```
InvRound(State,RoundKey)
{
	AddRoundKey(State,RoundKey);
	InvShiftRow(State);
	InvByteSub(State);
}
```

ฟังก์ชั่นที่ขึ้นต้นด้วย Inv มีการทำงานตรงข้ามกับฟังก์ชั่นที่ไม่มี Inv นำหน้า

ที่กล่าวมาแล้วเป็นขั้นตอนการเข้ารหัสแต่เมื่อนำไปใช้งานจริง จำเป็นต้องมีวิธีการดำเนินการ (Operation) มาเกี่ยวข้อง โดยทั่วไปแล้วการดำเนินการมีอยู่ 5 โหมดด้วยกัน อีบีซี (ECB: Electronic Codebook), ซีบีซี (Cipher Block Chaining), ซีเอฟบี (CFB: Cipher Feedback), โอเอฟบี (OFB: Output Feedback) และ ซีทีอาร์ (CTR: Counter) แต่ละโหมดทำงานต่างกันออกไปซึ่งทำให้มีจุดเด่นและจุดด้อยต่างกัน

อีซีบี (ECB: Electronic Codebook)

เป็นการดำเนินการแบบพื้นฐานที่สุด ทำโดยการแบ่งข้อความที่เข้ามาให้มีขนาดพอเหมาะกับบล็อคไซเฟอร์จากนั้นเข้ารหัสแล้วก็จะได้ผลลัพธ์

กำหนดให้ P เป็นข้อความใหญ่ที่แยกออกเป็นข้อความย่อยๆดังนี้

$$P = \{P_1, P_2, \ldots, P_L\}$$

เมื่อข้อความย่อยเข้ารหัสด้วยฟังก์ชั่น E_k ได้เป็นข้อความที่ถูกเข้ารหัสแล้ว

$$C = \{C_1, C_2, \ldots, C_L\}$$

โดย $C_j = E_k(P_j)$ คือการเข้ารหัสของ P_j โดยใช้คีย์ K

ข้อเสียของการดำเนินการโหมดอีซีบีคือการถูกโจมตีได้ง่าย ผู้ไม่หวังดีสามารถทำการถอดรหัสได้โดยง่ายโดยอาศัยจุดอ่อนจากการสังเกตการเข้ารหัสที่ได้ผลลัพธ์เหมือนเดิมตลอด

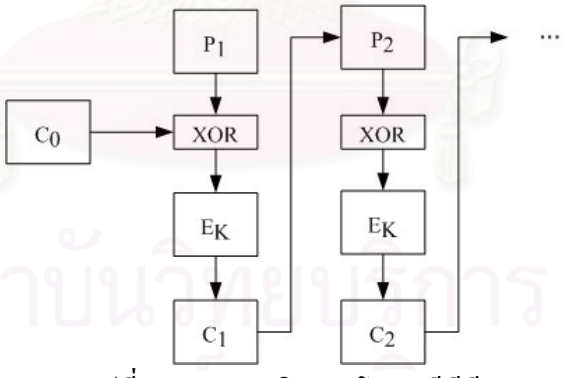
ซีบีซี (CBC: Cipher Block Chaining)

การดำเนินการประเภทนี้ถูกสร้างเพื่อปิดช่องโหว่ที่เกิดขึ้นในอีซีบีด้วยการเข้ารหัสแบบต่อเนื่อง ข้อความขาออกชุดต่อไปที่ถูกนำมาเข้ารหัสมีค่าเป็นเท่าไรก็ขึ้นอยู่กับข้อความในรอบที่แล้วด้วย ส่งผลให้ผู้ไม่หวังดีถอดรหัสได้ยากขึ้นมีวิธีการเข้าและถอดรหัสดังนี้

เข้ารหัส $C_j = E_K(P_j \text{ xor } C_{j-1})$

ถอดรหัส $P_j = D_K(C_j) \text{ xor } C_{j-1}$

แสดงได้เป็นดังรูปที่ 2.5



รูปที่ 2.5 การดำเนินการโหมดซีบีซี

ซีเอฟบี (CFB: Cipher Feedback)

เป็นโหมดการดำเนินการชนิดที่กระทำเป็นแบบสตรีม (Stream) ขนาด k บิต มีหลักการทำงานดังนี้ เริ่มแรกข้อความที่เข้ามาจะถูกแยกออกเป็นข้อความย่อยขนาด 8 บิต (ในที่นี้ขอยกตัวอย่างในกรณี k=8 บิต) จากนั้นกำหนด X_1 ขนาดเท่ากับขนาดของบล็อคไซเฟอร์ที่เราใช้ให้ค่าเริ่มต้น (ในที่นี้ขอใช้เป็น 64 บิต) จะได้ว่า

$$O_j = L_8(E_K(X_j))$$

$$C_j = P_j \text{ xor } O_j$$

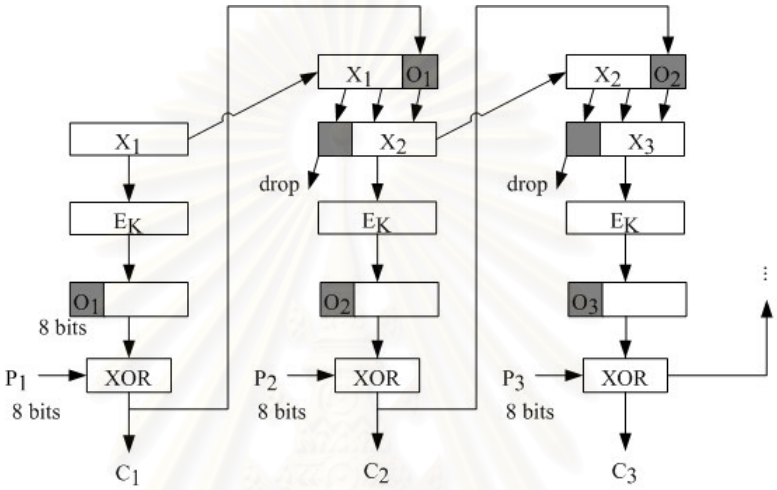
$$X_{j+1} = R_{56}(X_j)||C_j$$

โดยที่ $L_8(X)$ บิตซ้ายสุด 8 บิตของ X

$R_{56}(X)$ บิตขวาสุด 56 บิตของ X

X||Y คือการนำสตริง X มาต่อกับ Y

สามารถสรุปได้เป็นดังรูปที่ 2.6



รูปที่ 2.6 การดำเนินการโหมดซีเอฟบี

ในโหมดนี้การถอดรหัสจะใช้สมการดังนี้

$$P_j = C_j \text{ xor } L_8(E_K(X_j))$$

$$X_{j+1} = R_{56}(X_j)||C_j$$

เห็นได้ว่าไม่มีการใช้ฟังก์ชั่นถอดรหัสมาเกี่ยวข้องเนื่องจากฟังก์ชั่นถอดรหัสใช้เวลาทำงานที่มากกว่าฟังชั่นการเข้ารหัส ข้อเสียของวิธีดำเนินการแบบนี้คือเมื่อข้อมูลที่ผู้รับรับได้เกิดความผิดพลาดขึ้นก็จะทำให้ข้อมูลที่ถูกส่งต่อถัดมาผิดพลาดไปด้วย

โอเอฟบี (OFB: Output Feedback)

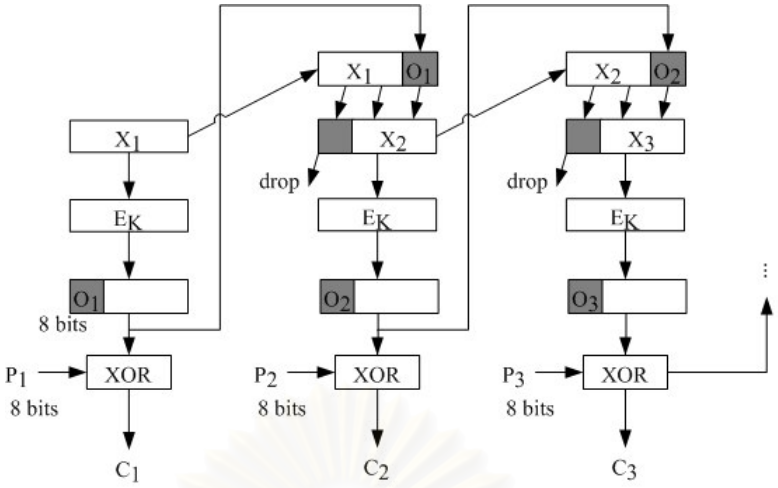
ลักษณะการทำงานคล้ายๆกับซีเอฟบีสามารถแสดงด้วยสมการดังนี้

$$O_j = L_8(E_K(X_j))$$

$$X_{j+1} = R_{56}(X_j)||O_j$$

$$C_j = P_j \text{ xor } O_j$$

สิ่งที่แตกต่างกันคือการต่อสตริงด้วย O_j จุดประสงค์ก็เพื่อเป็นการปิดกั้นไม่ให้ความผิดพลาดในการรับส่งถูกส่งต่อไปในข้อมูลที่ตามมาภายหลัง ดังรูปที่ 2.7



รูปที่ 2.7 การดำเนินการโหมดโอเอฟบี

ข้อเสียของการดำเนินการโหมดนี้คือสามารถถูกโจมตีได้ง่ายเช่นในกรณีที่ผู้ไม่หวังดีรู้ถึง P_j และ C_j ก็รู้ถึง O_j ได้จากสมการ

$$O_j = C_j \text{ xor } P_j$$

เมื่อรู้ O_j ก็สามารถสร้างข้อความเท็จขึ้นมาได้จาก

$$C'_j = P'_j \text{ xor } O_j$$

ซีทีอาร์ (CTR: Counter)

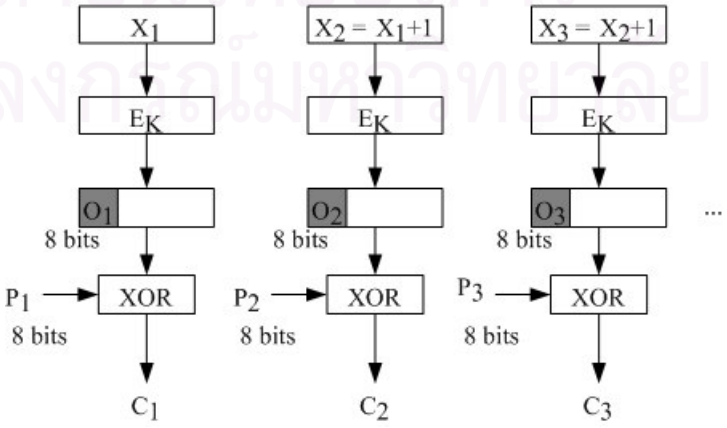
วิธีดำเนินการนี้เหมาะสำหรับกับการนำไปประยุกต์ในการทำงานแบบขนานเพราะข้อความถูกเข้ารหัสไปพร้อมๆกันไม่ต้องรอข้อความก่อนหน้า สมการการดำเนินการเป็นดังนี้

$$X_j = X_{j-1}+1$$

$$O_j = L_8(E_K(X_j))$$

$$C_j = P_j \text{ xor } O_j$$

ดังรูปที่ 2.8



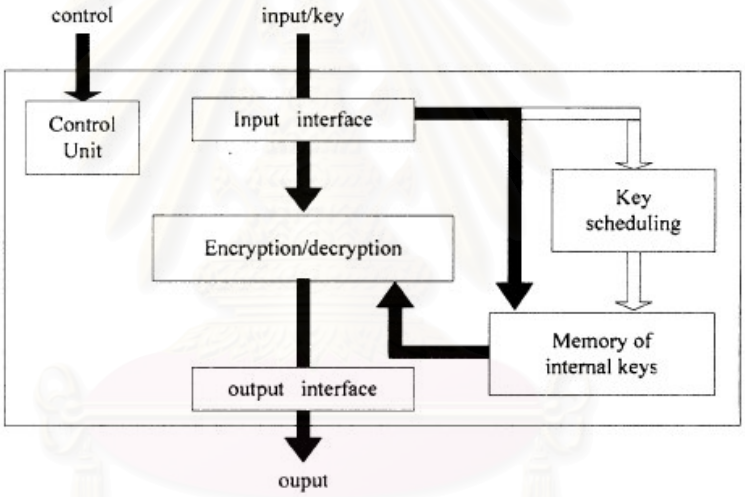
รูปที่ 2.8 การดำเนินการโหมดซีทีอาร์

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่นำมายกตัวอย่างในที่นี้จะเป็นงานเกี่ยวกับด้านการเข้ารหัสบนคอมพิวเตอร์เปลี่ยนโครงแบบได้ (Reconfigurable Computer) และวิธีการแบ่งส่วนของวงจรเพื่อใช้กับคอมพิวเตอร์เปลี่ยนโครงแบบได้ขนาดเล็ก ซึ่งแนวคิดการตัดแบ่งการทำงานของวงจรสามารถนำมาประยุกต์ใช้กับงานวิจัยนี้ได้เป็นอย่างดี

2.2.1 แกนไอพีเออีเอสแบบเปลี่ยนโครงแบบได้ (Reconfigurable AES IP Core)

ในงานวิจัยของซูเจียน (Xu Jian), หลิวหยวนเฟิง (Liu Yuan-feng), ไดจือบิน (Dai Zi Bin) และซุนอี้ (Sun Yi) [10] เป็นแกนไอพี (IP Core) ที่เข้า/ถอดรหัสของเออีเอสโดยเฉพาะสามารถทำงานได้ทั้ง 128/192/256 บิต และด้วยการออกแบบโดยอาศัยการเปลี่ยนโครงแบบทำให้ประหยัดองค์ประกอบตรรกะ (Logic Element) ได้ถึงร้อยละ 43 และประหยัดบิตหน่วยความจำ (Memory Bit) ได้ร้อยละ 17 สถาปัตยกรรมจะเป็นดังรูปที่ 2.9

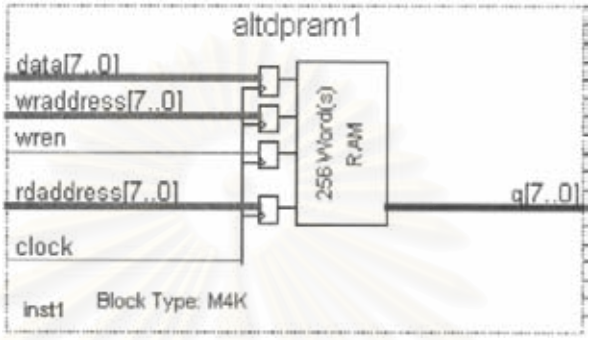


รูปที่ 2.9 แกนไอพีวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้ (Reconfigurable AES IPCore)

จากรูปที่ 2.9

1. เริ่มแรกจะรับค่าอินพุตต์ผ่านทางอินเตอร์เฟสของอินพุตต์
2. ทำการเข้ารหัสและถอดรหัส (Encryption/Decryption)
3. มอดูลจัดกำหนดการของคีย์ (Key Scheduling) ทำการหาค่าของคีย์รอบต่อไปโดยอาศัยค่าจากคีย์รอบปัจจุบัน
4. หน่วยความจำ (Memory) ใช้เก็บค่าของคีย์ที่หาได้
5. หน่วยควบคุม (Control Unit) ใช้ควบคุมวิถีข้อมูล (Data path) ทั้งหมด
6. สุดท้ายเอ้าพุตต์จะออกมาทางอินเตอร์เฟสเอ้าพุตต์ (Output Interface)

เนื่องจากต้องการประหยัดทรัพยากรที่ใช้งานดังนั้นงานวิจัยชิ้นนี้จึงมีการประยุกต์ระบบการเปลี่ยนโครงแบบเข้าไปด้วยที่หน่วยเอสบ็อกซ์ (S-Box unit) โดยเมื่อใดที่ต้องการเข้ารหัสค่าในเอสบ็อกซ์ (S-Box) ก็จะเปลี่ยนไปให้สอดคล้องกับการเข้ารหัสเช่นเดียวกันกับการถอดรหัส นั่นคือเอสบ็อกซ์จะเปลี่ยนตัวเองไปตามการใช้งานในขณะนั้นเพื่อลดทรัพยากรที่สิ้นเปลืองเอสบ็อกซ์ที่ใช้มีลักษณะดังรูปที่ 2.10



รูปที่ 2.10 หน่วยเอสบ็อกซ์ (S-Box Unit) โดยหน่วยความจำแอลพีเอม (lpm_ram)

จากรูปที่ 2.10 คือหน่วยความจำแอลพีเอม (lpm_ram) ของอัลเทร่า (Altera) ซึ่งจะใช้เป็นเอสบ็อกซ์เมื่อมีการเปลี่ยนแปลงการทำงานจากเข้ารหัสเป็นถอดรหัสค่าในหน่วยความจำแอลพีเอมก็จะเปลี่ยนตาม ดังนั้นถ้าเอสบ็อกซ์ (S-Box) ของการถอดรหัสหรือเข้ารหัสยังไม่ได้ใช้ก็จะยังไม่ถูกสร้างลงในวงจรถือว่าไม่เป็นการสิ้นเปลืองทรัพยากร เมื่อวัดประสิทธิภาพแล้วจะได้เป็นดังตารางที่ 2.1

ตารางที่ 2.1 ประสิทธิภาพของวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบเอสบ็อกซ์

IP-CORE support key lengths(bits)	System Clock f_{max}(MHz)	Area	
		Memory bits	LEs
128	70.11	65536	1550
192	65.34	65536	2052
256	61.26	65536	2554
128/192/256	60.34	32768	2612

จากตารางที่ 2.1 จะเห็นได้ว่าการสร้างแกนไอพี (IP-Core) ในแบบสุดท้ายจะประหยัดทรัพยากรมากที่สุดเนื่องจากอาศัยเอสบ็อกซ์ที่เปลี่ยนโครงแบบได้ (Reconfigurable S-Box) ในเวลาใช้งาน

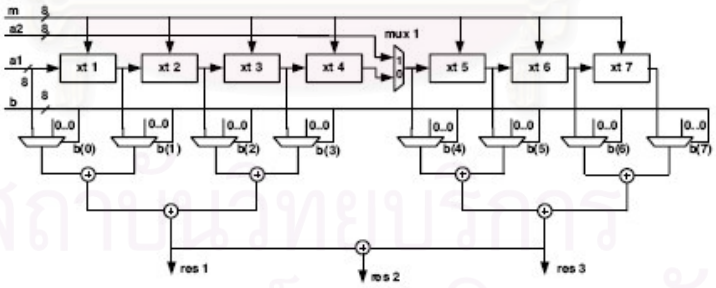
2.2.2 เครื่องเข้าและถอดรหัสแบบเปลี่ยนโครงแบบได้ (A Reconfigurable Encryption/Decryption Engine)

งานวิจัยชิ้นนี้เป็นของธีโล พิโอเทค (Thilo Pionteck), ทอร์สเทน สเทก (Thorsten Staake), โทมัส สตีฟเมเยอร์ (Thomas Stiefmeier), ลูคูซ่า ดี คาบูลีปา (Lukusa D. Kabulepa) และ แมนเฟรด เกรสเนอร์ (Manfred Glesner) [11] เครื่องเข้า/ถอดรหัสที่สามารถเปลี่ยนโครงแบบได้ (Reconfigurable Encryption/Decryption Engine) ตัวนี้ถูกรวมเข้าไปกับหน่วยประมวลผล RISC ขนาด 32 บิต เพื่อทำหน้าที่เป็นหน่วยฟังก์ชัน (Functional Unit) สำหรับหน่วยประมวลผลและมีความสามารถเปลี่ยนโครงแบบ (Reconfigurable) ระหว่างรันไทม์ งานชิ้นนี้ถูกนำไปใช้งานในแมคเลเยอร์ (MAC-layer) ของระบบไวเลสแลนด์ (WLAN)

เครื่อง (Engine) ในส่วนของการเข้ารหัส/ถอดรหัสประกอบด้วยสองส่วนคือ

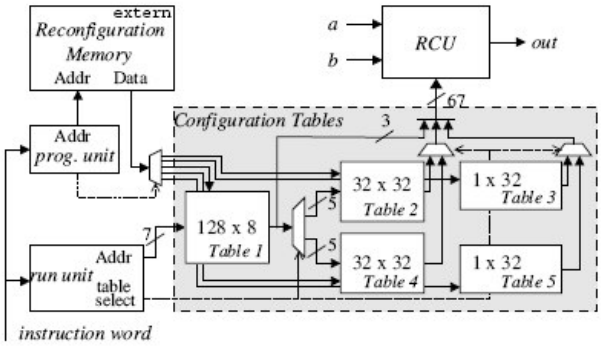
- หน่วยเข้ารหัสเปลี่ยนโครงแบบได้ (Reconfigurable Crypto – Unit)
- ตรรกะควบคุมการเปลี่ยนโครงแบบ (Reconfiguration Control Logic)

หน่วยเข้ารหัสเปลี่ยนโครงแบบได้ (Reconfigurable Crypto – Unit) หรือ อาร์ซียู (RCU) เป็นหน่วยการเข้า/ถอดรหัสแบบที่ออกแบบมาเพื่อให้คุ้มค่ากับทรัพยากรและใช้งานได้หลากหลาย ในด้านการรู้จำความผิดพลาด (Error recognition) และการแก้ไขข้อผิดพลาด (Error Correction) ดังนั้นจึงทำให้ฟังก์ชันผสมหลัก (MixColumns) ต้องมีตัวคูณ (Multiplier) ที่ทำงานบนฟังก์ชัน GF(2^8) อยู่ด้วยนอกจากทำการคูณ (Multiplication) กับค่าคงที่ตามปกติซึ่งไม่จำเป็นต้องพึ่งตัวคูณ (Multiplier) รูปที่ 2.11 แสดงวงจรการคูณ (Multiplication) บน GF(2^8)



รูปที่ 2.11 วงจรการคูณ (Multiplication) บน GF(2^8)

ดังนั้นหน่วยอาร์ซียูจะสามารถรองรับการทำงานได้ทั้งการเข้า/ถอดรหัสและการรู้จำข้อผิดพลาดและการแก้ไขข้อผิดพลาดซึ่งขึ้นอยู่กับว่าตัวหน่วยประมวลจะจัดโครงแบบของหน่วยควบคุมของอาร์ซียูเป็นอย่างไรวงจรตรรกะควบคุมการเปลี่ยนโครงแบบ (Reconfiguration Control Logic) เป็นลักษณะในรูปที่ 2.12



รูปที่ 2.12 ตรรกะควบคุมการเปลี่ยนโครงแบบ (RCL: Reconfiguration Control Logic)

หลักการทำงานเป็นดังนี้

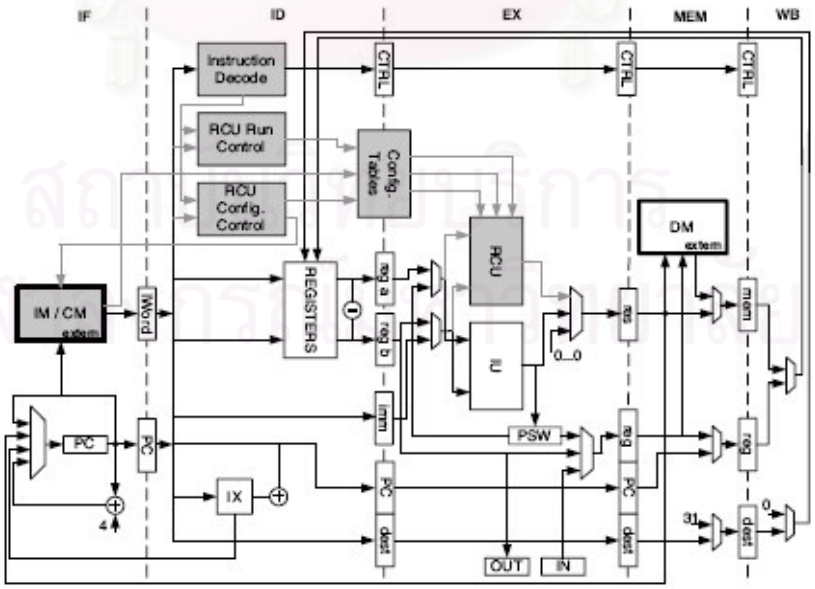
อาร์ซีแอลจะถูกควบคุมโดยเวกเตอร์โครงแบบ (Configuration Vector) ซึ่งประกอบด้วย

$$config = Tab[i][2..0]\&Tab\_x[Tab1[i][7..3]]\&Tab\_y$$

โดย *Tab_x* และ *Tab_y* จะขึ้นกับว่าจะเลือก *Table2/3* หรือ *Table4/5*

- *Table3* กับ *Table5* จะเก็บข้อมูลของโครงแบบเอาไว้ในแบบ 1 x 32 บิต
- *Table2* กับ *Table4* จะเก็บข้อมูลของโครงแบบเอาไว้ในแบบ 32 x 32 บิต
- ส่วน *Table1* จะเก็บข้อมูลลำดับของการเปลี่ยนโครงร่างไว้ในแบบ 128 x 8 บิต

ตารางเหล่านี้สามารถโหลดได้จากหน่วยความจำสำหรับการเปลี่ยนโครงแบบภายนอก (External reconfiguration memory) โดยตัว program unit ในระหว่างที่อาร์ซียูกำลังทำงานอยู่ก็ได้ ส่งผลให้สามารถเปลี่ยนโครงแบบตอนรันไทม์ได้และเมื่อรวมอาร์ซียูและอาร์ซีแอลและหน่วยประมวลผล 32 บิต แบบอาร์ไอเอสซี (RISC) เข้าด้วยกันแล้วจะได้ดังรูปที่ 2.13



รูปที่ 2.13 ภาพรวมของอาร์ไอเอสซี (RISC)

มอดูลที่มีการแรเงาในรูปที่ 2.13 คือมอดูลที่ใส่เพิ่มลงไปในอาร์ไอเอสซี (RISC) 32 บิต เพื่อให้มีความสามารถในการเปลี่ยนโครงแบบได้ เมื่อเพิ่มส่วนของการเปลี่ยนโครงแบบเข้าไปแล้วจำเป็นต้องเพิ่มคำสั่งใหม่เข้าไปอีกสามคำสั่งคือ โหลดเวกเตอร์โครงแบบ (Load Configuration Vector), เริ่มต้นคำสั่งหลายรอบ (Start a Multi-Cycle Command) และประมวลผลคำสั่งรอบเดียว (Execute Single –Cycle Command)

- *คำสั่งหลายรอบ (Multi-Cycle Command)*

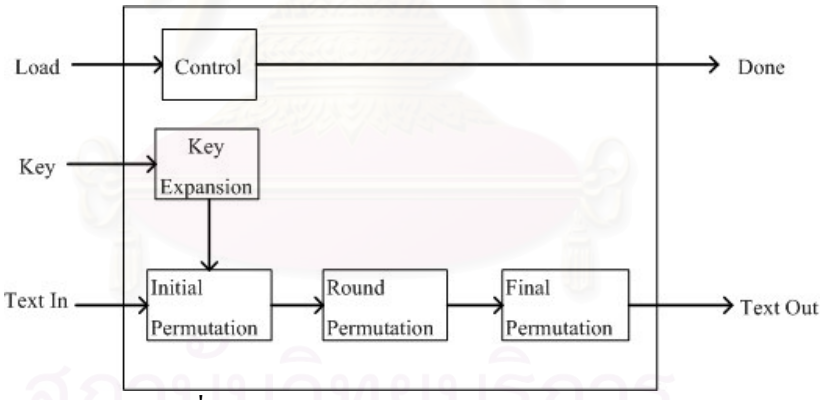
เป็นคำสั่งให้ *Table1* ในรูปที่ 2.12 ตรรกะควบคุมการเปลี่ยนโครงแบบโหลดค่าใหม่เข้ามา

- *คำสั่งรอบเดียว (Single-Cycle Command)*

เป็นคำสั่งที่ใช้กำหนดโครงแบบของอาร์ซียู

2.2.3 แกนไอพีเออีเอส (AES IP Core)

วงจรที่มาเป็นตัวอ้างอิงของงานวิจัยได้มาจากงานวิจัยของรูดอฟ อูซเซลแมน (Rudolf Usselmann) [12] วงจรเข้ารหัสเออีเอสชิ้นนี้ถูกสร้างให้เป็นมาตรฐานเดียวกับที่รินเดลล์ (Rijndael) เสนอและสามารถนำลงไปใช้ในเอฟพีจีเอ (FPGA) ได้และยังคงมีความเร็วในการประมวลผล มีโครงสร้างเป็นดังรูปที่ 2.14 โครงสร้างของเออีเอส



รูปที่ 2.14 โครงสร้างของวงจรเข้ารหัสเออีเอส

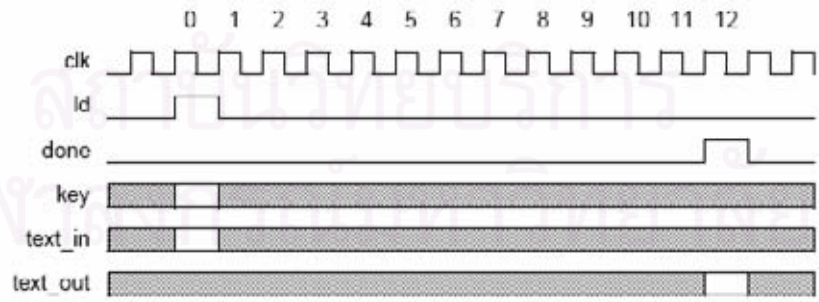
หน้าที่ของแต่ละมอดูลคือ

1. ตัวควบคุม (Control) เป็นหน่วยควบคุมมอดูลที่ใช้ในการปฏิบัติงานให้ทำงานไปอย่างถูกต้อง โดยรับอินพุตต์ (Input) เป็นสัญญาณโหลด (Load)
2. ตัวขยายคีย์ (Key Expansion) เป็นมอดูลที่มีไว้เพื่อหาคีย์ในแต่ละรอบ (Round Key) รวมทั้งหมด 10 รอบ

3. ตัวเรียงสับเปลี่ยนเริ่มแรก (Initial Permutation), ตัวเรียงสับเปลี่ยนแต่ละรอบ (Round Permutation) และตัวเรียงสับเปลี่ยนสุดท้าย (Final Permutation) สามมอดูลนี้ใช้ประมวลผลเพื่อเข้ารหัสข้อความเป็นเอ้าพุตต์ (Output) ออกไป

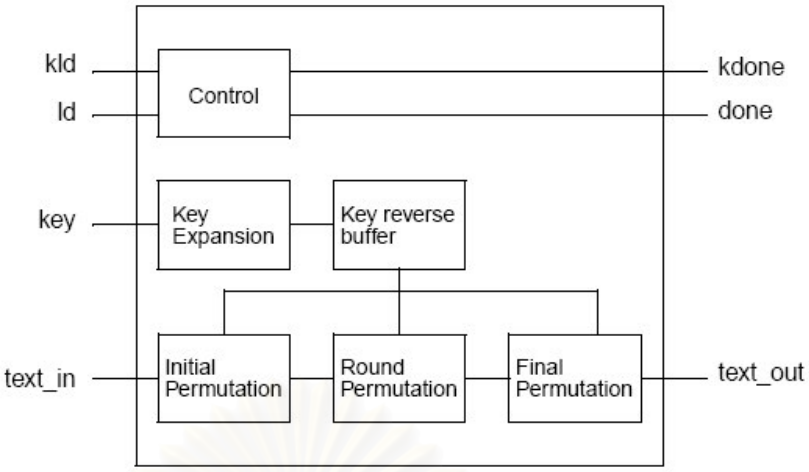
ขั้นตอนการทำงานเป็นดังนี้

1. ระบบจะรับอินพุตต์เข้ามา 3 ทางคือโหลด (Load), คีย์ (Key) และ ข้อความขาเข้า (Text In)
2. เมื่อสัญญาณโหลดเข้ามายังตัวควบคุม ตัวควบคุมจะสั่งให้ตัวขยายคีย์เริ่มทำงาน
3. ตัวขยายคีย์ก็จะนำคีย์ที่ได้จากอินพุตต์เข้าไปประมวลผลหาคีย์ที่ใช้ในแต่ละรอบแล้วส่งต่อไปยังตัวเรียงสับเปลี่ยนเริ่มแรก, ตัวเรียงสับเปลี่ยนแต่ละรอบและตัวเรียงสับเปลี่ยนสุดท้าย
4. ตัวเรียงสับเปลี่ยนเริ่มแรก, ตัวเรียงสับเปลี่ยนแต่ละรอบและตัวเรียงสับเปลี่ยนสุดท้ายจะนำเอาคีย์ที่ได้ไปประมวลผลร่วมกับข้อความขาเข้าที่เข้ามาจนกระทั่งออกไปเป็นข้อความสุดท้ายที่ทำการเข้ารหัสเรียบร้อยแล้วและส่งสัญญาณเสร็จสิ้น (Done) ดังรูปที่ 2.15
5. จากรูปแบบของสัญญาณในรูปที่ 2.15 เห็นได้ว่าการทำงานของวงจรเข้ารหัสเออีเอสที่ยกมาเป็นตัวอย่างสัญญาณนาฬิกาทั้งหมด 12 ครั้งแต่เป็นการเข้ารหัสจริงๆแล้วแค่ 10 ครั้ง หรือ 10 รอบตามที่รินเดลล์ได้เสนอไว้ ส่วนอีกสองสัญญาณนาฬิกาที่เกินมาคือ ส่วนที่เป็นการโหลดค่าของข้อมูลเข้ามาในวงจรเพื่อหาคีย์เริ่มต้นในการเปลี่ยนสถานะของข้อความ และอีกหนึ่งสัญญาณนาฬิกาเพื่อนำค่าเอาท์พุตต์ออกมาแสดงผลรวมทั้งหมดเป็น 12 สัญญาณนาฬิกาพอดี ส่วนสัญญาณเสร็จสิ้น (Done) นั้นจะเป็นตัวบ่งบอกว่าทำงานเสร็จสิ้นแล้วหรือยัง ส่วนสัญญาณแอลดี (ld) เป็นตัวสั่งให้วงจรเริ่มทำงานพร้อมๆกับการโหลดค่าเข้าวงจรด้วย



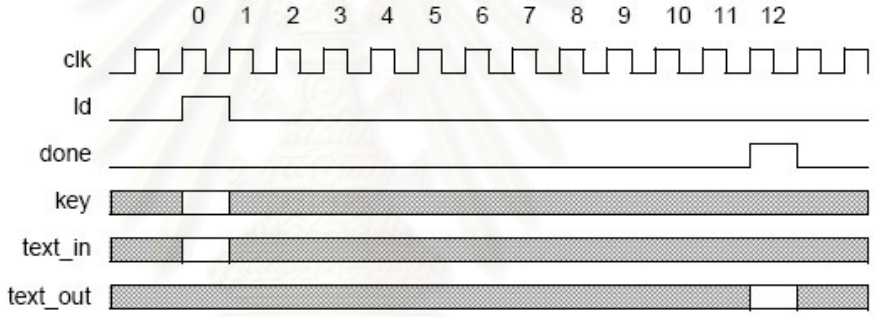
รูปที่ 2.15 รูปแบบของสัญญาณวงจรเข้ารหัสแบบเออีเอสตามมาตรฐานรินเดลล์

วงจรถอดรหัสมีโครงสร้างคล้ายกันกับวงจรเข้ารหัสเพียงแต่มีตัวบัฟเฟอร์สำรองข้อมูลสำหรับหาคีย์ที่ใช้ถอดรหัสแต่ละรอบแล้วเก็บสำรองไว้ใช้ดังแสดงในรูปที่ 2.16



รูปที่ 2.16 โครงสร้างวงจรถอดรหัสเออีเอส

และลักษณะสัญญาณที่ส่งออกมาในการถอดรหัสเป็นไปดังรูปที่ 2.17



รูปที่ 2.17 รูปแบบของสัญญาณวงจรถอดรหัสแบบเออีเอสตามมาตรฐานรินเดลล์

2.2.4 การตัดแบ่งและจัดกำหนดการของกราฟการไหลของข้อมูลสำหรับคอมพิวเตอร์ที่เปลี่ยนโครงแบบได้ (Partitioning and Scheduling Data Flow Graphs for Reconfigurable Computers)

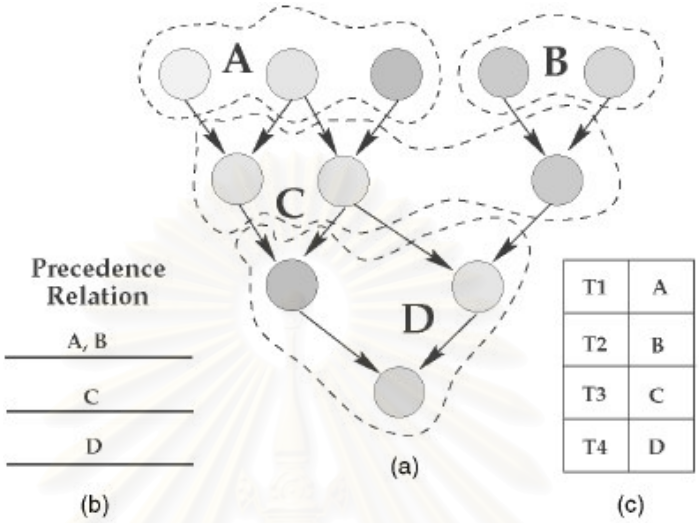
งานวิจัยชิ้นนี้เป็นของคาร์ธิเคยา เอ็ม (Karthikeya M), ไกจาลา เพอร์น่า (Gajjala Purna) และ ดิเนส บาเทีย (Dinesh Bhatia) [13] ได้กล่าวถึงการตัดแบ่ง (Partition) และจัดกำหนดการของกราฟเพื่อจะนำไปใช้งานในคอมพิวเตอร์ที่เปลี่ยนโครงแบบได้ (Reconfigurable Computer) ที่มาของงานวิจัยคือต้องการจะแยกการทำงานของวงจรออกไปเป็นส่วนๆเพื่อให้เล็กพอที่จะสามารถนำวงจรที่ตัดแบ่งแล้วใส่ลงไปในหน่วยเปลี่ยนโครงแบบ (Reconfiguration Unit) ได้ แล้วทำการประมวลผลส่วนของวงจรที่ทำการตัดแบ่งแล้วให้ครบทุกส่วนจนกระทั่งสิ้นสุดการทำงาน

ในการที่จะตัดแบ่งวงจรเราจะใช้กราฟการไหลของข้อมูล (Data Flow Graph: DFG) เป็นตัวแทนของงานที่วงจรรวมทั้งหมดโดยที่

G=(V,E), V คือการปฏิบัติโดยฟังก์ชัน (Function Operation), E คืออินพุตต์ที่มาจากเอ้าพุตต์

นิยาม 1. ค่าอินพุตต์จะ “เสถียร” ก็ต่อเมื่อค่าดังกล่าวเป็นผลลัพธ์ที่ถูกต้องเมื่อทำบนวงจรที่ยังไม่ได้ถูกตัดแบ่ง

นิยาม 2 ความสัมพันธ์พรีซีเด้นซ์ (Precedence-relation) คือความสัมพันธ์ที่ “เสถียร”



รูปที่ 2.18 ความสัมพันธ์พรีซีเด้นซ์ (Precedence relation)

จากรูปที่ 2.18 (a) อินพุตต์ที่ C จะเสถียรก็ต่อเมื่อ A และ B ประมวลผลเสร็จสิ้น มิฉะนั้นจะถือว่า C ยังไม่เสถียร

(b) A กับ B ไม่ขึ้นแก่กันจึงเป็นความสัมพันธ์พรีซีเด้นซ์จะประมวลผลก้อนใดก่อนก็ได้

(c) แสดงลำดับในการประมวลผลที่จะส่งในไปในหน่วยเปลี่ยนโครงแบบ

ระเบียบวิธีการตัดแบ่ง

แบ่งเป็นสองแบบด้วยกันคือ

1.การตัดแบ่งแบบฐานระดับ (Level-Based Partitioning)

เป็นการแบ่งวงจรโดยอาศัยระดับขั้นของการประมวลผลเป็นหลัก เขียนเป็นโค้ดเทียมได้ดังนี้

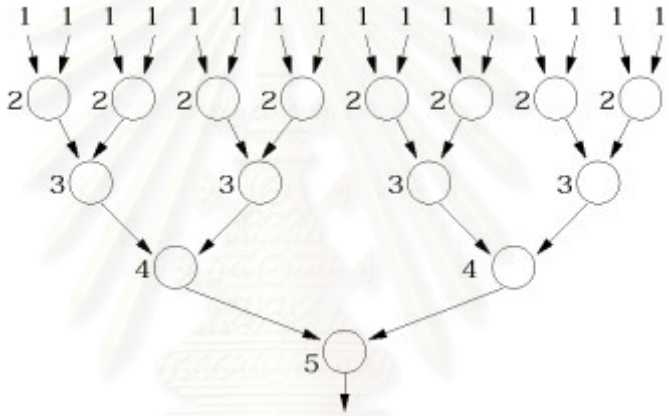
```
For each node vi with Level(vi.) = 0 do
   Partition(vi) <= 0
End For Each
i <= 2
Lev <= 2
Area Filled <= 0

While(Lev _ Max Level)
   For each node vi with Level(vi) = Lev do
      e <= Identify Terminal Edges(vi)
      Total Cost <= Calculate FSMCos(e) . Size(vi) . RCost
      If((Area Filled + Total Cost <= SRPU) then
         Partition(vi) <= i
         Area Filled = Area Filled + Total Cost
```

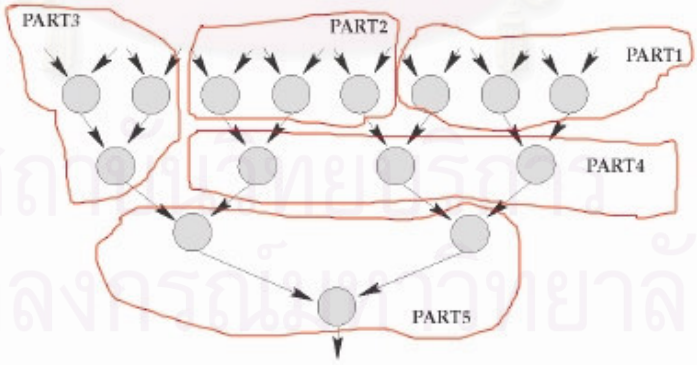
```
      End If
      Else
         i <= i + 1
         Partition.vi. <= i
         Area Filled <= Total Cost
      End Else
   End For each
   Lev <= Lev + 1
End while
```

จากโค้ดเทียมจะสามารถอธิบายเป็นคำพูดได้ดังนี้

เริ่มแรกจะบัพในระดับเดียวกันทยอยใส่ลงไปในหน่วยเปลี่ยนโครงแบบจนเต็มก็ถือว่าจบการตัดแบ่ง 1 ครั้ง จากนั้นทำต่อจนหมดครบทุกบัพในระดับเดียวกันจึงเลื่อนไปทำที่ระดับต่อไป ทำเช่นนี้ไปเรื่อยๆจนกว่าจะตัดแบ่งได้ครบทุกส่วนของวงจรรูปที่ 2.19 จะเป็นรูปก่อนทำการตัดแบ่งและรูปที่ 2.20 เป็นผลลัพธ์หลังจากการตัดแบ่งแล้ว



รูปที่ 2.19 กราฟการไหลของข้อมูลก่อนทำการตัดแบ่ง



รูปที่ 2.20 กราฟการไหลของข้อมูลหลังทำการตัดแบ่ง

ข้อดีของการตัดแบ่งแบบฐานระดับคือการลดความซับซ้อนของการทำงานภายใน หน่วยเปลี่ยนโครงแบบเพราะวงจรที่ถูกนำไปประมวลผลนั้นเป็นอิสระต่อกันจึงประมวลผลได้ง่ายเร็ว แต่ข้อเสียก็คือค่าใช้จ่ายด้านการสื่อสาร (Communication Overhead) เนื่องจากต้องรับอินพุตต์และเอ้าพุตต์ในแต่ละขั้น

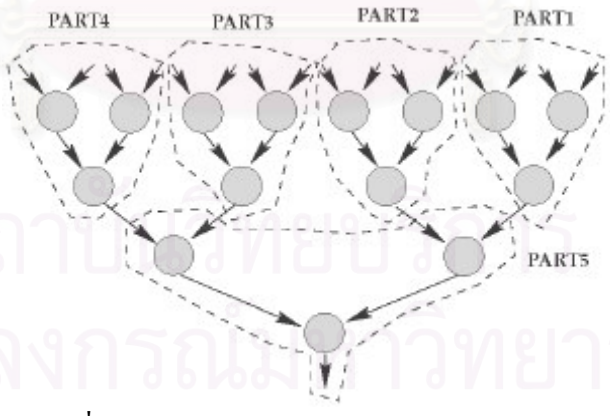
2.การตัดแบ่งแบบฐานกลุ่ม (Clustering-Based Partitioning)

เป็นการแบ่งแบบเน้นการทำงานเป็นหลักเพื่อแก้ปัญหาเรื่องค่าใช้จ่ายการสื่อสาร (Communication Overhead) สามารถเขียนโค้ดเทียมได้ดังนี้

```
ReadyList <= 0
For each node vi in V
       Calculate_successor_id(vi)
End For each
For each node vi in V with Level(vi) = 1
       Partition(vi) <= 1
       ReadyList.update(vi)
End For each

i <= 2
While ReadyList is not empty do
       ui <= ReadyList.front()
       e <= Identify_Terminal_Edges(ui)
       Total_Cost <= Calculate_FSMCost(e) + Size(ui) + RCost
       If((Area_Filled + Total_Cost <= SRPU) then
            Partition(ui) <= i
            Area_Filled <= Area_Filled + Total_Cost
       End If
       Else
            i <= i + 1
            Partition(ui) <= i
            Area_Filled <= Total_Cost
       End Else
       ReadyList.update(ui)
End While
```

ระเบียบวิธีแบ่งกลุ่ม (Clustering Algorithm) จะนำบัพตั้งแต่ระดับแรกลงไปจนถึงระดับที่ลึกที่สุดเท่าที่จะทำได้ใส่ลงไปในหน่วยเปลี่ยนโครงแบบได้ โดยบัพทั้งหมดจะต้องมีความเกี่ยวข้องกัน จากโจทย์ในรูปที่ 2.19 เมื่อแบ่งกลุ่มจะได้ผลลัพธ์เป็นดังรูปที่ 2.21



รูปที่ 2.21 การตัดแบ่งโดยใช้ระเบียบวิธีแบ่งกลุ่ม

วิธีการแบ่งด้วยระเบียบวิธีแบ่งกลุ่มมีข้อดีตรงที่สามารถลด ค่าใช้จ่ายด้านการสื่อสารได้ แต่จะสิ้นเปลืองเวลาทำงานในหน่วยเปลี่ยนโครงแบบ การที่เลือกใช้ระเบียบวิธีใดนั้นก็ขึ้นอยู่กับว่าสถานการณ์ที่เจอเป็นอย่างไรและต้องการลดค่าใช้จ่ายอื่นชนิดใด

จากการแบ่งแบบฐานระดับ (Level-Based) แล้วงานทั้งหมดจะถูกแบ่งย่อยเป็นงานชิ้นเล็กชิ้นน้อยที่ไม่มีความต่อเนื่องกันในเนื้องาน ในขณะที่การตัดแบ่งแบบฐานกลุ่ม (Clustering-Based) จะให้เนื้องานที่มีความต่อเนื่องแต่ในขณะเดียวกันจะใช้เวลาทำงานในแต่ละส่วน (part) มากกว่าเช่นกัน ด้วยเหตุผลข้างต้นนี้ส่งผลให้การตัดแบ่งแบบฐานระดับจำเป็นต้องเสียค่าใช้จ่ายด้านการสื่อสารมากกว่า ส่วนการแบ่งแบบฐานกลุ่มจะเสียเวลาไปกับการทำงานในแต่ละส่วนมากกว่า เมื่อคิดเฉลี่ยเป็นร้อยละโดยใช้ปัญหา MATRIX4, DCT8, BTREE32 และ MEDIAN เป็นเกณฑ์ ผลที่ได้คือการแบ่งแบบฐานระดับใช้เวลาทำงานย่อยน้อยกว่าเฉลี่ยเป็นร้อยละ 40.03 ส่วนการแบ่งแบบฐานกลุ่มใช้ค่าใช้จ่ายในการสื่อสารน้อยกว่าเฉลี่ยเป็นร้อยละ 36.48

บทที่ 3

กราฟแสดงการทำงานและรับส่งข้อมูลของวงจร

การอธิบายถึงข้อแตกต่างระหว่างวงจรเข้ารหัสเออีเอสแบบเดิมกับแบบที่ทำในงานวิจัยชิ้นนี้นั้นเป็นเรื่องที่ค่อนข้างยุ่งยาก กราฟเป็นเครื่องมือที่ในมาช่วยให้การอธิบายถึงข้อแตกต่างดังกล่าวเข้าใจได้ง่ายมากขึ้น การทำงานแสดงด้วยกราฟแสดงการทำงานและกราฟแสดงการรับส่งข้อมูล

3.1 กราฟแสดงการทำงาน

คือกราฟที่ประกอบด้วยจุดยอดและเส้นเชื่อมที่มีทิศทางหรือไม่ก็ได้ ที่ใช้ในการอธิบายพฤติกรรมการเปลี่ยนโครงแบบของวงจรเปลี่ยนโครงแบบได้

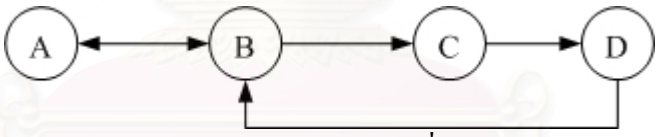
กำหนดให้

จุดยอด แสดงถึง มอดูล

เส้นเชื่อมสองทิศทางที่เชื่อมจุดยอด แสดงถึง ความสามารถในการเปลี่ยนโครงแบบไปมาระหว่างสองมอดูลที่ถูกเชื่อมติดกันได้

เส้นเชื่อมทิศทางเดียวที่เชื่อมจุดยอด แสดงถึง ความสามารถในการเปลี่ยนโครงแบบของมอดูลที่ถูกเชื่อมติดกันในทิศทางเดียวเท่านั้น

กราฟชนิดนี้ใช้อธิบายพฤติกรรมการเปลี่ยนโครงแบบของวงจรชนิดเปลี่ยนโครงแบบได้ กราฟในรูปที่ 3.1 สาธิตการอธิบายการทำงานของวงจรเปลี่ยนโครงแบบได้



รูปที่ 3.1 กราฟแสดงการทำงานของวงจรเปลี่ยนโครงแบบได้วงจรหนึ่ง

กราฟนี้จะแสดงลำดับการทำงานของวงจร รวมทั้งพฤติกรรมการเปลี่ยนโครงแบบของวงจร จากตัวอย่างรูปที่ 3.1 หมายความว่าวงจรดังกล่าวสามารถเปลี่ยนตัวเองไปเป็นวงจรอื่นๆได้ทั้งหมด 4 วงจรซึ่งได้แก่ A, B, C และ D โดย

A เชื่อมกับ B แบบสองทิศทางซึ่งหมายความว่าวงจร A สามารถเปลี่ยนตัวเองไปเป็นวงจร B ได้ทั้งไปและกลับ

B เชื่อมกับ C แบบมีทิศทางไปหา C หมายความว่าวงจร B สามารถเปลี่ยนไปเป็นวงจร C แต่วงจร C ไม่สามารถเปลี่ยนตัวเองกลับมายังวงจร B ได้โดยตรง

C เชื่อมกับ D แบบมีทิศทางไปหา D หมายความว่าวงจร C สามารถเปลี่ยนไปเป็นวงจร D แต่วงจร D ไม่สามารถเปลี่ยนตัวเองกลับมายังวงจร C ได้โดยตรง

D เชื่อมกับ B แบบมีทิศทางไปหา B หมายความว่าวงจร D สามารถเปลี่ยนไปเป็นวงจร B แต่วงจร B ไม่สามารถเปลี่ยนตัวเองกลับมายังวงจร D ได้โดยตรง

จะเห็นได้ว่ากราฟไม่ได้กล่าวถึงการทำงานภายในมอดูลต่างๆรวมทั้งไม่มีการอธิบายการถ่ายโอนข้อมูลของแต่ละมอดูลอีกด้วย ซึ่งจำเป็นต้องอาศัยกราฟแสดงการรับส่งข้อมูลเข้ามาช่วยอธิบาย

3.2 กราฟแสดงการรับส่งข้อมูล

คือกราฟที่ประกอบด้วยจุดยอดสีขาว, จุดยอดสีดำ และเส้นเประที่มีทิศทาง ซึ่งแสดงการรับส่งถ่ายโอนข้อมูลระหว่างมอดูล

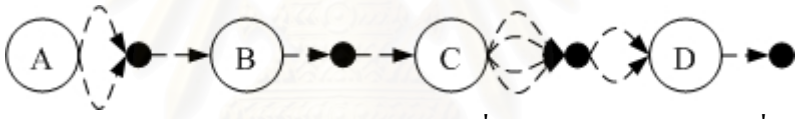
กำหนดให้

จุดยอดสีขาว แสดงถึง มอดูล

จุดยอดสีดำ แสดงถึง หน่วยความจำระหว่างการเปลี่ยนโครงแบบ

เส้นประที่มีทิศทาง แสดงถึง ทิศทางการไหลของข้อมูลจากมอดูลหนึ่งไปมอดูลหนึ่งโดยมีขนาดเท่ากันหมดทุกเส้น

จากที่กล่าวมาแล้ว กราฟชนิดนี้จะอธิบายพฤติกรรมการรับส่งข้อมูลที่เกิดขึ้นระหว่างมอดูล ดังในรูปที่ 3.2



รูปที่ 3.2 กราฟแสดงการรับส่งข้อมูลของวงจรชนิดเปลี่ยนโครงแบบได้วงจรหนึ่งขนาด 16 บิต

จากกราฟแสดงให้เห็นว่า

ที่มอดูล A ไม่จำเป็นต้องโหลดข้อมูลจากหน่วยความจำเข้ามาประมวลผล เมื่อทำงานเสร็จจะส่งข้อมูลขนาด 32 บิตไปเก็บยังหน่วยความจำ ซึ่งจำเป็นจะต้องทำการส่งเข้าไปเก็บสองครั้งจึงเพียงพอ

ที่มอดูล B ต้องทำการโหลดข้อมูลขนาด 16 บิตจากหน่วยความจำเข้ามาประมวลผล เมื่อทำงานเสร็จจะส่งข้อมูลขนาด 16 บิตไปเก็บยังหน่วยความจำ ซึ่งส่งไปเก็บเพียงครั้งเดียวก็เพียงพอ

ที่มอดูล C สามารถทำงานได้โดยโหลดข้อมูล 16 บิตจากหน่วยความจำเข้ามาประมวลผลแล้วส่งผลลัพธ์ขนาด 64 บิตไปเก็บยังหน่วยความจำซึ่งจำเป็นต้องส่งไปเก็บทั้งหมด 4 ครั้งด้วยกัน

ที่มอดูล D จะทำการรับข้อมูลขนาด 32 บิตจากหน่วยความจำมาประมวลผลแล้วส่งผลลัพธ์ขนาด 16 บิตคืนไปยังหน่วยความจำ

กราฟแสดงการรับส่งข้อมูลบ่งบอกถึงจำนวนข้อมูลที่มอดูลต่างๆต้องการในการประมวลผลและจำนวนข้อมูลที่เป็นผลลัพธ์ที่ได้ กราฟชนิดนี้มีประโยชน์ในการวิเคราะห์เวลาที่ใช้ในการจัดการกับ

ข้อมูลที่เข้ามาได้เป็นอย่างดี กราฟทั้งสองแบบจะถูกนำมาใช้อธิบายแนวคิดพื้นฐานตลอดงานวิจัยชิ้นนี้

บทที่ 4

การออกแบบและพัฒนาต้นแบบวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้

4.1 แนวคิดในการออกแบบ

การออกแบบวงจรเข้ารหัสให้เป็นแบบวงจรเปลี่ยนโครงแบบได้นั้นจำเป็นต้องเข้าใจโครงสร้างการทำงานของเออีเอสอย่างละเอียด (กล่าวถึงไปแล้วส่วนหนึ่งในบทที่ 2) จากนั้นจึงนำวิธีการเข้ารหัสดังกล่าวมาวิเคราะห์ว่าส่วนใดควรแยกออกจากกัน ส่วนใดควรอยู่ด้วยกัน ส่วนที่จะแยกออกจากกันได้ควรจะเป็นส่วนที่ไม่ได้ทำงานพร้อมกันเนื่องด้วยสะดวกต่อการออกแบบและการสร้างวงจร จากนั้นจึงทำการปรับปรุงประสิทธิภาพ

4.1.1 แนวคิดทั่วไป

จากจุดประสงค์ที่ต้องการประหยัดเนื้อที่วงจร วงจรเข้ารหัสเปลี่ยนโครงแบบได้จึงควรมีขนาดเล็กกว่าวงจรเข้ารหัสโดยทั่วไป งานวิจัยชิ้นนี้อาศัยข้อได้เปรียบของการเปลี่ยนโครงแบบได้มาลดขนาดของวงจรลง โดยวงจรรวมจะมีขนาดเท่ากับวงจรที่ใหญ่ที่สุดเท่านั้น ถ้าเราทำการตัดแบ่งวงจรให้เล็กลงได้มากเท่าไร ขนาดโดยรวมจะยิ่งเล็กและประหยัดพื้นที่มากขึ้นเท่านั้น การตัดแบ่งวงจรให้แบ่งย่อยลงไปใช่ว่าจะมีแต่ข้อดีเสมอไป ถ้าเราตัดแบ่งย่อยจนเหลือแค่ขนาดของเกตเพียงตัวเดียวแน่นอนว่าวงจรเข้ารหัสแบบนั้นใช้เวลาทำงานนานเกินที่จะรับได้ แต่ในทางกลับกันถ้าทำการตัดแบ่งวงจรไม่เล็กพอ วงจรสุดท้ายที่ออกมาก็จะไม่เล็กและไม่ประหยัดตามที่คาดไว้ นอกจากนี้การตัดแบ่งย่อยวงจรยังจำเป็นต้องคงลักษณะการทำงานที่ถูกต้องไว้ด้วย ก่อนอื่นจึงต้องเข้าใจหลักการทำงานของวงจรเข้ารหัสก่อนแล้วจึงมาวิเคราะห์ว่าจะทำการตัดแบ่งหรือแยกส่วนใดออกจากกัน จากนั้นจึงสรุปผลแล้วปรับปรุงประสิทธิภาพ

เออีเอสมีการทำงานเป็นรอบๆ ทั้งหมด 10 รอบ แต่ละรอบแรกมีขั้นตอนใหญ่ๆสองอย่างนั่นคือ

1. หาคีย์สำหรับเข้ารหัสเพื่อเปลี่ยนสถานะของข้อความ
2. เปลี่ยนสถานะของข้อความ

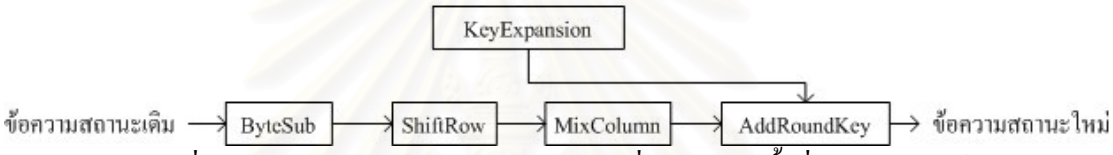
รูปที่ 4.1 แสดงให้เห็นความสัมพันธ์ระหว่างการหาคีย์และการเปลี่ยนสถานะข้อความ

รูปที่ 4.1 ความสัมพันธ์ระหว่างการหาคีย์และการเปลี่ยนสถานะข้อความ

นอกจากนี้กระบวนการเปลี่ยนสถานะข้อความยังแบ่งย่อยไปได้เป็นอีกสี่กระบวนการ การเปลี่ยนสถานะของข้อความประกอบด้วยสี่กระบวนการ นั่นคือ

1. การแทนที่ไบต์ (Byte Substitute หรือ ByteSub) – เป็นกระบวนการเปลี่ยนแปลงสถานะของข้อความโดยการแทนที่ด้วยค่าในเอสบ็อกซ์
2. การเลื่อนแถว (Shift Row หรือ ShiftRow) – เป็นกระบวนการเปลี่ยนแปลงสถานะของข้อความโดยการเลื่อนแถว
3. การผสมหลัก (Mix Column หรือ MixColumn) - เป็นกระบวนการเปลี่ยนสถานะของข้อความโดยการผสมแต่ละหลัก
4. การบวกคีย์แต่ละรอบ (Add Round Key หรือ AddRoundKey) – เป็นกระบวนการเปลี่ยนแปลงสถานะของข้อความโดยการทำเอ็กซ์ออร์กับคีย์ในรอบนั้นๆ

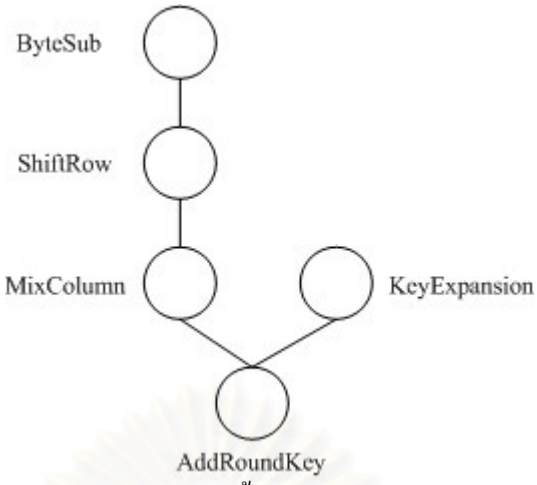
เมื่อนำกระบวนการทั้งสี่มาผนวกเข้ากับกระบวนการหาคีย์ (Key Expansion หรือ KeyExpansion) จะได้ความสัมพันธ์ดังรูปที่ 4.2



รูปที่ 4.2 ความสัมพันธ์ของกระบวนการเปลี่ยนสถานะทั้งสี่และการหาคีย์

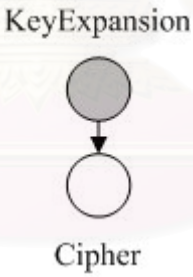
นอกจากการทำงานที่แบ่งเป็นสองขั้นตอนใหญ่ๆดังนี้แล้ว เราอาจสามารถวิเคราะห์การทำงานของวงจรโดยใช้กราฟเข้ามาช่วย ซึ่งการวิเคราะห์วงจรโดยแบบนี้จะพิจารณาจากความเป็นอิสระแก่กันในการทำงานโดยให้ผลลัพธ์คล้ายกับการวิเคราะห์จากความสัมพันธ์ที่ได้กล่าวไปแล้วข้างต้นดังนี้

กราฟที่ใช้อธิบายประกอบด้วยจุดยอดซึ่งคือกระบวนการทำงานและเส้นเชื่อมแบบไร้ทิศทางที่แสดงถึงความขึ้นแก่กันในการทำงาน ซึ่งหมายความว่ากระบวนที่มีเส้นเชื่อมถึงกันไม่สามารถจะทำงานพร้อมๆกันได้จำเป็นต้องรอจนกระบวนการก่อนหน้านั้นทำงานให้เสร็จก่อนแล้วจึงนำผลลัพธ์ไปทำงานของตนเองได้ กราฟเรียงลำดับของเหตุการณ์โดยจุดยอดที่อยู่ปลายด้านบนของเส้นเชื่อมเป็นเหตุการณ์ที่ทำงานก่อนจุดปลายด้านล่าง การเข้ารหัสแบบเออีเอสจะได้กราฟเป็นลักษณะดังนี้



รูปที่ 4.3 กราฟแสดงความขึ้นแก่กันของกระบวนการทำงาน

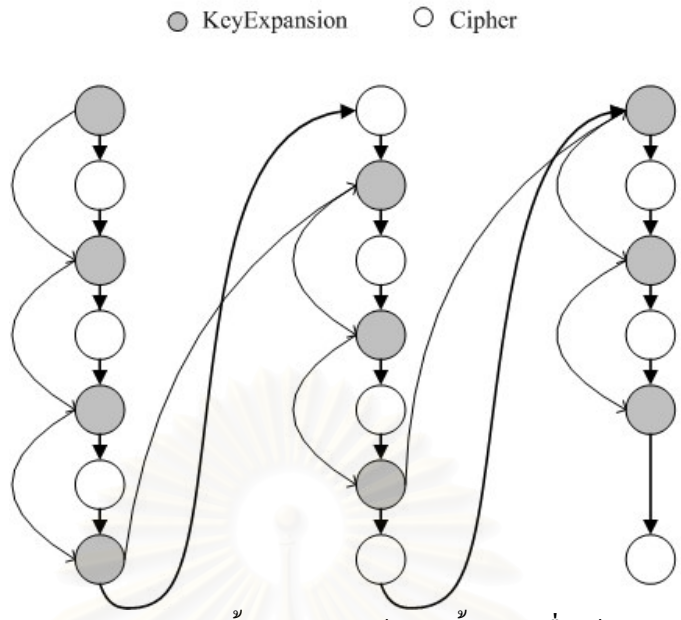
จากรูปที่ 4.3 จะเห็นได้ว่าลำดับการทำงานจะขึ้นแก่กันเป็นดังนี้ แทนที่ไบต์ →เลื่อนแถว → ผสมหลัก → บวกคีย์ในแต่ละรอบ แต่กระบวนการขยายคีย์ไม่ขึ้นกับกระบวนการใดๆเลย นอกจากการบวกคีย์แต่ละรอบจึงเป็นการสนับสนุนว่าเราสามารถทำการเข้ารหัสแบบเออีเอสโดยแยกการทำงานเป็นสองส่วนหลักๆได้คือ ส่วนที่ใช้ในการหาคีย์ และ ส่วนที่ใช้ในการเปลี่ยนสถานะของข้อความ เมื่อนำส่วนย่อยๆจากรูปที่ 4.3 มารวมเป็นก้อนใหญ่ๆสองก้อนจะได้เป็นดังรูปที่ 4.4



รูปที่ 4.4 วงจรเข้ารหัสเมื่อแบ่งเป็นสองก้อนจะทำงานขึ้นแก่กัน

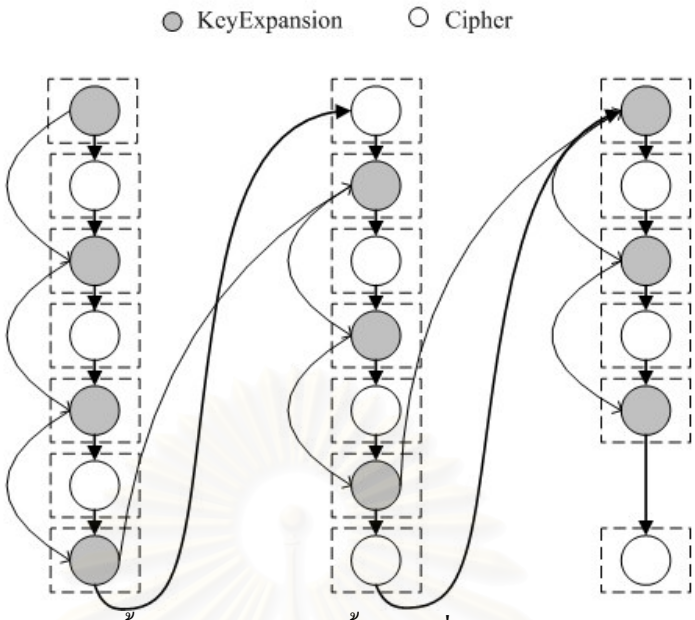
จากรูปที่ 4.4 เมื่อรวมเอาขั้นตอนเล็กๆจากรูปที่ 4.3 เป็นวงจรที่ใหญ่ขึ้น จะได้วงจรสองส่วน ส่วนแรกคือวงจรขยายคีย์ (KeyExpansion) ส่วนที่สองคือไซเฟอร์ (Cipher) ที่ประกอบด้วยส่วนย่อย 4 อย่างคือ การแทนที่ไบต์ (ByteSub), การเลื่อนแถว (ShiftRow), การผสมหลัก (MixColumn) และการบวกคีย์แต่ละรอบ (AddRoundKey)

ถัดมาจะใช้แนวคิดของงานวิจัย [13] ประยุกต์ใช้แบ่งลำดับงานที่นำลงไปทำในเอฟพีจีเอ แนวคิดของงานวิจัยดังกล่าวสามารถนำมาวาดกราฟที่ประกอบด้วยงานสองอย่าง (วงจรขยายคีย์ และ วงจรไซเฟอร์) ในงานวิจัยชิ้นนี้ได้เป็นดังรูปที่ 4.5



รูปที่ 4.5 กราฟแสดงการขั้นตอนการใช้วงจรทั้งหมดเพื่อเข้ารหัสเออีเอส

กราฟในรูปที่ 4.5 เกิดจากการนำกราฟในรูปที่ 4.4 มาต่อกันจนครบเป็นการเข้ารหัสที่สมบูรณ์ สังเกตได้ว่ากราฟดังกล่าวมีส่วนที่เพิ่มเข้ามาจากกราฟเดิมคือเส้นโค้งที่เชื่อมระหว่างการขยายคีย์เข้าไว้ด้วยกันซึ่งหมายความว่าก่อนที่จะใช้วงจรขยายคีย์ในรอบหน้าเราจำเป็นต้องใช้การขยายคีย์ในรอบปัจจุบันให้เสร็จก่อนจึงค่อยทำการขยายคีย์ในครั้งต่อไปได้ สาเหตุเกิดมาจากการขยายคีย์ถูกออกแบบมาให้มีหน้าที่เพื่อหาคีย์แบบรอบต่อรอบ ไม่ได้หาค่าของคีย์ทั้งหมดแล้วค่อยใช้งานดังนั้นทุกๆรอบของการใช้วงจรไซเฟอร์เพื่อเปลี่ยนสถานะข้อความจำเป็นต้องมีการใช้การขยายคีย์ก่อนเสมอ และจำเป็นต้องใช้การขยายคีย์เรียงลำดับกันไปตามแต่ละรอบด้วยตามลำดับของเส้นเชื่อม ขั้นตอนต่อไปคือการกำหนดว่าลำดับว่างานใดจะลงไปทำงานในวงจรเอฟพีจีเอก่อนหลัง ดังเช่นในงานวิจัยของ [13] การทำงานในขั้นตอนต่อไปนี้จำเป็นต้องทราบข้อกำหนดของเอฟพีจีเอที่ใช้และตัวโครงสร้างของวงจรทั้งสองก่อน ข้อกำหนดที่มีผลต่อวงจรนี้อย่างมากคือขนาดของวงจร เนื่องจากวงจรการขยายคีย์และวงจรไซเฟอร์ไม่สามารถจะใส่ลงไปในเอฟพีจีเอพร้อมๆกันได้ในคราวเดียว ดังนั้นการตัดแบ่งกราฟเพื่อใช้ทำงานจึงไม่จำเป็นต้องพิจารณาว่าจะใส่ลงไปกี่วงจรเพียงแค่พิจารณาว่าจะใส่อะไรก่อนหลังเท่านั้น การตัดแบ่งกราฟมีสองลักษณะคือการแบ่งแบบฐานระดับ (Level-Based partition) และการแบ่งแบบฐานกลุ่ม (Clustering-Based partition) ตามที่ได้กล่าวมาแล้วว่าเอฟพีจีเอที่ใช้ไม่สามารถทำงานทั้งสองวงจรไปพร้อมๆกันได้อีกทั้งงานที่ทำเป็นงานที่ขึ้นแก่กันทั้งหมดไม่มีงานที่เป็นอิสระต่อกันเลย การแบ่งกราฟจะออกมามีหน้าตาที่เหมือนกันจะเป็นไปดังรูปที่ 4.6



รูปที่ 4.6 กราฟแสดงการขั้นตอนการใช้วงจรทั้งหมดเพื่อเข้ารหัสเออีเอสหลังจากตัดแบ่งแล้ว

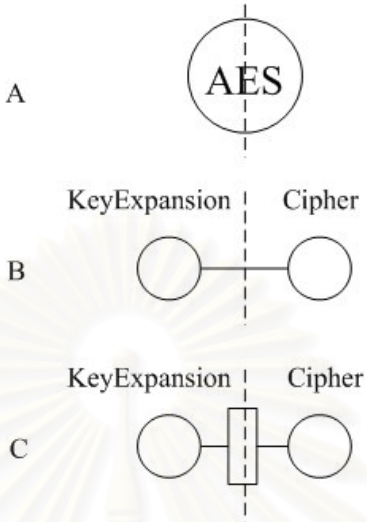
เมื่อตัดแบ่งเสร็จสิ้นแล้วผลลัพธ์ระหว่างการตัดด้วยฐานระดับและฐานกลุ่มจะไม่ต่างกัน ดังที่แสดงในรูปที่ 4.6 โดยสรุปแล้ววงจรการขยายคีย์และวงจรไซเฟอร์จะผลัดกันลงไปทำงานในเอฟพีจีเอ

4.1.2 การแยกย่อยวงจร

ในหัวข้อนี้เราจะมุ่งประเด็นไปที่เรื่องการแบ่งแยกย่อยวงจรเข้ารหัสเออีเอสออกเป็นสองส่วนคือการขยายคีย์และไซเฟอร์ เริ่มแรกของการตัดแบ่งต้องวิเคราะห์ส่วนประกอบและหาส่วนที่สามารถนำมาสร้างเป็นรอยต่อระหว่างวงจรได้ เราสามารถสร้างรอยต่อให้เกิดขึ้นภายในวงจรได้จากการใส่ตัวหน่วยความจำเข้าไประหว่างการส่งถ่ายข้อมูลระหว่างกัน คือเมื่อวงจรแรกทำงานมาถึงรอยต่อก็ให้นำข้อมูลมาเก็บไว้ที่หน่วยความจำแล้วหน่วยความจำจะทำการส่งต่อข้อมูลจากรอยต่อนั้นไปยังวงจรหลังได้อธิบายให้เข้าใจได้ตามรูปที่ 4.7

จากรูปที่ 4.7 ทั้งสามรูปย่อยแสดงถึงขั้นตอนการแบ่งย่อยวงจรเข้ารหัสเออีเอสทั้งวงจรให้แยกออกเป็นสองวงจรย่อยการขยายคีย์และไซเฟอร์ในรูปที่ 4.7 A คือการกำหนดเส้นที่จะทำให้เกิดรอยแยกขึ้นซึ่งจะเป็นเส้นที่ใดก็ได้ เส้นดังกล่าวจะเป็นเส้นที่แบ่งระหว่างการหาคีย์และการเปลี่ยนสถานะของข้อความ เมื่อได้เส้นดังกล่าวมาแล้วเราจะทำการดึงออกจากกันตามรูปที่ 4.7 B นั่นคือจับแบ่งเป็นสองวงจรแต่ยังมีบัสเชื่อมระหว่างกันอยู่เพื่อตรวจสอบก่อนว่าวงจรที่เราทำการแบ่งออกมาดังกล่าวยังทำงานได้ถูกต้องหรือไม่ เมื่อตรวจสอบเสร็จสิ้นเราจะดำเนินการในขั้นต่อไปคือตามรูปที่ 4.7 C เพื่อให้วงจรทั้งสองตัดขาดกันอย่างสมบูรณ์สำหรับการทำงานในลักษณะเปลี่ยนโครงแบบได้จำเป็นต้องใส่หน่วยความจำคั่นลงไปเพราะสองวงจรย่อยนี้ไม่ได้ทำงานพร้อมกันตามที่ได้กล่าวมา

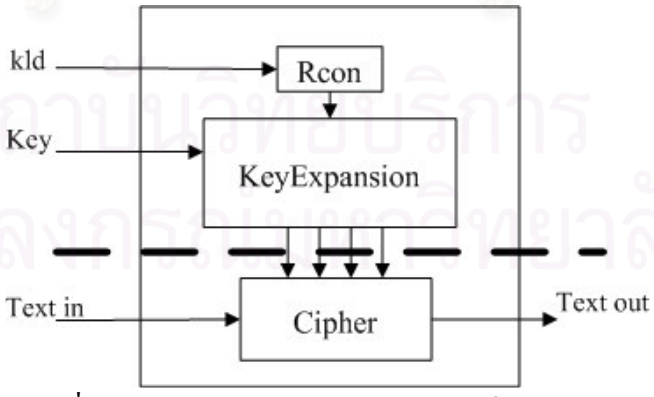
ข้างต้นจึงจำเป็นต้องมีหน่วยความจำมารับแล้วส่งค่าข้อมูลที่เป็นผลลัพธ์ให้กับวงจรถัดไปที่จะมาทำงานต่อ



รูปที่ 4.7 วิธีการตัดแบ่งย่อยวงจรเข้ารหัสให้เป็นสองวงจรย่อย

นอกจากหน่วยความจำจะใช้สำหรับแบ่งแยกย่อยงานแล้วยังมีหน้าที่อีกคือเชื่อมแต่ละงานเข้าด้วยกันดังนั้นถ้าวงจรต้องการเปลี่ยนโครงแบบจากวงจรไซเฟอร์ไปเป็นวงจรการขยายคีย์ก็จำเป็นต้องใส่หน่วยความจำไปด้วย

เมื่อได้ตำแหน่งที่ต้องการแบ่งแล้วก็จะมาพิจารณาถึงวิธีในการแบ่งวงจร การแบ่งวงจรให้ได้อย่างที่วางไว้จำเป็นต้องเข้าใจการทำงานและองค์ประกอบภายใน การทำงานของวงจรเข้ารหัสที่นำมาใช้งานอยู่ในงานวิจัย [12] วงจรดังกล่าวเมื่อกำหนดเส้นแบ่งแล้วจะมีโครงสร้างดังรูปที่ 4.8

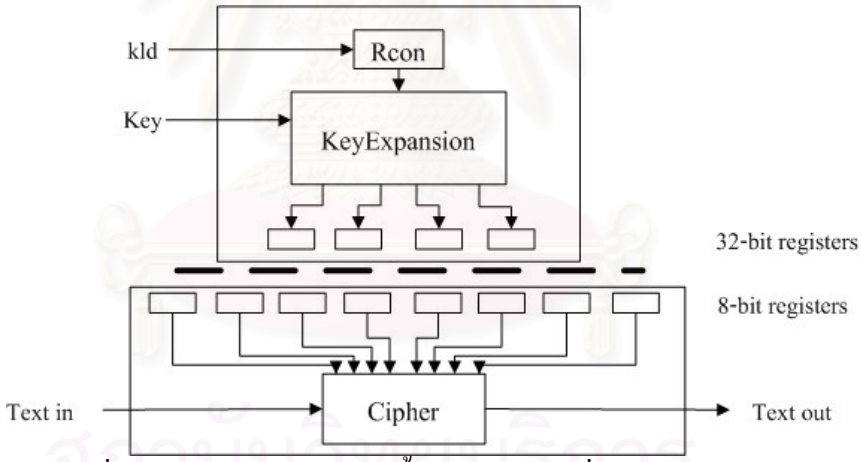


รูปที่ 4.8 เส้นแบ่งย่อยวงจรเข้ารหัสออกเป็นสองส่วน

จากรูปที่ 4.8 ส่วนของวงจรเข้ารหัสที่ทำงานขยายคีย์จะติดต่อกับวงจรไซเฟอร์โดยผ่านทางบัสขนาด 32 บิตจำนวน 4 เส้น บัสนี้เป็นบัสทางเดียวทำการส่งข้อมูลจากวงจรขยายคีย์ไปยังวงจรไซเฟอร์ ข้อมูลชุดขนาด 128 บิตที่ส่งไป เป็นข้อมูลของคีย์ในรอบนั้นๆที่จะนำไปเปลี่ยนสถานะของ

ข้อความวงจรไซเฟอร์จะดึงเอาข้อมูลจากข้อความขาเข้า (Text in) มาประมวลผลแล้วส่งออกไปเป็นผลลัพธ์สุดท้าย ดังนั้นถ้าเราจะทำการแยกวงจรขยายคีย์ออกจากวงจรไซเฟอร์ผลกระทบจะตกไปอยู่ที่การส่งค่าคีย์ตัวใหม่ไปยังวงจรไซเฟอร์เท่านั้น ตรงบริเวณรอยตัดแก้ไขโดยการใส่รีจิสเตอร์เข้าไป ซึ่งจะกล่าวต่อไป ส่วนการทำงานภายในวงจรมีการเปลี่ยนแปลงเล็กน้อยคือส่วนหน่วยควบคุมที่ต้องบังคับให้วงจรนำค่าไปเก็บที่รีจิสเตอร์ด้วย

เมื่อมีการวางแนวแบ่งวงจรแล้วดังในรูปที่ 4.8 เราจะเริ่มการแบ่งย่อยวงจรให้ออกเป็นสองส่วนโดยอาศัยรอยตัดที่ได้ เมื่อได้รอยแบ่งที่แน่นอนแล้วจะทำใส่หน่วยความจำรีจิสเตอร์ลงไปในวงจรย่อยแต่ละส่วนเพื่อเก็บค่าที่ต้องไปใช้ในวงจรถัดไป (ในการทำงานจริงเมื่อรีจิสเตอร์ได้ผลลัพธ์ครบถูกต้องแล้วจะส่งผลดังกล่าวให้รีจิสเตอร์แบงก์เป็นตัวส่งต่อข้อมูลอีกที) การพิจารณาว่ารีจิสเตอร์ที่ใส่เข้าไปควรมีขนาดและจำนวนเท่าไรขึ้นอยู่กับความสะดวกในการเก็บข้อมูลและความสะดวกในการนำข้อมูลไปใช้ต่อซึ่งรีจิสเตอร์ที่เชื่อมต่ออาจไม่จำเป็นต้องมีขนาดและจำนวนที่เท่ากันก็ได้ รีจิสเตอร์ที่อยู่ในส่วนของวงจรขยายคีย์มีขนาด 32 บิต จำนวน 4 ตัวตามจำนวนบัสที่ออกจากวงจรขยายคีย์นอกจากนั้นยังต้องใส่รีจิสเตอร์เพิ่มลงไปในวงจรไซเฟอร์ด้วยเช่นกัน โดยเป็นรีจิสเตอร์ขนาด 8 บิต จำนวน 16 ตัว เมื่อใส่รีจิสเตอร์ลงไปจนครบแล้วจะได้เป็นดังรูปที่ 4.9



รูปที่ 4.9 โครงสร้างของวงจรทั้งสองหลังจากเพิ่มเติมรีจิสเตอร์แล้ว

จากรูปที่ 4.9 วงจรที่อยู่ด้านบนจะเป็นวงจรที่เรียกว่าวงจรขยายคีย์และวงจรที่อยู่ด้านล่างจะถูกเรียกว่าวงจรไซเฟอร์ส่วนลักษณะการทำงานและพฤติกรรมของวงจรทั้งคู่จะขอกล่าวในหัวข้อถัดไป

4.1.3 แพลตฟอร์ม

หลังจากที่เรากล่าวถึงแนวคิดในการแยกกันทำงานแล้ว ครานี้เราจะมาดูในด้านแพลตฟอร์ม (Platform) การออกแบบจะเริ่มด้วยการวางแนวคิดให้แพลตฟอร์มจะต้องสามารถรองรับการคำนวณได้หลายประเภทและมีความยืดหยุ่นพอสมควรเนื่องจากแพลตฟอร์มดังกล่าวนอกจากเป็นวงจรเปลี่ยนโครงแบบได้เพื่อการเข้ารหัสเออีเอสแล้วยังต้องสามารถรองรับการคำนวณในแบบอื่นในอนาคตได้อีกด้วย ซึ่งความยืดหยุ่นนี่เองทำให้การออกแบบแพลตฟอร์มจำเป็นต้องออกมาให้ง่ายที่สุดเท่าที่จะทำได้ ส่วนสำคัญอย่างหนึ่งที่จะบกพร่องไม่ได้นั่นคือแพลตฟอร์มจำเป็นต้องรองรับการส่งต่อข้อมูลจากหน่วยประมวลผลหนึ่งไปยังอีกหน่วยประมวลผลหนึ่ง ดังนั้นแพลตฟอร์มในงานวิจัยชิ้นนี้จึงต้องมีหน่วยความจำรีจิสเตอร์ (Register) เพื่อนำมาใช้ในการเก็บค่าผลลัพธ์ชั่วคราวที่ออกมาระหว่างการประมวลผล การประมวลผลงานใดๆ ของแพลตฟอร์มนี้จะเป็นไปในลักษณะที่ทำงานทีละส่วนยกตัวอย่างเช่น

กำหนดให้

W เป็นงานชิ้นใหญ่ที่ประกอบด้วยงานย่อย $w_0, w_1, w_2 \ldots w_n$ หรือเขียนแทนได้ดังนี้

$$W = \{w_0, w_1, w_2 \ldots w_n\}$$

P(x, y) คือแพลตฟอร์มที่รับงาน x และค่าอินพุตต์ y แล้วคืนค่ามาเป็น r หรือ

$$r = P(x, y)$$

เริ่มแรกแพลตฟอร์มจะรับงานและค่าอินพุตต์เริ่มต้นซึ่งผลลัพธ์จะเป็นดังนี้

$$r_0 = P(w_0, \text{initialValue})$$

แต่สิ่งที่เราต้องการคือผลลัพธ์สุดท้ายที่ออกมาจาก w_n ดังนั้นเราจึงต้องนำ r_0 ไปประมวลผลต่อจนกว่าจะได้ r_n ดังนี้

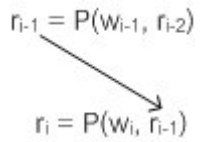
$$r_1 = P(w_1, r_0)$$

$$r_2 = P(w_2, r_1)$$

$$\vdots$$

$$r_n = P(w_n, r_{n-1})$$

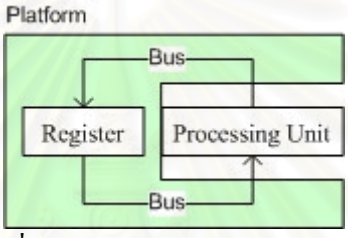
จะเห็นได้ว่าทุกขั้นตอนที่จะมีผลลัพธ์ชั่วคราว (r_i) ออกมาด้วยซึ่งรีจิสเตอร์ในแพลตฟอร์มมีหน้าที่เก็บค่าพวกนี้ไว้เพื่อส่งต่อให้กับหน่วยประมวลที่จะมาทำงานต่อไปนั่นเองสามารถแสดงไปเป็นดังรูปที่ 4.10



รูปที่ 4.10 หน้าที่ของรีจิสเตอร์บนแพลตฟอร์ม

จากรูปที่ 4.10 เส้นตรงนั้นแสดงถึงหน้าที่ของรีจิสเตอร์ที่ทำการรับค่าผลลัพธ์ชั่วคราวจากแพลตฟอร์มที่มีการทำงานแบบ w_{i-1} แล้วทำการส่งค่าไปยังแพลตฟอร์มเดิมแต่มีงานเป็น w_i เพื่อประมวลผลต่อไป

การต่อเชื่อมแพลตฟอร์มกับหน่วยประมวลผลใช้บัส (Bus) และเพื่อเป็นการง่ายที่สุดบัสที่เข้าไปหาหน่วยประมวลผลทางเดียวและออกจากหน่วยประมวลผลทางเดียว เนื่องด้วยตัวหน่วยประมวลผลหนึ่งๆนั้นมีการรับขนาดของอินพุตต์ที่เข้าไปไม่เท่ากัน ส่วนถ้าหน่วยประมวลผลใดต้องการอินพุตต์ที่มีขนาดมากกว่าขนาดของบัสแพลตฟอร์มจะทำการส่งค่าอินพุตต์เพิ่มเติมเข้าไปให้ยังหน่วยประมวลผลนั้นๆ สรุปเป็นดังรูปที่ 4.11 หน่วยประมวลผลหรือ Processing Unit คืองานย่อยๆ $w_0, w_1, w_2 \ldots w_n$ แต่ละงานผลัดเปลี่ยนกันเข้าไปทำงานกับแพลตฟอร์ม



รูปที่ 4.11 แสดงช่องทางเข้าออกที่ใช้ในการติดต่อกันระหว่างแพลตฟอร์มกับหน่วยประมวลผล

4.1.4 หน่วยควบคุม

แนวคิดสุดท้ายของแพลตฟอร์มที่จะกล่าวถึงคือหน่วยควบคุม (Control Unit) ในระบบทั่วไปนั้นหน่วยควบคุมมีหน้าที่ทำการจัดการการไหลของข้อมูล หน่วยควบคุมจะทำการควบคุมการทำงานภายในระบบผ่านสัญญาณควบคุม (Control Signal) เมื่อส่วนที่ปฏิบัติงานได้รับสัญญาณดังกล่าวก็เริ่มทำงาน เมื่อทำงานเสร็จแล้วหน่วยควบคุมก็จะสั่งงานต่อไปยังส่วนอื่นๆอีกเพื่อทำงานจนเสร็จสมบูรณ์ ซึ่งลักษณะของสัญญาณที่ออกไปรอบถัดไปนั้นจะต่างกับรอบที่แล้วขึ้นกับว่าสถานะ (state) ของหน่วยควบคุมสัญญาณในขณะนั้นเป็นอย่างไร สามารถอธิบายได้ดังนี้

กำหนดให้

$C(x)$ – สัญญาณควบคุมที่สถานะ x

s^x_i – สัญญาณควบคุมในสถานะที่ x ที่ส่งไปยังหน่วยปฏิบัติการที่ i

จะได้ว่า

$$C(x) = \{s^x_0, s^x_1, s^x_2, s^x_3 \ldots s^x_n\}$$

เมื่อสถานะทั้งหมดของระบบเป็น

$$x \rightarrow y \rightarrow z$$

แปลเป็นสัญญาณควบคุมจะเป็น

ที่สถานะ x

$$C(x) = \{s^x_0, s^x_1, s^x_2, s^x_3 \ldots s^x_n\}$$

ที่สถานะ y

$$C(y) = \{s^y_0, s^y_1, s^y_2, s^y_3 \ldots s^y_n\}$$

ที่สถานะ z

$$C(z) = \{s^z_0, s^z_1, s^z_2, s^z_3 \ldots s^z_n\}$$

จะเห็นได้ว่าสัญญาณควบคุมทั้งสามชุดออกมาจากฟังก์ชัน C(x) เพียงตัวเดียวเนื่องจากระบบทั่วไปนั้นมีหน่วยควบคุมการประมวลผลเพียงหนึ่งตัวและมีวิถีข้อมูล (Data Path) แค่แบบเดียว ดังนั้นแค่เปลี่ยนลักษณะการให้สัญญาณก็สามารถเปลี่ยนการทำงานหน่วยปฏิบัติงานในวิถีข้อมูลได้ แต่ระบบที่เรากำลังออกแบบนี้ไม่ได้มีรูปแบบการทำงานอย่างที่ได้ยกตัวอย่างไปข้างต้น ระบบที่สามารถเปลี่ยนโครงแบบได้จะมีข้อแตกต่างหลักๆคือ มีวิถีข้อมูลมากกว่าหนึ่งแบบที่ใช้ในเวลาต่างๆกันไป เมื่อวิถีข้อมูลเปลี่ยนไปตามเวลาหน่วยควบคุมการประมวลผลในลักษณะเดิมๆจำเป็นต้องมีการปรับเปลี่ยนไปด้วยซึ่งเป็นไปได้สองลักษณะคือ

1. สร้างหน่วยควบคุมโดยให้รู้จักทุกๆวิถีข้อมูลที่มีอยู่

   ให้การทำงานหนึ่งมีวิถีข้อมูลทั้งหมดเป็น

   $$D = \{d_0, d_1, d_2 \ldots d_n\}$$

   กำหนดให้วิถีข้อมูล d_i มีเซตของสัญญาณควบคุมเป็นดังนี้

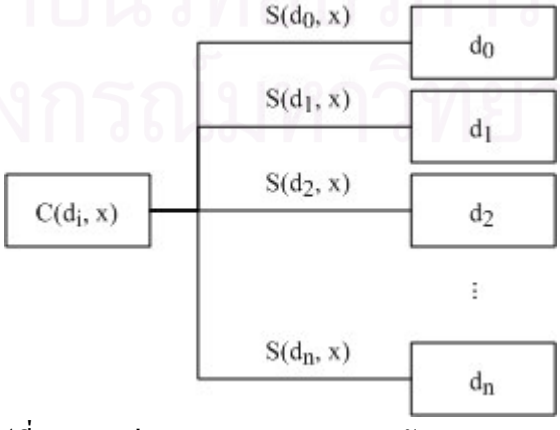
   $$S(d_i, x) = \{s^i_0(x), s^i_1(x), s^i_2(x) \ldots s^i_n(x)\}$$

   โดยที่ x คือสถานะในขณะใดๆ

   ดังนั้นหน่วยควบคุมการประมวลผลมีการทำงานดังนี้

   $$C(x, d_i) = \{S(d_0, x), S(d_1, x), S(d_3, x) \ldots S(d_n, x)\}$$

แนวคิดนี้เมื่อนำมาใช้จะได้ลักษณะของหน่วยควบคุมเป็นไปดังรูปที่ 4.12



รูปที่ 4.12 หน่วยควบคุมแบบรวมทุกสัญญาณควบคุม

2. สร้างหน่วยประมวลขึ้นมาหลายๆอันแล้วเลือกใช้อันที่เหมาะสมกับวิถีข้อมูลในขณะนั้น นั่นคือ

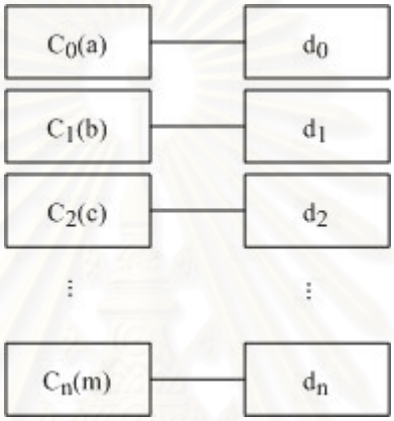
ให้การทำงานหนึ่งมีวิถีข้อมูลทั้งหมดเป็น

$$D = \{d_0, d_1, d_2 \ldots d_n\}$$

หน่วยควบคุมที่เราต้องใช้คือ

$$C = \{C_0(a), C_1(b), C_2(c) \ldots C_n(m)\}$$

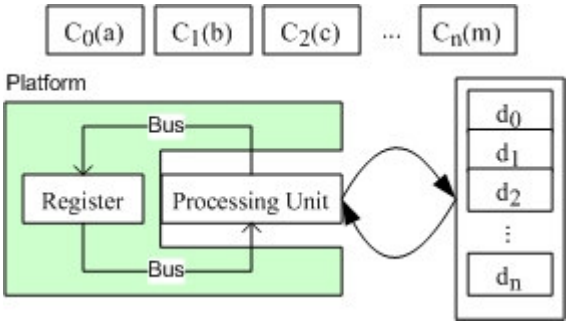
หน่วยควบคุมแบบนี้สามารถแสดงไว้ในรูปที่ 4.13



รูปที่ 4.13 หน่วยควบคุมที่รวมเอาทุกหน่วยควบคุมเล็กๆไว้ด้วยกัน

หน่วยควบคุมแบบแรกนั้นทำงานซับซ้อนกว่าซึ่งจะทำให้มีขนาดใหญ่กว่าปกติส่งผลให้ประสิทธิภาพไม่ดี สาเหตุเพราะต้องรู้จักวิถีข้อมูลทั้งหมดที่มีในระบบ นอกจากนี้ถ้าหากต้องการจะเพิ่มวิถีข้อมูลใหม่เข้าไปจะเป็นเรื่องที่ยุ่งยากเพราะต้องมาทำการแก้ไขลักษณะของสัญญาณให้สอดคล้องกับของเดิมที่มีอยู่ ส่วนในหน่วยควบคุมในแบบที่สองนั้นมีข้อดีตรงที่ทุกตัวมีขนาดไม่ใหญ่เกินไปทุกตัวจะมีขนาดพอๆกันถ้าลักษณะของสัญญาณมีจำนวนพอๆกัน ข้อดีคือเมื่อมีการใส่วิถีข้อมูลใหม่เข้าไปก็สามารถสร้างหน่วยควบคุมตัวใหม่ใส่ตามเข้าไปได้โดยไม่จำเป็นต้องไปแก้หน่วยควบคุมเดิมแต่อย่างใด ส่วนข้อเสียคือหน่วยควบคุมแต่ละตัวไม่รู้จักกัน

เมื่อนำหน่วยควบคุมมารวมเข้ากับแพลตฟอร์มที่วางแนวคิดไว้ก่อนหน้านี้แล้วจะได้ระบบโดยรวมเป็นดังรูปที่ 4.14



รูปที่ 4.14 ระบบเมื่อมีการรวมแพลตฟอร์มกับหน่วยควบคุมการประมวลผลเข้าไว้ด้วยกัน

จากรูปที่ 4.14 จะเห็นได้ว่าตรงหน่วยประมวลผลจะมีการสลับกันเข้ามาทำงานของวิถีข้อมูลซึ่งก็แล้วแต่ว่าในขณะนั้นระบบมีความจำเป็นต้องใช้วิถีข้อมูลแบบใดและย่อมจะต้องสอดคล้องกันกับหน่วยควบคุมด้วย แต่แนวคิดนี้ยังติดปัญหาว่าหน่วยควบคุมแต่ละตัวไม่รู้จักกัน ปัญหาที่ตามมาคือวิถีข้อมูลที่ถูกควบคุมอยู่นั้นทำงานไม่สอดคล้องกัน

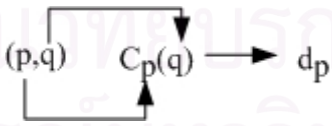
หน่วยควบคุมหลายๆตัวสามารถทำงานไปพร้อมๆกันได้ถ้ามีตัวคอยจัดลำดับการทำงานของแต่ละหน่วยให้ไปพร้อมๆกันเช่น การใช้หน่วยควบคุมอีกระดับมาจัดการ หน้าที่คือสั่งว่า ณ ขณะนี้หน่วยประมวลผลใดต้องทำงานหน่วยใดหยุดพัก ตัวที่นำมาใช้ควบคุมควรจะไม่ทำให้ระบบซับซ้อนไปกว่าการใช้หน่วยควบคุมในแบบแรกที่กล่าวไปข้างต้น ตัวควบคุมนำมาใช้ในงานนี้จึงเป็นแค่ตัวชี้ (Pointer) ธรรมดาซึ่งไปยังหน่วยควบคุมการประมวลผลที่ทำงานเท่านั้น ซึ่งการทำงานจะเป็นดังนี้

กำหนดให้ (p,q) – คู่ลำดับที่แสดงถึงการเรียกใช้ หน่วยควบคุม p ที่สถานะ q

$C_i(x)$ – หน่วยควบคุม i สถานะ x

$S_j(y)$ – เซตของสัญญาณควบคุม j สถานะ y

d_k – วิถีข้อมูล k



รูปที่ 4.15 วิธีการทำงานของหน่วยควบคุมการประมวลผลเมื่อรวมเข้ากับวิถีข้อมูล

จากรูปที่ 4.15 จะเห็นได้ว่า คู่ลำดับ (p,q) ทำหน้าที่เป็นตัวชี้เพื่อบอกให้หน่วยควบคุมทำงาน เราจึงสามารถใช้ประโยชน์ของคู่ลำดับนี้ในการจัดลำดับการทำงานให้หน่วยควบคุมทุกตัวทำงานไปพร้อมๆกันได้ เริ่มจากว่าต้องการให้หน่วยควบคุมการประมวลผลตัวใดทำอะไรในเวลาที่เท่าไร มาเขียนเรียงกันเป็นเซตของคู่ลำดับเพื่อไล่ลำดับการทำงานไปจนถึงกระบวนการสุดท้าย สามารถสรุปเป็นรูปให้ดูได้เข้าใจง่ายตามในรูปที่ 4.16 และรูปที่ 4.17

กำหนดให้ P – เซตของคู่ลำดับที่

$C_i(x)$ – หน่วยควบคุม i สถานะ x

d_k – วิถีข้อมูล k

$$P \longrightarrow C_p(q) \longrightarrow d_p$$

รูปที่ 4.16 เซตของตัวชี้จะบ่งบอกการทำงานของหน่วยควบคุมการประมวลผลทั้งหมด

จากรูปที่ 4.17 เห็นได้ว่าค่า (p_0,a_0) ที่ออกมาจากเซต P จะไปชี้ยังหน่วยควบคุม $C_0(k)$ ซึ่งในที่นี้ตัวชี้จะทำการส่งพารามิเตอร์สถานะ a_0 ไปด้วย ซึ่งหน่วยควบคุมจะนำเอาสถานะดังกล่าวไปสร้างสัญญาณควบคุมที่เหมาะกับสถานะนั้นๆ จากรูปที่ 4.17 ที่สถานะ a_0 หน่วยควบคุม $C_0(k)$ จะเป็น $C_0(a_0)$ นั่นคือให้ $k = a_0$ จากนั้นหน่วยประมวลผลส่งสัญญาณควบคุมที่ตัวมันเองเก็บไว้ตามที่ได้นิยามไว้ ($C(x) = \{s^x_0, s^x_1, s^x_2, s^x_3 \ldots s^x_n\}$) สัญญาณเหล่านี้จะส่งตรงไปยังวิถีข้อมูลแล้วไปยังหน่วยปฏิบัติการต่างๆตามที่ได้รับมอบหมาย วิถีข้อมูลที่ได้รับสัญญาณไปก็จะทำงานจนเสร็จแล้วจากนั้นก็จะรับสัญญาณชุดใหม่ที่ส่งมาอีกซึ่งจะทำไปจนกว่าหน่วยประมวลผลจะเปลี่ยนโครงแบบตัวเองเป็นวิถีข้อมูลแบบอื่น อย่างไรก็ตามการจะเปลี่ยนโครงแบบก็ยังจำเป็นต้องอาศัยตัวชี้คอยกำหนดว่าขณะใดควรเป็นวิถีข้อมูลแบบใด ดังนั้นจึงเห็นได้ว่าตัวชี้ที่เรามีจึงเก็บเป็นเซต เซตของคู่ลำดับจะว่าถัดไปจากคู่ลำดับปัจจุบันเมื่อทำงานเสร็จแล้วจะมีใช้คู่ลำดับใดมาทำงานต่อและพฤติกรรมของหน่วยประมวลผลรวมทั้งสัญญาณควบคุมจะเป็นอย่างไร เมื่อระบบทำงานมาจนถึงคู่ลำดับสุดท้ายของเซตก็จะหยุดทำงานและถือว่างานเสร็จสิ้นโดยสมบูรณ์

ถ้ามีการเพิ่มเติมวิถีข้อมูลแบบใหม่เข้าไปในระบบ ก็สามารถทำได้โดยนำวิถีข้อมูลมาใส่ใน D และนำหน่วยควบคุมการประมวลผลมาใส่ใน C จากนั้นจึงนิยามพฤติกรรมของเซต P ที่เก็บตัวชี้

4.1.5 แนวคิดรวบยอด

หลังจากที่เราได้แนวคิดทั้งด้านการเข้ารหัสแบบทีละส่วนรวมทั้งแพลตฟอร์มและหน่วยควบคุมแล้วนำมาประยุกต์เข้าด้วยกัน ในที่นี้การเข้ารหัสแบบเออีเอสจะแบ่งเป็นสองส่วนตามที่ได้กล่าวไปข้างต้นคือ

1.ส่วนที่ใช้หาคีย์

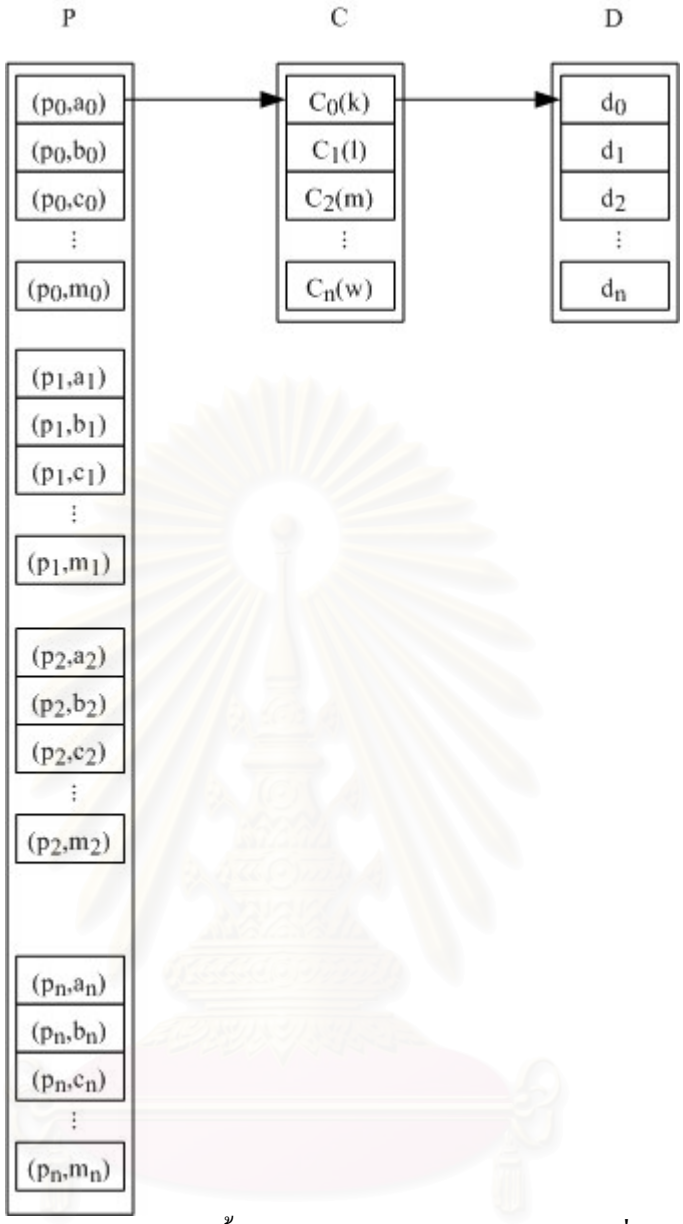
2.ส่วนที่ใช้เปลี่ยนสถานะ

และในส่วนแพลตฟอร์มจะประกอบด้วย

1.หน่วยความจำรีจิสเตอร์

2.หน่วยประมวลผลที่เปลี่ยนโครงแบบได้

3.หน่วยควบคุม (รวมไปถึง ตัวชี้และหน่วยควบคุมย่อย)

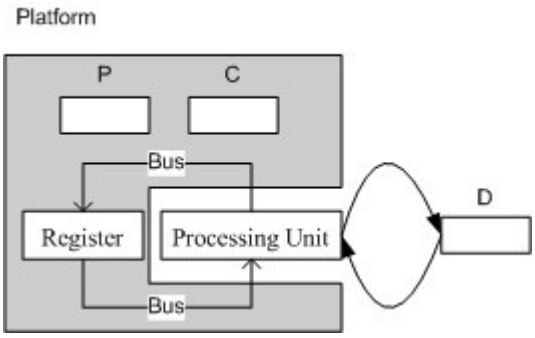


รูปที่ 4.17 ตัวชี้แรกคือ (p_0,a_0) ทำการชี้และส่งสถานะไปยังหน่วยควบคุมที่ต้องการ จากนั้นหน่วยควบคุมการประมวลผลจะส่งต่อไปยังวิถีข้อมูลที่เป็นเป้าหมาย

เมื่อนำทั้งสองส่วนมารวมกันรวมทั้งทำให้เฉพาะเจาะจงกับงานมากขึ้นจะได้ส่วนประกอบทั้งหมดเป็นดังนี้

1. หน่วยความจำรีจิสเตอร์
2. หน่วยประมวลที่เปลี่ยนโครงแบบไปเป็น ส่วนหาคีย์ และ ส่วนที่เป็นสถานะข้อความ
3. หน่วยควบคุมการประมวลผลทำงานสอดคล้องกับ ส่วนหาคีย์ และ ส่วนที่เปลี่ยนสถานะข้อความ

สามารถแสดงได้เป็นดังรูปที่ 4.18



รูปที่ 4.18 แนวคิดโดยรวมของต้นแบบวงจรเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้

4.2 พฤติกรรมของวงจรต้นแบบเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้

ในหัวข้อที่แล้วได้กล่าวถึงแนวคิดในการออกแบบและวางระบบและทำงานของวงจรไปแล้ว มาในหัวข้อนี้จะขออธิบายถึงวิธีการทำงานหรือพฤติกรรมการทำงานของวงจร พฤติกรรมการทำงานสามารถอธิบายไปได้ในสามลักษณะ

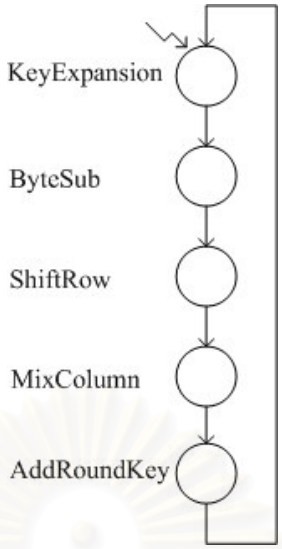
1. แผนภาพสถานะ (State Diagram)
2. กราฟแสดงการทำงาน (บทที่ 3)
3. กราฟแสดงการรับส่งข้อมูล (บทที่ 3)

ทั้งสามอย่างจะอธิบายลักษณะของวงจรที่แตกต่างกันโดยนำเสนอคนละแง่มุม แผนภาพสถานะนำเสนอด้านวิธีการดำเนินไปของสถานะในวงจรว่าขณะนั้นวงจรทำงานอะไรอย่างไรติดต่อกับอะไรอยู่ กราฟแสดงการทำงานจะกล่าวถึงลักษณะภาพรวมของการใช้วิถีข้อมูลที่มีในระบบ กราฟแสดงการรับส่งข้อมูลใช้แสดงการถ่ายโอนข้อมูลระหว่างวิถีข้อมูลที่เกี่ยวข้องกัน

4.2.1 แผนภาพสถานะ

วงจรเข้ารหัสในแบบทั่วไปมีแผนภูมิสถานะเพียงอันเดียวเนื่องจากตัววงจรมีเพียงอันเดียว พฤติกรรมการทำงานจึงต่อเนื่องกันไป แผนภูมิสถานะอันเดียวจึงเพียงพอต่อการอธิบายพฤติกรรมวงจรเข้ารหัสเออีเอสแบบปกติมีแผนภูมิสถานะดังรูปที่ 4.19

การทำงานจะเริ่มต้นที่สถานะขยายคีย์โดยทำการรับค่าของคีย์ตั้งต้นเข้าไปแล้วคืนค่าคีย์ที่ใช้ในรอบหน้ามาให้ จากนั้นวงจรเปลี่ยนสถานะตัวเองไปเป็นสถานะ ByteSub เพื่อเปลี่ยนสถานะของข้อความที่เข้ามาในสถานะนี้ เมื่อ ByteSub เปลี่ยนสถานะเป็นที่เรียบร้อยแล้วก็จะเป็นสถานะ ShiftRow เพื่อเปลี่ยนสถานะของข้อความที่ได้รับมาจากสถานะ ByteSub สถานะต่อไปคือ MixColumn เช่นเคย สถานะนี้ก็จะรับค่ามาจากสถานะ ShiftRow เพื่อหาค่าสถานะใหม่ของข้อความแล้วจึงค่อยส่งต่อแต่มีส่วนที่ต่างออกไปคือสถานะ MixColumn นี้ต้องรับค่าคีย์ตั้งต้นเข้ามาด้วย เมื่อหาค่าสถานะใหม่ออกมาได้จะส่งต่อไปยังสถานะการบวกคีย์แต่ละรอบซึ่งมีโครงสร้างการทำงานเหมือน MixColumn คือรับค่าคีย์และข้อความ สิ่งที่ได้มาที่สถานะนี้คือข้อความที่เปลี่ยนสถานะจนสมบูรณ์แล้วหนึ่งรอบ เนื่องจากขั้นตอนวิธี (Algorithm) ของเออีเอสนั้นต้องเปลี่ยน



รูปที่ 4.19 วงจรเข้ารหัสในแบบทั่วไปทำงานจากสถานะขยายคีย์จนถึงการบวกคีย์แต่ละรอบ

สถานะจนเสร็จสมบูรณ์ทั้งหมดสิบรอบด้วยกัน ดังนั้นเพื่อหาคำตอบสุดท้ายจึงจำเป็นต้องประมวลผลตามกระบวนการข้างต้นอีกเก้ารอบด้วย

กำหนดให้ k_0 – ค่าของคีย์เริ่มต้น

t_0 – ข้อความเริ่มต้น

KE(a) – สถานะ KeyExpansion

BS(b) – สถานะ ByteSub

SR(c) – สถานะ ShiftRow

MC(d,e) – สถานะ MixColumn

AR(f,g) – สถานะการบวกคีย์แต่ละรอบ

สิ่งที่เกิดขึ้นที่วงจรขยายคีย์

$$k_1 = KE(k_0)$$

หลังจากนั้น ByteSub และ ShiftRow จะได้ค่าเป็น

$$SrBs = SR(BS(t_0))$$

แล้วทำ MixColumn จะเป็น

$$MxSrBs = MC(SrBs,t_0)$$

สุดท้ายการเปลี่ยนสถานะหนึ่งครั้งจะสมบูรณ์โดยการทำการบวกคีย์แต่ละรอบ

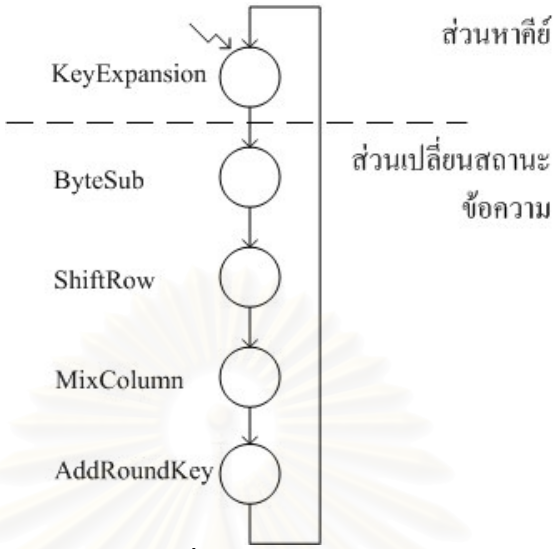
$$t_1 = AR(MxSrBs,t_0)$$

เมื่อจบรอบแรกค่าที่เป็นผลลัพธ์ชั่วคราวนี้คือ

k_1 – คีย์ที่ใช้ในรอบต่อไป

t_1 – ข้อความเริ่มต้นของรอบต่อไป

ค่าที่เราได้ทั้งสองค่านี้จะถูกนำไปใช้ในรอบต่อๆไปจนกว่าจะครบสิบรอบ

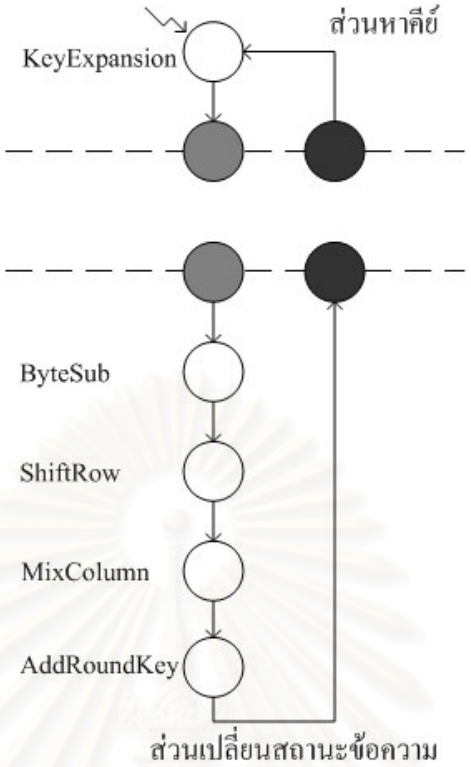


รูปที่ 4.20 วงจรเข้ารหัสชนิดเปลี่ยนโครงแบบได้มีการแบ่งออกเป็นงานสองงาน

หลังจากที่ได้ศึกษาหลักการทำงานของวงจรเข้ารหัสแบบปกติแล้วในตอนนี้เราจะกล่าวถึงลักษณะการเข้ารหัสในงานวิจัยชิ้นนี้ ตามที่ได้กล่าวมาในแนวคิดซึ่งแบ่งวิธีการเข้ารหัสออกเป็นงานย่อยๆสองงานคือ 1. การหาคีย์ 2. การเปลี่ยนสถานะของข้อความ งานสองอย่างนี้เมื่อนำมาพิจารณาลงไปถึงโครงสร้างสถานะด้านในจะพบได้ว่าไม่ต่างกับวงจรปกติที่มีห้าสถานะในการทำงานเพียงแต่จุดที่ต่างออกไปคือจุดเชื่อมต่อหรือส่งต่อการทำงานจากงานการหาคีย์ไปยังงานเปลี่ยนสถานะของข้อความหรือถ้าจะพูดให้ในอีกมุมมองคือการส่งการทำงานจากสถานะ KeyExpansion ไปยัง ByteSub นั่นเองดังรูปที่ 4.20

จากรูปที่ 4.20 สถานะการทำงานโดยทั่วไปแล้วจะเหมือนกับการเข้ารหัสในกรณีปกติแตกต่างกันเพียงตรงจุดเชื่อมต่อที่กล่าวไปแล้ว ตรงจุดเชื่อมต่อมีการทำงานที่ต้องติดต่อกับหน่วยความจำรีจิสเตอร์ก่อนแล้วจึงค่อยทำงานต่อในสถานะถัดไป ด้วยเหตุนี้แผนภาพสถานะจึงต้องถูกแยกออกเป็นสองส่วนโดยเพิ่มสถานะ(สถานะสีเทากับสถานะสีดำ)ที่ติดต่อกับหน่วยความจำรีจิสเตอร์ลงไปตรงบริเวณเส้นประในรูปที่ 4.20 สถานะที่ใส่เข้าไปเป็นสถานะที่ใช้เชื่อมต่อระหว่างงานการหาคีย์และงานการเปลี่ยนสถานะของข้อความหรือสถานะขยายคีย์ (KeyExpansion) และสถานะแทนที่ไบต์ (ByteSub) ดังแสดงในรูปที่ 4.21

จากรูปที่ 4.21 แสดงให้เห็นว่าหลังจากที่แยกงานออกจากกันแล้วจะมีเพียงสถานะที่ติดต่อกับหน่วยความจำเท่านั้นที่เป็นหนทางในการเชื่อมต่อกันเรายังสามารถใช้สถานะที่ติดต่อกับหน่วยความจำรีจิสเตอร์นี้ในการเปลี่ยนโครงแบบของวิถีข้อมูลไปในคราวเดียวกันได้ด้วย



รูปที่ 4.21 แผนภาพสถานะของการเข้ารหัสเออีเอสแบบเปลี่ยนโครงได้หลังจากตัดแบ่งสถานะแล้ว

สามารถแสดงเป็นสมการจะได้ตามนี้

กำหนดให้ k_0 – ค่าของคีย์เริ่มต้น

t_0 – ข้อความเริ่มต้น

register – หน่วยความจำรีจิสเตอร์

KE(a) – สถานะ KeyExpansion

BS(b) – สถานะ ByteSub

SR(c) – สถานะ ShiftRow

MC(d,e) – สถานะ MixColumn

AR(f,g) – สถานะการบวกคีย์แต่ละรอบ

$R_0(x)$ – สถานะติดต่อหน่วยความจำรีจิสเตอร์ที่ 0

$R_1(x)$ – สถานะติดต่อหน่วยความจำรีจิสเตอร์ที่ 1

เมื่อตอนเริ่มต้นที่สถานะขยายคีย์จะเป็นดังนี้

$$k1 = KE(k0)$$

ส่งค่าคีย์ที่หาได้ไปเก็บที่หน่วยความจำรีจิสเตอร์

$$register = R_0(k1)$$

หลังจากนั้น ByteSub และ ShiftRow จะได้ค่าเป็น

$$SrBs = SR(BS(t_0))$$

แล้วทำ MixColumn จะเป็น

$$MxSrBs = MC(SrBs,t_0)$$

เปลี่ยนสถานะโดยการทำการบวกคีย์แต่ละรอบ

$$t_1 = AR(MxSrBs,t_0)$$

ส่งไปเก็บที่รีจิสเตอร์

$$register = R1(t1)$$

ส่วนในรอบถัดไปนั้นก็นำค่า k1 ที่เคยเก็บลงในรีจิสเตอร์ในสถานะก่อนหน้านี้มาใช้เพื่อหาค่าของคีย์ตัวใหม่ต่อไป

$$k1 = register$$

การทำงานจะวนเช่นนี้ไปเรื่อยๆจนครบสิบรอบเช่นเดียวกันกับการเข้ารหัสบนวงจรปกติ

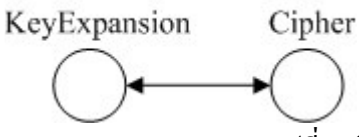
เนื่องจากวงจรเข้ารหัสในงานวิจัยชิ้นนี้ทำงานแบบเปลี่ยนโครงแบบไปเรื่อยๆ ดังนั้นแบบแผนในการเปลี่ยนโครงแบบจึงเป็นเรื่องที่ต้องพิจารณา ในหัวข้อต่อไปจะกล่าวถึงขั้นตอนวิธีการเปลี่ยนโครงแบบ ความสัมพันธ์ของแต่ละโครงแบบ

4.2.2 กราฟแสดงการทำงาน

จากเนื้อหาในบทที่ 3 เราจะนำความรู้ดังกล่าวมาใช้อธิบายพฤติกรรมการเปลี่ยนโครงแบบและพฤติกรรมในการแลกเปลี่ยนข้อมูลจากโครงแบบหนึ่งไปยังอีกโครงแบบหนึ่ง จากแนวคิดเบื้องต้นที่ได้กล่าวมาประกอบกับแนวคิดด้านแพลตฟอร์มสามารถสร้างกราฟที่ใช้แสดงการทำงานได้ดังนี้

จากแนวคิดเบื้องต้นการเข้ารหัสเมื่อทำเป็นส่วนวงจรย่อยๆแล้วจะประกอบด้วยสองส่วนคือ

ส่วนหาคีย์และส่วนที่เปลี่ยนสถานะข้อความซึ่งในหัวข้อนี้จะขอเรียกส่วนหาคีย์ใหม่ว่า “KeyExpansion” และส่วนที่ใช้เปลี่ยนสถานะข้อความเรียกว่า “Cipher” ทั้งสองส่วนทำงานผลัดกันไปขึ้นกับว่าในขณะนั้นมีจุดประสงค์อะไร เมื่อนำมาสร้างเป็นกราฟก็จะได้กราฟที่สามารถเปลี่ยนตัวเองไปมาระหว่างสองวงจรนี้ได้เป็นดังรูปที่ 4.22

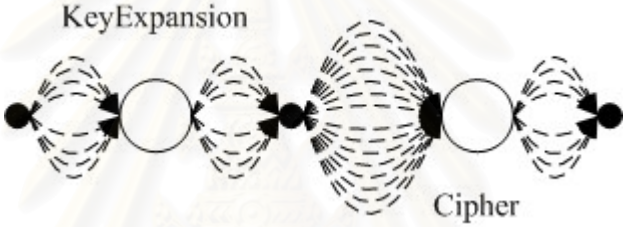


รูปที่ 4.22 กราฟแสดงการทำงานของวงจรเปลี่ยนโครงแบบต้นแบบ

จากกราฟเห็นได้ว่าวงจรย่อยสองอันมีพฤติกรรมที่สามารถเปลี่ยนโครงแบบกันไปมาระหว่างสองวงจรย่อยได้ ซึ่งในวงจรปกติทั่วไปนั้นจะไม่มีเส้นเชื่อมระหว่างวงจรคือเป็นแค่วงจรเดี่ยวๆ โดยไม่มีการเปลี่ยนแปลงใดๆ

4.2.3 กราฟแสดงการส่งถ่ายข้อมูล

หัวข้อที่ได้อธิบายถึงพฤติกรรมการทำงานการเปลี่ยนโครงแบบของวงจรย่อย ต่อไปนี้จะขออธิบายเรื่องการส่งถ่ายข้อมูลบ้าง เนื่องจากระบบมีส่วนประกอบสองส่วนหลักๆคือ รีจิสเตอร์ และส่วนที่เปลี่ยนโครงแบบได้ ซึ่งเปรียบได้กับส่วนที่มีส่วนประกอบคงที่กับส่วนที่เปลี่ยนโครงแบบไปเรื่อยๆ การติดต่อสื่อสารกันจำเป็นต้องมีบัสครอบคลุม (Global Bus) ในการส่งต่อข้อมูล พฤติกรรมการส่งข้อมูลผ่านบัสครอบคลุมนี้จะแตกต่างกันไปตามสถานการณ์และวงจรย่อยในขณะนั้น เช่น วงจรย่อย A ใช้ข้อมูลในการประมวลผลมากกว่าวงจร B เป็นต้น กราฟของวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้จะเป็นดังในรูปที่ 4.23



รูปที่ 4.23 กราฟแสดงการส่งถ่ายข้อมูลระหว่างวงจรย่อยแต่ละวงจรกับรีจิสเตอร์

จากรูปจะเห็นได้ว่าความต้องการใช้ข้อมูลของแต่ละวงจรย่อยมีไม่เท่ากันรวมทั้งการส่งต่อข้อมูลให้กับรีจิสเตอร์ก็ไม่เท่ากันด้วยเช่นกัน ขออธิบายการทำงานของวงจรดังนี้ เริ่มแรกวงจรย่อยที่ชื่อว่าวงจรขยายคีย์จะทำการโหลดค่ามาจากรีจิสเตอร์โดยทำการโหลดเข้ามาทั้งหมดแปดครั้งครั้งละ 16 บิตรวมแล้วเป็น 128 บิตซึ่งเมื่อวงจรขยายคีย์ทำการประมวลผลเพื่อหาค่าคีย์เสร็จสิ้นแล้วจะทำการส่งค่าผลลัพธ์ไปเก็บที่รีจิสเตอร์โดยเก็บทั้งหมด 128 บิต เวลาถัดมาวงจรย่อยไซเฟอร์จึงมีการโหลดค่าเข้าไปทำงานค่าที่โหลดเข้าไปมีทั้งหมด 256 บิต นั่นคือประกอบด้วยคีย์ 128 บิต และข้อความที่ต้องการเปลี่ยนสถานะอีก 128 บิต หลังจากเปลี่ยนสถานะข้อความเสร็จสิ้นแล้วก็จะส่งค่าไปเก็บที่รีจิสเตอร์เช่นเดิม แต่จะส่งไปเก็บเพียงแค่ค่าของข้อความเท่านั้นเพราะว่าไม่มีความจำเป็นที่ต้องใช้คีย์เก่าอีกครั้ง ปัญหาที่เกิดขึ้นคือความล่าช้าในการทำงานเนื่องจากทุกๆการเปลี่ยนสถานะข้อความครั้งหนึ่งๆจำเป็นต้องเสียเวลาในการโหลดมากเสียยิ่งกว่าเวลาทำงานของวงจรย่อยเสียอีก ปัญหาการโหลดที่ว่านี้จะถูกแก้ไขให้ดีขึ้นในวงจรรุ่นต่อไปซึ่งจะกล่าวถึงในภายหลัง

4.3 รายละเอียดการออกแบบ

ในหัวข้อนี้จะลงลึกถึงรายละเอียดการออกแบบทั้งหมดของต้นแบบวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้ ก่อนอื่นต้องนำแนวความคิดที่กล่าวไปในหัวข้อก่อนหน้านี้มาทำเป็นสิ่งที่จับต้องได้มากขึ้น แนวคิดรวบยอดของหัวข้อก่อนหน้านี้มีใจความดังนี้

การทำงานเข้ารหัสในแบบวงจรปรับเปลี่ยนโครงร่างได้มีการทำงานสองส่วนหลักๆที่เกี่ยวข้องกันนั่นคือ

1. การหาคีย์
2. การเปลี่ยนสถานะของข้อความ

เมื่อนำการทำงานทั้งสองมาพิจารณาร่วมกันแล้วจะได้เป็นวงจรจากแนวคิดข้างต้นเป็นดังนี้

1. KeyExpansion – วงจรสำหรับการหาคีย์
2. Cipher – วงจรที่ใช้เปลี่ยนสถานะของข้อความ

วงจรย่อยสองวงจรดังกล่าวจะถูกนำมารวมเข้ากับแพลตฟอร์มซึ่งตัวแพลตฟอร์มประกอบด้วยสามส่วน นั่นคือ รีจิสเตอร์, หน่วยประมวลผลที่สามารถเปลี่ยนโครงแบบได้และหน่วยควบคุม วงจรย่อยทั้งสองจะถูกนำมารวมเข้ากับหน่วยประมวลผลที่เปลี่ยนโครงแบบได้ นอกจากนี้เมื่อเราเอาแนวคิดมาลงในวงจรจริงๆจำเป็นต้องมีการปรับปรุงดัดแปลงแก้ไขเพื่อความเหมาะสมด้วย ส่วนประกอบสามส่วนเมื่อลงมือทำจริงจะได้เป็นดังนี้

1. รีจิสเตอร์แบงก์ (Register Bank) – ซึ่งมาจากรีจิสเตอร์
2. หน่วยประมวลผลเปลี่ยนโครงแบบได้ – ซึ่งเปลี่ยนไปได้ทั้ง KeyExpansion, Cipher
3. หน่วยควบคุมการประมวลผลที่ควบคุมวิถีข้อมูลทั้งสอง

4.3.1 รีจิสเตอร์แบงก์

หน้าที่หลักของรีจิสเตอร์แบงก์คือเก็บข้อมูลข้อความและคีย์ที่เพื่อนำไปคำนวณค่าหาสถานะต่อไป คีย์และข้อความที่ใช้มีขนาด 128 บิต ดังนั้นจำเป็นต้องมีรีจิสเตอร์แบงก์ที่มีขนาดอย่างน้อย 256 บิต แต่ก่อนที่จะลงรายละเอียดไปมากกว่านี้จะขอกล่าวถึงโครงสร้างข้อมูลเพื่อเก็บข้อความและคีย์ก่อน

ข้อความและคีย์จะถูกนำไปใช้งานในลักษณะของแถวลำดับ (Array) สองมิติ นั่นคือเป็นแถวลำดับช่องละแปดบิตจำนวน 16 ช่องเรียงต่อกันดังรูปที่ 4.24

	Col 0	Col 1	Col 2	Col 3
Row 0	00	01	02	03
Row 1	10	11	12	13
Row 2	20	21	22	23
Row 3	30	31	32	33

รูปที่ 4.24 โครงสร้างแถวลำดับสำหรับเก็บข้อมูลและคีย์

จากโครงสร้างๆง่ายนี้เราสามารถนำไปใช้เป็นแนวคิดพื้นฐานเพื่อสร้างรีจิสเตอร์แบงก์ขนาด 256 บิตได้โดยนำโครงสร้างแถวลำดับของทั้งสองอย่างมารวมเข้าไว้ด้วยกันซึ่งจะทำให้หน้าตาของรีจิสเตอร์แบงก์ออกมาเป็นดังรูปที่ 4.25

	Col 0	Col 1	Col 2	Col 3	
Row 0	00	01	02	03	KEY
Row 1	10	11	12	13	
Row 2	20	21	22	23	
Row 3	30	31	32	33	
Row 4	40	41	42	43	DATA
Row 5	50	51	52	53	
Row 6	60	61	62	63	
Row 7	70	71	72	73	

รูปที่ 4.25 รีจิสเตอร์แบงก์ขนาด 256 บิตแบ่งเป็นสองส่วนสำหรับเก็บข้อมูลและคีย์

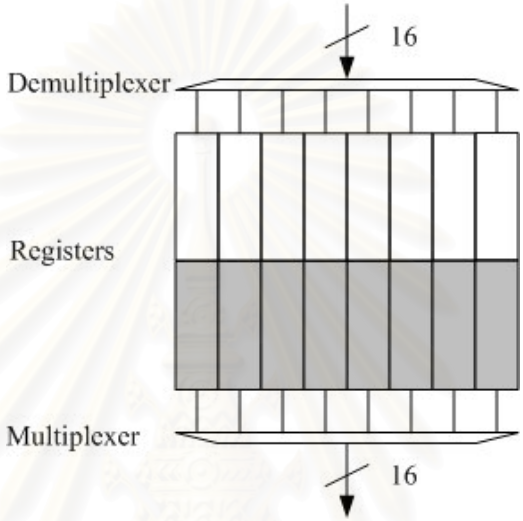
จากรูปที่ 4.25 เราจะทำการแยกอาณาเขตของข้อมูลออกเป็นสองส่วนคือที่ที่เก็บข้อความกับที่ที่เก็บคีย์ในแต่ละรอบที่หามาได้ เมื่อใดก็ตามที่มีการโหลดค่าจากรีจิสเตอร์แบงก์ก็สามารถเลือกได้ว่าจะใช้คีย์หรือข้อความที่เก็บไว้ซึ่งข้อมูลแต่ละอย่างจะถูกเก็บไว้ในลักษณะแยกบล็อค (Block) บล็อคละ 8 บิตโดยมีทั้งหมด 16 บล็อค แต่ในความเป็นจริงนั้นวงจรที่ทำงานบน 8 บิตแทบไม่มีให้ได้พบเห็นจึงล้าสมัยเกินไปดังนั้นเราจึงขยับมาทำงานที่ 16 บิตแทนส่งผลให้โครงสร้างในการจับรวมตัวกันเป็นบล็อคเปลี่ยนไปแต่ก็เพียงเล็กน้อยเท่านั้นตามรูปที่ 4.26

	Col 0		Col 1		
Row 0	00	01	02	03	KEY
Row 1	10	11	12	13	
Row 2	20	21	22	23	
Row 3	30	31	32	33	
Row 4	40	41	42	43	DATA
Row 5	50	51	52	53	
Row 6	60	61	62	63	
Row 7	70	71	72	73	

รูปที่ 4.26 รีจิสเตอร์แบงก์ที่ถูกจัดรูปแบบการเก็บข้อมูลใหม่

จากรูปที่ 4.26 เห็นได้ว่ารีจิสเตอร์แบงก์ถูกแบ่งออกเป็นบล็อคบล็อคละ 16 บิต เพื่อให้สะดวกในการโหลดและทำงานเนื่องจากถ้าเป็นแปดบิตจะทำให้วงจรล้าสมัยและทำงานได้ต่ำกว่าเกณฑ์ที่รับได้

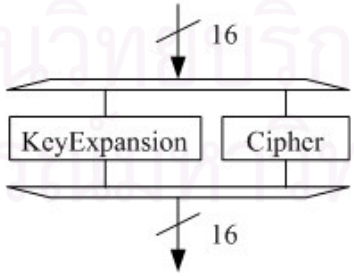
หลังจากที่เราเตรียมรูปแบบการเก็บข้อมูลไว้พร้อมแล้วเราจะเริ่มมาออกแบบรีจิสเตอร์แบงก์ ตามแนวคิดที่เราได้วางไว้รีจิสเตอร์แบงก์จะประกอบด้วยหน่วยความจำรีจิสเตอร์ย่อยๆขนาด 16 บิตจำนวน 16 ตัว หน่วยความจำรีจิสเตอร์ย่อยดังกล่าวแต่ละตัวสามารถโหลดค่าเข้ามาและนำค่าออกไปคนละทาง เราจะประกอบหน่วยความจำรีจิสเตอร์ย่อยเหล่านี้เข้าด้วยกันโดยให้สามารถทำการโหลดค่าเข้าและออกได้ทีละ 16 บิตซึ่งอาศัยอุปกรณ์รวมส่งสัญญาณ (Multiplexer) สำหรับการส่งข้อมูลออกจากรีจิสเตอร์แบงก์ และ อุปกรณ์แยกส่งสัญญาณ (Demultiplexer) สำหรับการโหลดข้อมูลเข้าไปยังรีจิสเตอร์แบงก์ โครงสร้างที่ว่าจะออกมาเป็นดังรูปที่ 4.27



รูปที่ 4.27 รีจิสเตอร์ในรูปแบบโครงสร้างที่นำไปใช้งานจริง

ข้อมูลออกรีจิสเตอร์แบงก์นี้ไปยังหน่วยประมวลผลที่เปลี่ยนโครงแบบได้เพื่อทำการประมวลผลแล้วนำค่าที่ได้มาเก็บไว้ที่รีจิสเตอร์แบงก์รอการนำไปใช้ในเวลาถัดไป

4.3.2 หน่วยประมวลผลที่เปลี่ยนโครงแบบได้



รูปที่ 4.28 หน่วยประมวลผลซึ่งประกอบด้วยวงจรย่อยขยายคีย์และ Cipher

ส่วนประกอบนี้เป็นส่วนสำคัญที่ทำให้วงจรเข้ารหัสในงานชิ้นนี้สมบูรณ์ หน่วยประมวลผลนี้มีจุดเด่นที่สามารถเปลี่ยนโครงแบบของตัวเองได้หรือที่เรียกว่า Reconfigurable แต่เนื่องจากในขณะนี้ยังไม่มีเทคโนโลยีที่สามารถทำการเปลี่ยนโครงแบบตัวเองได้ หน่วยประมวลผลนี้จึงถูกสร้าง

ขึ้นมาบนพื้นฐานที่เป็นเพียงการจำลองการทำงานของมันเท่านั้น โดยเราจะใช้อุปกรณ์รวมส่งสัญญาณและอุปกรณ์แยกส่งสัญญาณในการจำลองการเปลี่ยนโครงแบบของตัวเองโดยภายในวงจรจะเลือกได้ว่าจะส่งหรือรับข้อมูลจากวงจรย่อยใดไม่ว่าจะเป็นวงจรขยายคีย์และไซเฟอร์ซึ่งทั้งสองวงจรย่อยนี้จะผลัดกันทำงานตามคำสั่งของหน่วยควบคุมการประมวลผลอีกที โครงสร้างของหน่วยประมวลผลที่สร้างโดยใช้อุปกรณ์รวมส่งสัญญาณและอุปกรณ์แยกส่งสัญญาณจะเป็นดังรูปที่ 4.28

จากรูปที่ 4.28 จะเห็นได้ว่าวงจรย่อยทั้งสองมีทางติดต่อกับภายนอกได้เพียงสองทางเท่านั้น คือเข้าหรือออก

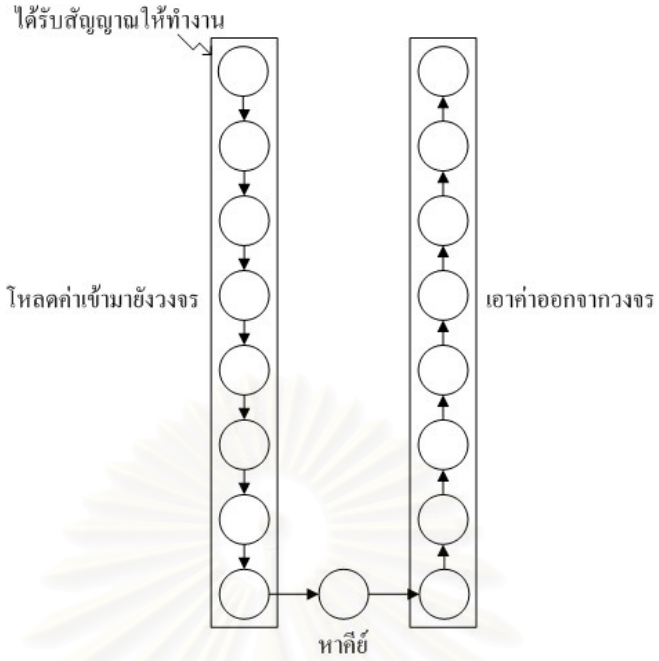
4.3.3 วงจรย่อย KeyExpansion

การเข้ารหัสเออีเอสจะทำงานด้วยกันทั้งหมดสิบรอบแต่ละรอบก็จะใช้คีย์ที่ทำการเข้ารหัสต่างกันไปแต่การได้มาซึ่งคีย์ที่ใช้ในแต่ละรอบมีหลักการเดียวกันคือการใช้วิธีการที่เรียกว่า Key Expansion วงจรย่อยขยายคีย์ที่กล่าวถึงนี้จะทำหน้าที่เป็นตัวหาคีย์เพื่อใช้งานในแต่ละรอบ

เริ่มแรกจะขออธิบายแผนภาพสถานะของวงจร KeyExpansion

รูปที่ 4.29 แผนภาพสถานะของวงจร KeyExpansion

รูปที่ 4.29 แสดงแผนภาพสถานะของวงจรขยายคีย์ซึ่งประกอบด้วยการทำงานห้าขั้นตอน ซึ่งห้าขั้นตอนที่แสดงในรูปนั้นเป็นการแสดงให้เห็นภาพรวมของการทำงานของวงจรย่อยขยายคีย์เท่านั้น ซึ่งเมื่อทำการย่อและขยายในส่วนที่ยังขาดไปแล้วจะทำให้สถานะทั้งหมดออกมาเป็นดังรูปที่ 4.30



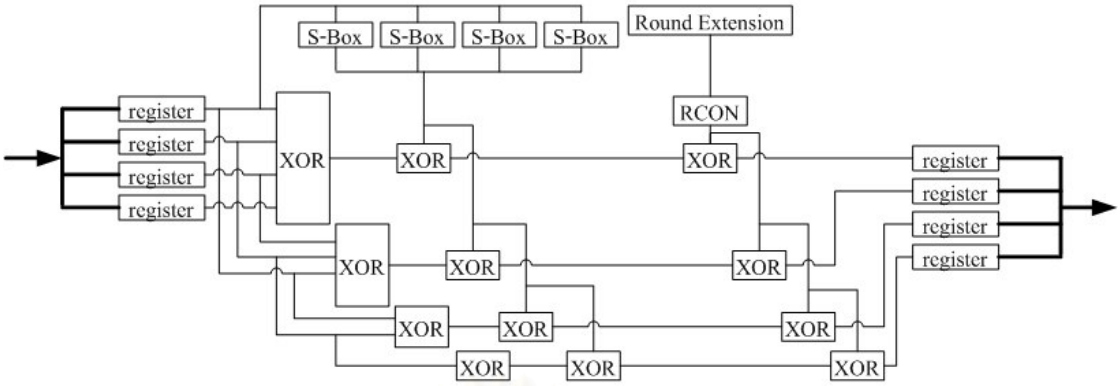
รูปที่ 4.30 สถานะของวงจรขยายคีย์ในรูปแบบที่นำไปใช้จริง

ส่วนที่เพิ่มเติมเข้ามาได้แก่สถานะที่ใช้ในการโหลดและนำค่าออกจากวงจรซึ่งใช้ถึงอย่างละ 8 สถานะด้วยกันนับว่าเป็นการใช้เวลามากพอสมควร ส่วนสถานะที่ใช้ในการหาคีย์นั้นมีแค่สถานะเดียวเท่านั้นเพราะสถานะอื่นๆสามารถทำงานพร้อมๆกันไประหว่างการโหลดได้ หลังจากที่วงจรย่อยนี้ทำงานจนถึงสถานะสุดท้ายหน่วยควบคุมการประมวลผลจะทำการสั่งให้วงจรอื่นทำงานต่อไป หลังจากอธิบายเรื่องสถานะของการทำงานแล้วเราจะมาอธิบายภายในวงจร

ภายในวงจรย่อยขยายคีย์มีส่วนประกอบสำคัญอยู่สามส่วนด้วยกันดังในรูปที่ 4.31 ได้แก่

1. หน่วยความจำรีจิสเตอร์ย่อย – เพื่อเก็บสำรองข้อมูลที่นำมาใช้ในวงจร
2. เอสบ็อกซ์ (S-Box) – แปลงค่าของข้อมูลเพื่อใช้ในการหาคีย์ใหม่
3. ราวด์เอ็กเทนชั่น (Round Extension) – หาค่าคงที่ในแต่ละรอบสำหรับหาคีย์

สังเกตได้ว่าภายในวงจรมีรีจิสเตอร์ทั้งหัวและท้ายเพื่อจุดประสงค์ในการเก็บข้อมูลไว้ระหว่างทำงานและขั้นตอนการทำงานภายในไม่จำเป็นต้องมีสถานะอะไรซับซ้อนเพราะเราสามารถปล่อยให้ข้อมูลไหลไปจนสุดโดยใช้แค่เพียงสถานะเดียว เส้นเข้มสีดำหมายถึงช่องทางเข้าออกวงจรโดยจะเข้านำข้อมูลเข้ามาสำรองไว้ในรีจิสเตอร์ทางขวาก่อนเมื่อคำนวณเสร็จสิ้นจึงปล่อยข้อมูลไปให้รีจิสเตอร์ทางซ้ายสุดก่อนที่จะนำออกจากวงจรไปเก็บไว้ในรีจิสเตอร์แบงก์ต่อไป

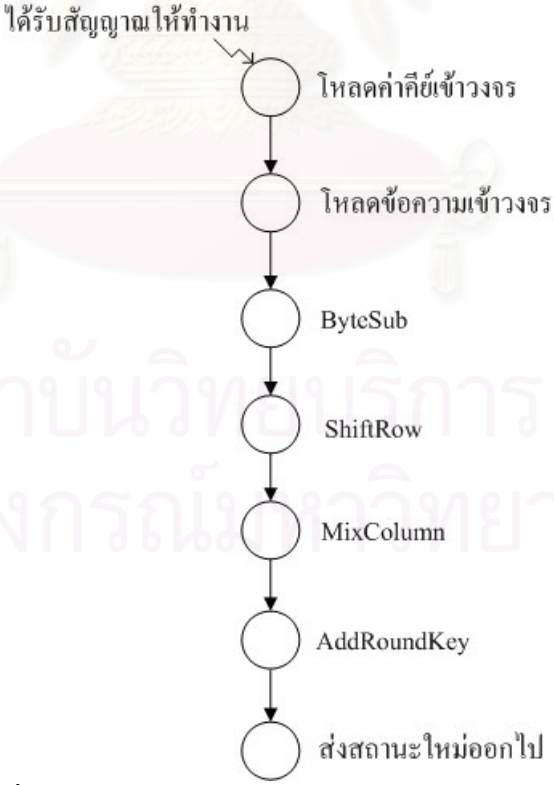


รูปที่ 4.31 ส่วนประกอบภายในวงจรย่อย KeyExpansion

4.3.4 วงจรย่อย Cipher

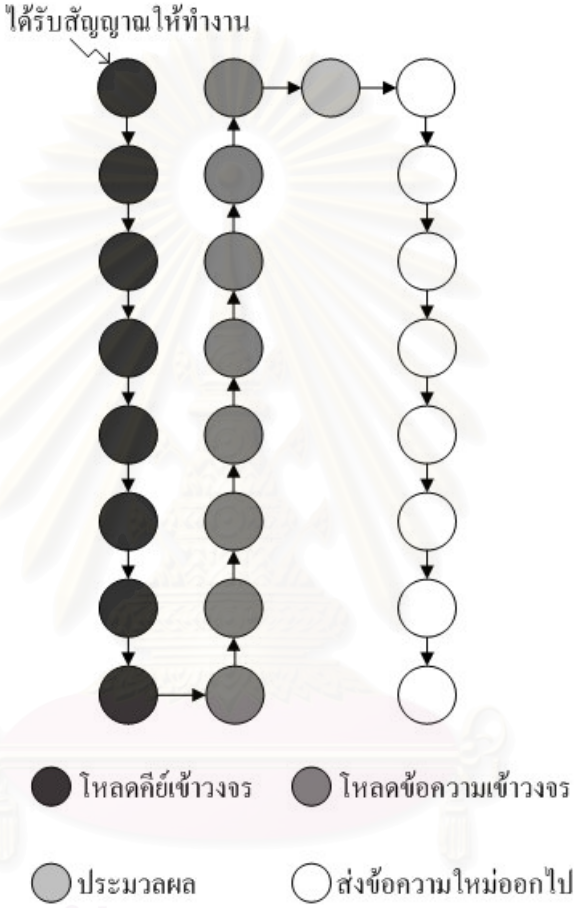
หลักการทำงานคือวงจรไซเฟอร์นี้ทำงานโดยรับอินพุตต์สองส่วนคือ ข้อความ และ คีย์ แล้วคืนผลลัพธ์เป็นข้อความสถานะใหม่ออกไป ในการเข้ารหัสแบบเออีเอสวงจรไซเฟอร์นี้มีหน้าที่ในการเปลี่ยนสถานะของข้อความทั้งหมดสิบรอบด้วยกันและผลลัพธ์ของรอบที่สิบจะเป็นข้อความสุดท้ายที่ได้เข้ารหัสเรียบร้อยแล้ว

เราสามารถเขียนแผนภาพสถานะโดยที่ยังไม่ได้ลงรายละเอียดได้เป็นดังรูปที่ 4.32



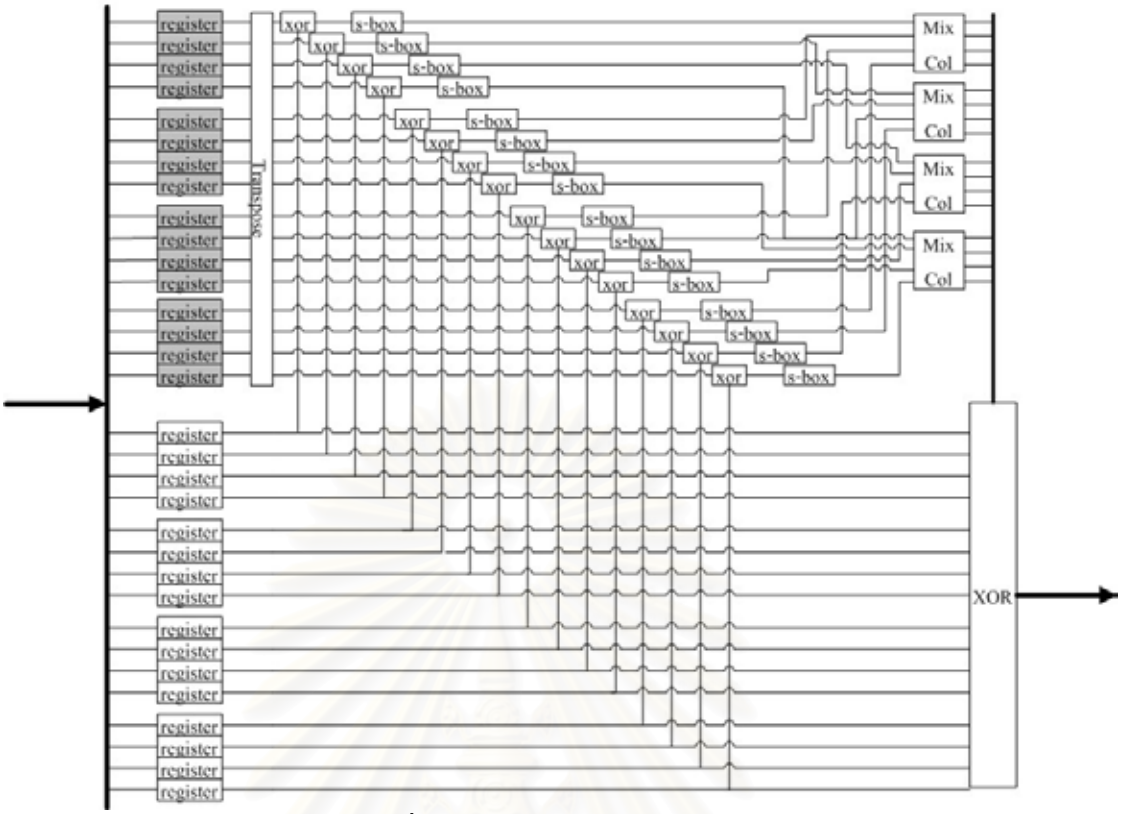
รูปที่ 4.32 แผนภาพสถานะคร่าวๆของวงจรย่อย Cipher

จากรูปที่ 4.32 วงจรย่อยไซเฟอร์เริ่มทำงานโดยการโหลดค่าคีย์และข้อความเข้ามาในวงจร เช่นเดียวกับวงจรขยายคีย์แล้วทำค่อยๆเปลี่ยนสถานะของข้อความตามลำดับคือ ByteSub, ShiftRow, MixColumn และการบวกคีย์แต่ละรอบเสร็จแล้วจึงส่งข้อความสถานะใหม่ออกไป รูปที่ 4.32 เป็นเพียงสถานะหยาบๆเท่านั้นไม่ได้ลงรายละเอียดว่าการสิ้นเปลืองไปกับการทำงานแต่ละกระบวนการเป็นเท่าไร รูปที่ 4.33 สามารถแสดงรายละเอียดได้ดีกว่าดังนี้



รูปที่ 4.33 แผนภาพแสดงสถานะเต็มของวงจรย่อ Cipher

จากรูปที่ 4.33 แผนภาพสถานะของไซเฟอร์ประกอบด้วยสามส่วนคือโหลดค่า ประมวลผล และส่งค่าออก เวลาในการทำงานของวงจรนี้ส่วนใหญ่หมดไปกับการโหลดค่าเนื่องจากจำเป็นต้องโหลดค่าเข้าทั้งคีย์และข้อความซึ่งรวมกันแล้วเป็นจำนวน 256 บิตนอกจากนั้นยังมีการนำข้อความสถานะใหม่ออกไปอีกเป็นจำนวน 128 บิต แผนภาพสถานะดังกล่าวนำมาสร้างเป็นวงจรดังในรูปที่ 4.34



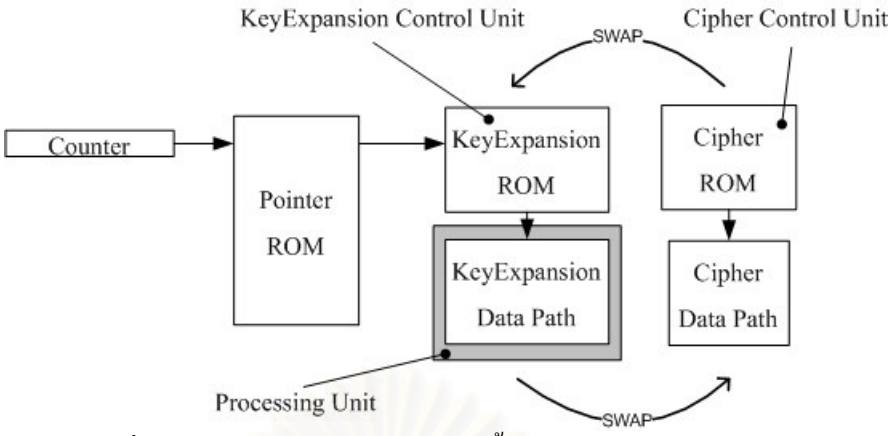
รูปที่ 4.34 ภายในวงจรย่อย Cipher

จากรูปรีจิสเตอร์สีขาวมีหน้าที่สำหรับสำรองข้อมูลที่เป็นคีย์ที่ได้รับมาจากรีจิสเตอร์แบงก์ ส่วนรีจิสเตอร์สีเทานั้นจะรับข้อมูลที่เป็นข้อความที่ต้องการเปลี่ยนสถานะที่ได้รับมาจากรีจิสเตอร์แบงก์เช่นกัน ในรอบแรกของการเปลี่ยนสถานะจำเป็นต้องมีการสลับเปลี่ยน (Transpose) ข้อความ จากนั้นข้อความจะไหลเข้ามาเอ็กซ์ออร์กับคีย์แล้วส่งผลลัพธ์ที่ได้ไปยังเอสบ็อกซ์เพื่อเทียบหาค่าแทนที่ไบต์ค่าที่ได้จะส่งต่อไปกระบวนการผสมหลักและการบวกคีย์แต่ละรอบตามลำดับแล้วค่อยส่งออกจากวงจรย่อย แต่ถ้าข้อความที่ต้องการเปลี่ยนสถานะเข้ามาไม่ใช่รอบแรกก็จะไม่มีการสลับเปลี่ยนก่อนเข้าไปทำกระบวนการต่อไป ถ้าเป็นการเปลี่ยนสถานะข้อความครั้งสุดท้ายจะไม่มีกระบวนการผสมหลักผลลัพธ์หลังจากรอบสุดท้ายจะมายังบัสขาสุดของวงจรเพื่อเตรียมส่งไปยังรีจิสเตอร์แบงก์

4.3.5 หน่วยควบคุม

เราได้กล่าวถึงแนวคิดของหน่วยควบคุมมาบ้างแล้วในหัวข้อที่ผ่านมา คราวนี้เราจะมาขยายความและลงถึงรายละเอียดในการออกแบบและสร้างหน่วยควบคุมเพื่อให้เหมาะกับงานวิจัยชิ้นนี้

หน่วยควบคุมที่วางแนวคิดไว้ตอนแรกคือเป็นเซตของหน่วยควบคุมที่ถูกควบคุมด้วยตัวชี้ (Pointer) เมื่อตัวชี้ชี้ไปที่สถานะใดและหน่วยควบคุมตัวใดแล้วหน่วยควบคุมตัวนั้นก็จะทำงานตามสถานะที่ได้ชี้ไปด้วย ตัวชี้ที่เราพูดถึงนี้จะใช้ตัวนับ (Counter) ร่วมกับรอม (ROM) เพื่อออกสัญญาณไปยังหน่วยควบคุมย่อย ส่วนหน่วยควบคุมย่อยก็สร้างขึ้นจากรอมเช่นกันดังจะเห็นได้จากรูปที่ 4.35



รูปที่ 4.35 ความสัมพันธ์ระหว่างตัวชี้หน่วยควบคุมแล้ววิถีข้อมูล

จากรูปที่ 4.35 เห็นได้ว่าสัญญาณจะเริ่มต้นออกจากตัวนับโดยตัวมันเองจะออกสัญญาณไปยังรอมเรียงลำดับไปเรื่อยๆจนกว่าจะหมดข้อมูลในรอม จากนั้นรอมจะส่งสัญญาณไปบอกหน่วยควบคุมย่อยเพื่อออกสัญญาณควบคุมที่แท้จริงลงไปยังหน่วยประมวล (วิถีข้อมูล) เพื่อประมวลผลแล้วส่งข้อมูลกลับมายังรีจิสเตอร์แบงก์ รอมที่เป็นตัวควบคุมใหญ่กับรอมที่เป็นหน่วยควบคุมย่อยมีหลักการออกแบบต่างกัน รอมที่เป็นตัวควบคุมหลักหรือรอมที่เป็นตัวชี้ถูกออกแบบมาโดยต้องรู้ว่ารอมตัวย่อยแต่ละตัวทำงานอย่างไรในแต่ละสถานะ ส่วนรอมย่อยแต่ละตัวถูกสร้างขึ้นมาให้ทำงานเฉพาะทางของมันโดยไม่ต้องรู้จักตัวอื่นเลย ดังนั้นเราจึงควรออกแบบและสร้างรอมตัวย่อยแต่ละตัวให้เสร็จก่อนจากนั้นค่อยออกแบบและสร้างรอมตัวหลัก ในหัวข้อต่อไปจะขอกล่าวถึงการวางโครงสร้างข้อมูลในรอมหลักแต่จะไม่ขอลงรายละเอียดไปกับรอมย่อยๆ เส้นที่ชื่อ SWAP หมายถึงการผลัดเปลี่ยนทำงานระหว่างสองวงจรในรูป

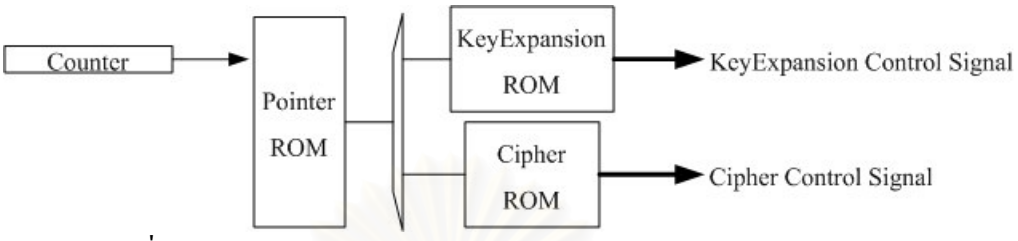
4.3.6 รอมตัวชี้ (Pointer ROM)

เป็นหน่วยความจำสำคัญในการกำหนดทิศทางการทำงานของระบบโดยรวม มีหลักการทำงานคือรับสัญญาณมาจากตัวนับแล้วส่งผลลัพธ์ไปยังรอมที่เป็นหน่วยประมวลผลย่อย มีรูปแบบการเก็บข้อมูลเป็นคู่ลำดับดังนี้ (ชนิดของวงจรย่อย, สถานะของวงจรย่อย) คู่ลำดับพวกนี้จะถูกเรียงลำดับก่อนแล้วค่อยใส่ลงไปในรอม สาเหตุที่ต้องเรียงลำดับเพราะต้องการลดงานที่ตัวนับต้องทำ กล่าวคือตัวนับไม่จำเป็นรู้ว่าในขณะนั้นต้องใช้วงจรอะไรและสถานะใดมีหน้าที่แค่นับไปเรื่อยๆจนสุดรอมแล้วจึงหยุดทำงาน

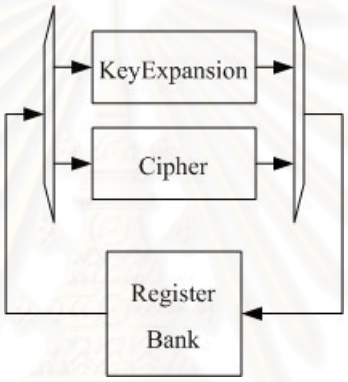
4.3.7 วงจรเข้ารหัสเออีเอสชนิดเปลี่ยนโครงแบบได้ต้นแบบ

หลังจากที่เราได้กล่าวถึงแนวคิดและวิธีการออกแบบมาอย่างละเอียดแล้วสุดท้ายในส่วนของหัวข้อนี้ เราจะนำเอาความรู้ที่ได้ทั้งหมดมาประกอบกันเป็นวงจรเข้ารหัส เมื่อทุกส่วนมาทำงานร่วมกันวิธีการทำงานจะคล้ายกับการสั่งให้วงจรเฉพาะทางวงจรหนึ่งทำงานไปสักพักพอถึงจุดที่นอกเหนือการทำงานของมันแล้วก็จะเปลี่ยนไปให้อีกวงจรหนึ่งที่ถนัดงานนั้นทำงานแทน และ

ระหว่างการเปลี่ยนวงจรก็จะมีการเก็บผลลัพธ์ที่ได้ลงในรีจิสเตอร์แบงก์ก่อนจะนำไปใช้ในขั้นตอนต่อไปโครงสร้างของระบบแสดงเป็นรูปได้สองลักษณะคือ โครงสร้างสัญญาณควบคุม และโครงสร้างของวงจรปฏิบัติงาน ตามรูปที่ 4.36 และรูปที่ 4.37



รูปที่ 4.36 ภาพรวมของโครงสร้างการส่งสัญญาณควบคุมไปยังวงจรย่อย



รูปที่ 4.37 ภาพรวมวิถีข้อมูลของหน่วยประมวลผล

จุดเชื่อมต่อระหว่างสองรูปนี้คือสัญญาณควบคุมที่ออกมาจากรูปที่ 4.36 สัญญาณเหล่านี้จะวิ่งตรงเข้ามายังวงจรย่อยขยายคีย์และไซเฟอร์ซึ่งนอกจากจะควบคุมวงจรย่อยทั้งสองแล้วสัญญาณดังกล่าวยังควบคุมรีจิสเตอร์แบงก์เพื่อให้ทำงานรับส่งค่าได้สอดคล้องกันกับวงจรย่อยที่กำลังทำงาน

4.4 ประสิทธิภาพ

การวัดประสิทธิภาพของวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้ ทำได้สองลักษณะคือวัดด้านความเร็วและความประหยัด ความเร็วสามารถวัดได้จากจำนวนสัญญาณนาฬิกา (Clock) และความถี่สูงสุดที่ทำได้เนื่องจากจำนวนสัญญาณนาฬิกาที่ใช้จะบอกว่าใช้สถานะเป็นจำนวนเท่าไรถึงจะทำงานเสร็จยิ่งมากก็ยิ่งสิ้นเปลืองเวลาทำงาน ส่วนความประหยัดวัดได้จากจำนวนเกตสมมูล (Equivalent gates) เนื่องจากการสร้างวงจรหนึ่งๆจำเป็นต้องใช้เกตประกอบกันขึ้นมา ขนาดของวงจรรวมขึ้นโดยตรงกับจำนวนเกตถ้าจำนวนเกตยิ่งมากหมายถึงขนาดที่ใหญ่ตาม ถ้าจำนวนเกตน้อยนั่นคือวงจรจะออกมามีขนาดเล็ก ราคาของวงจรก็ได้รับผลกระทบมาจากขนาดวงจรด้วยเช่นกัน

4.4.1 ความเร็วในการทำงาน

ประสิทธิภาพการทำงานเป็นไปตามตารางที่ 4.1

ตารางที่ 4.1 ความเร็วของวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้

จำนวนสัญญาณนาฬิกา	ความถี่สูงสุด (เมกะเฮิรตซ์)
471	43.743

ตารางที่ 4.1 แสดงให้เห็นว่าวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้มีการสิ้นเปลืองจำนวนสัญญาณนาฬิกาเป็นอย่างมากเพราะจริงๆแล้วการเข้ารหัสแบบในกรณีที่เป็นวงจรเฉพาะทางโดยตรงนั้นจะใช้น้อยสุดแค่จำนวนสัญญาณประมาณ 12 สัญญาณนาฬิกา [12] สิ่งที่ทำให้แตกต่างกันอย่างมากคือเวลาที่เสียไปในการติดต่อสื่อสารกับรีจิสเตอร์แบงก์ เนื่องจากมีช่องทางที่ติดต่อได้แค่ 16 บิต แต่ในแต่ละครั้งจะใช้ข้อมูลที่มากกว่า 16 บิตไปหลายเท่าตัวจึงใช้ประมาณ 40 สัญญาณนาฬิกาต่อรอบด้วยกัน ทางแก้ปัญหาคอขวดนี้อาจมีได้หลายทางเช่นขยายช่องสัญญาณที่ใช้ในการติดต่อหรือลดขนาดของข้อมูลที่นำมาประมวลผล เป็นต้น ซึ่งจะกล่าวถึงในบทต่อไป

4.4.2 ขนาดของวงจร

เนื่องจากวงจรถูกออกแบบบนสมมติฐานที่ว่าสามารถเปลี่ยนโครงแบบได้ ดังนั้นการวัดขนาดของวงจรจะวัดเฉพาะแค่ส่วนที่ใหญ่ที่สุดเพียงส่วนเดียวแล้วนำมารวมกับส่วนของวงจรที่เปลี่ยนโครงแบบไม่ได้ก็จะได้ค่าขนาดของวงจรออกมา ขนาดของวงจรในบทนี้จึงหาได้จากการนำเอาขนาดของรีจิสเตอร์แบงก์รวมกับวงจรย่อยที่ใหญ่ที่สุดระหว่างวงจรย่อยขยายคีย์กับวงจรย่อยไซเฟอร์

ตารางที่ 4.2 ตารางแสดงขนาดของวงจรย่อยแต่ละวงจรย่อย

	KeyExpansion (รวมรีจิสเตอร์แบงก์)	Cipher (รวมรีจิสเตอร์แบงก์)	หน่วยควบคุม
จำนวนเกตสมมูล	14,584	29,127	1,757

จากตารางที่ 4.2 สามารถสรุปได้ว่าขนาดของวงจรทั้งหมดเป็น 29,127 + 1,757 = 30,884 เกตสมมูล นับว่ายังออกมาไม่ค่อยดีนักเนื่องจากตัววงจรเฉพาะทางนั้นมีขนาดแค่ประมาณ 40,000 เกตเท่านั้น วิธีการปรับปรุงสามารถทำได้ด้วยการแบ่งวงจรย่อยให้มีขนาดเล็กลงไปอีกหรืออาจทำโดยออกแบบวงจรใหม่ให้เล็กลงกว่าเดิมจะกล่าวในบทถัดไป

บทที่ 5

การปรับปรุงและพัฒนาประสิทธิภาพวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้

ดังเห็นได้จากผลการทดลองต้นแบบวงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้ในบทที่แล้วยังไม่มีประสิทธิภาพเป็นที่น่าพอใจเท่าไรนัก เนื่องจากใช้ทรัพยากรน้อยลงแค่หนึ่งในสี่แต่ประสิทธิภาพการประมวลผลต่ำกว่าหลายเท่าตัว ในบทนี้จึงได้นำเอาวงจรดังกล่าวมาปรับปรุงใหม่โดยมีจุดที่แตกต่างไปจากตัววงจรเดิมค่อนข้างมาก เนื้อหาที่จะกล่าวถึงจึงมีตั้งแต่การวิเคราะห์วงจรเดิมว่ามีข้อดีข้อเสียอย่างไรแก้ไขได้อย่างไร มีส่วนใดที่ควรเพิ่มเข้าไปหรือตัดทิ้งเพื่อเพิ่มประสิทธิภาพการทำงาน

5.1 ปัญหาที่เกิดขึ้น

การทำงานของวงจรที่ใช้แนวคิดการเปลี่ยนโครงแบบได้เป็นไปอย่างถูกต้องตามที่กำหนดไว้ทุกอย่างแต่ก็ยังมีสิ่งที่ไม่ประทับใจเท่าที่ควรคือวงจรต้นแบบดังกล่าวไม่ประหยัดทรัพยากรตามที่ได้คาดหวังไว้ เนื่องจากแนวคิดในตอนเริ่มต้นเพียงต้องการให้วงจรสามารถใช้งานได้จริงก่อนที่จะไปคำนึงถึงประสิทธิภาพการใช้ แต่หลังจากผ่านระยะนั้นมาแล้วเราจะมาแก้ไขปัญหาเกี่ยวกับประสิทธิภาพที่พบระหว่างการทดลองเข้ารหัสจริงๆ ประสิทธิภาพเมื่อเปรียบเทียบกับวงจรเข้ารหัสเออีเอสแบบปกติจะเป็นดังตารางที่ 5.1

ตารางที่ 5.1 ตารางเปรียบเทียบการใช้ทรัพยากรและความเร็วของวงจรเข้ารหัสสองแบบ

	ทรัพยากร (เกตสมมูล)	ความถี่สูงสุด (เมกะเฮิรตซ์)
วงจรเข้ารหัสเออีเอสปกติ	40,621	137.3
วงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้	30,884	43.7

จากตารางที่ 5.1 เห็นได้ว่าทรัพยากรที่ใช้ต่างกันแค่ประมาณร้อยละ 25 เท่านั้น แต่ในขณะที่ความเร็วต่างกันสามเท่าซึ่งเกินขอบเขตที่รับได้ จากตารางสิ่งที่จะแก้ไขเป็นอันดับแรกจึงเป็นเรื่องความเร็วแต่งานวิจัยชิ้นนี้เน้นเรื่องการประหยัดมากกว่าความเร็วจึงขอพิจารณาเรื่องการใช้ทรัพยากรมาเป็นอันดับแรกส่วนความเร็วเป็นเรื่องรองลงไป

5.2 วิเคราะห์และแนวคิดแก้ปัญหา

ปัญหาแรกที่จะนำมาวิเคราะห์เป็นปัญหาเรื่องการใช้ทรัพยากร วงจรทั้งหมดที่มีในระบบได้แก่ วงจรไซเฟอร์ (Cipher) ส่วนวงจรขยายคีย์ (KeyExpansion) นอกจากนี้ยังจำเป็นต้องนับวงจรเสริมด้วยเช่นหน่วยความจำรีจิสเตอร์แบงก์ โดยในส่วนของรีจิสเตอร์แบงก์คงไม่มีอะไรเหลือให้ปรับปรุงเพราะทรัพยากรถูกใช้ไปไม่มากนักเมื่อเปรียบเทียบกับสองวงจรที่ได้กล่าวมาข้างต้น

ดังนั้นเราจะมาวิเคราะห์โครงสร้างของวงจรทั้งสองอีกทีหนึ่ง จากบทที่แล้วเห็นได้ว่าตัววงจรที่ใช้ทรัพยากรมากที่สุดเป็นวงจรไซเฟอร์มากกว่าวงจรขยายคีย์ถึงเท่าตัว ถ้าเราทำการลดทรัพยากรของวงจรขยายคีย์ย่อมจะไม่ส่งผลต่อขนาดของวงจรรวมแน่นอนเพราะอย่างไรก็ตามวงจรที่ใหญ่ที่สุดนั้นก็ยังมีขนาดเท่าเดิม ดังนั้นทางออกที่ดีสุดในเวลานี้คือการลดวงจรที่มีขนาดใหญ่สุดในระบบก่อนนั่นคือวงจรไซเฟอร์ ส่วนการลดทรัพยากรที่ใช้อาจทำได้สองวิธีคือลดส่วนที่ไม่จำเป็นออกไปจากระบบหรือแบ่งย่อยวงจรให้เล็กลงไปอีกซึ่งวิธีหลังเป็นวิธีที่เหมาะสมกว่าเนื่องด้วยเราควรจะใช้คุณสมบัติการเปลี่ยนโครงแบบได้ให้เป็นประโยชน์สูงสุด

เริ่มแรกเราจะมาวิเคราะห์โครงสร้างของวงจรไซเฟอร์กันก่อน ดังที่ได้กล่าวไว้ในบทที่แล้วว่าวงจรไซเฟอร์ประกอบด้วยอะไรบ้าง คราวนี้จะวิเคราะห์ว่าส่วนใดที่ใช้ทรัพยากรมากเป็นพิเศษเพื่อที่จะได้แยกส่วนนั้นออกไปเป็นอีกวงจรเพื่อทำให้ขนาดของวงจรรวมลดลง ส่วนประกอบของวงจรไซเฟอร์เป็นดังตารางที่ 5.2

ตารางที่ 5.2 ตารางเปรียบเทียบการใช้ทรัพยากรในวงจรไซเฟอร์

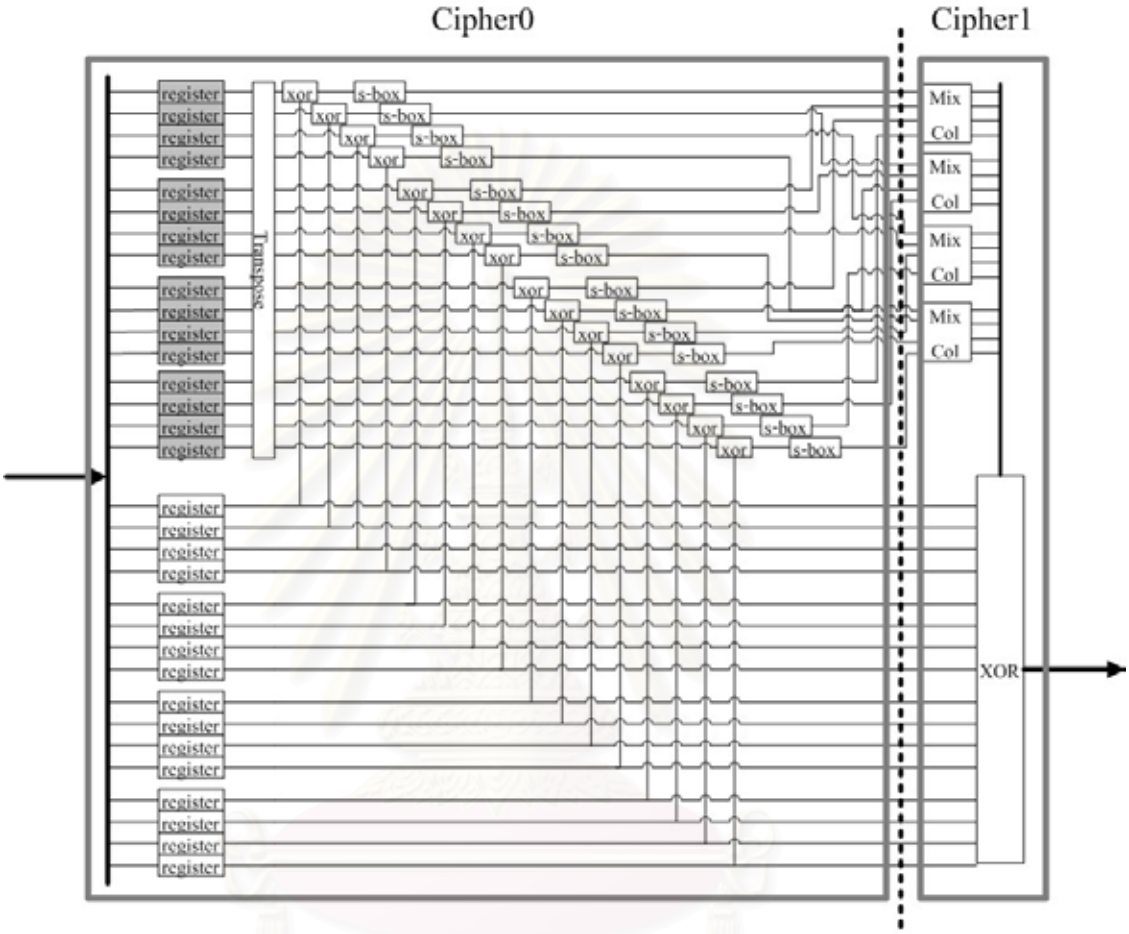
ส่วนประกอบ	ขนาด
เอสบ็อกซ์	9,024 (1,128*8)
ส่วนผสมหลัก	1,536 (384*4)
ส่วนบวกคีย์แต่ละรอบ	< 500

ส่วนประกอบที่ใช้พื้นที่มากสุดตามตารางแล้วจะเห็นได้ว่าเป็นเอสบ็อกซ์เพราะตัวมันเองถูกสร้างขึ้นมาจากตารางขนาด 2^8 ทั้งหมด (look-up table) ซึ่งใช้ในวงจรถึง 16 อันด้วยกัน ดังนั้นถ้าจะลดทรัพยากรลงควรพิจารณาที่เอสบ็อกซ์เป็นอันดับแรก แต่อย่างไรก็ตามการแยกวงจรเอสบ็อกซ์มาทำงานเดี่ยวๆเลยจะไม่เหมาะสมเท่าไรนักเพราะจะเสียค่าใช้จ่ายอื่น (Overhead) เพิ่มเติมเข้าไปด้วยไม่ว่าจะเป็นการสร้างรีจิสเตอร์เพื่อรับค่ามาประมวลผลหรือเวลาที่ใช้ในการทำงานที่ต้องมากขึ้นโดยใช่เหตุเพราะเกิดจากการโหลดค่าเข้าวงจรเพื่อมาทำงานเล็กๆเช่นงานแทนที่ไบต์ (ByteSub) เพียงงานเดียว ด้วยเหตุดังกล่าวทำให้โจทย์ที่ต้องการจะนำเอาส่วนที่สิ้นเปลืองจากวงจรไซเฟอร์ไปสร้างเป็นวงจรใหม่นั้นต้องกลับมาทบทวนอีกครั้ง เราควรจะแยกเซตของวงจรที่สิ้นเปลืองออกมาไม่ใช่แค่ส่วนใดส่วนหนึ่งเพื่อจะได้คุ้มค่ากับการโหลดข้อมูลในแต่ละครั้งอีกทั้งยังทำให้วงจรมีขนาดเล็กลงได้มากกว่าแบบแยกมาแค่ส่วนใดส่วนหนึ่งโดยเฉพาะอีกด้วย ดังนั้นวงจรไซเฟอร์จะถูกแบ่งออกมาเป็นสองก้อนโดยแต่ละก้อนมีองค์ประกอบดังนี้

1. ก้อนแรกประกอบด้วยกระบวนการ แทนที่ไบต์ (ByteSub) และ เลื่อนแถว(ShiftRow) โดยให้ชื่อว่าวงจรไซเฟอร์ 0 (Cipher0)

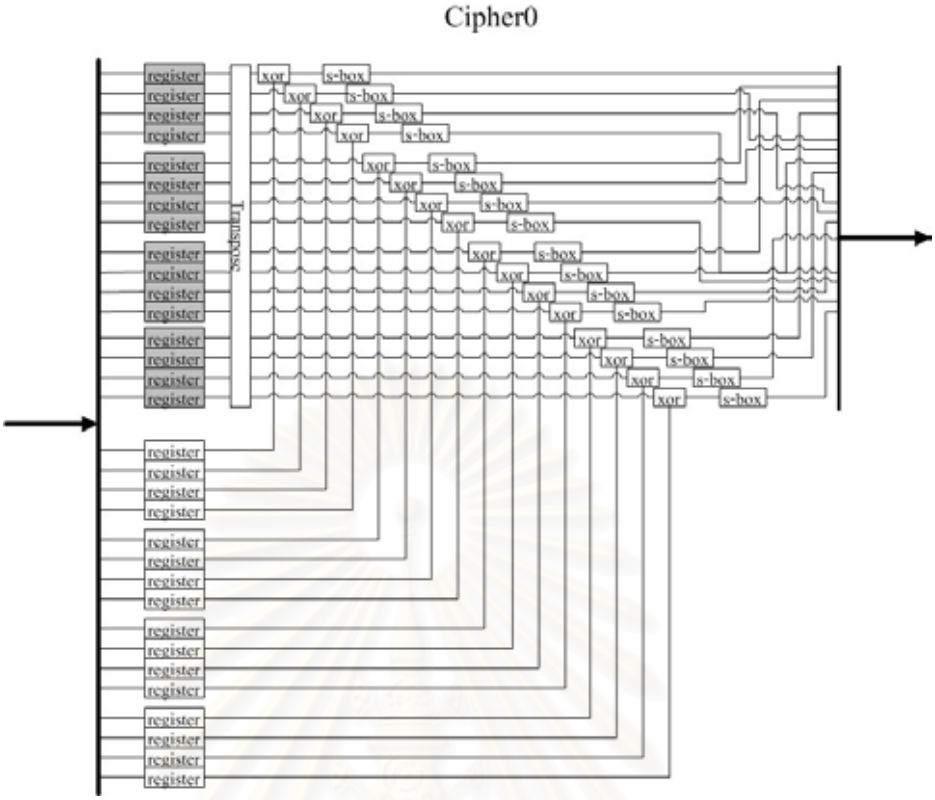
2. ก้อนสองประกอบด้วยกระบวนการ ผสมหลัก (MixColumn) และ บวกคีย์แต่ละรอบ (AddRoundKey) โดยให้ชื่อว่าวงจรไซเฟอร์ 1 (Cipher1)

รูปที่ 5.1 แสดงถึงเส้นแบ่งระหว่างวงจรใหม่ที่ได้ทั้งสอง

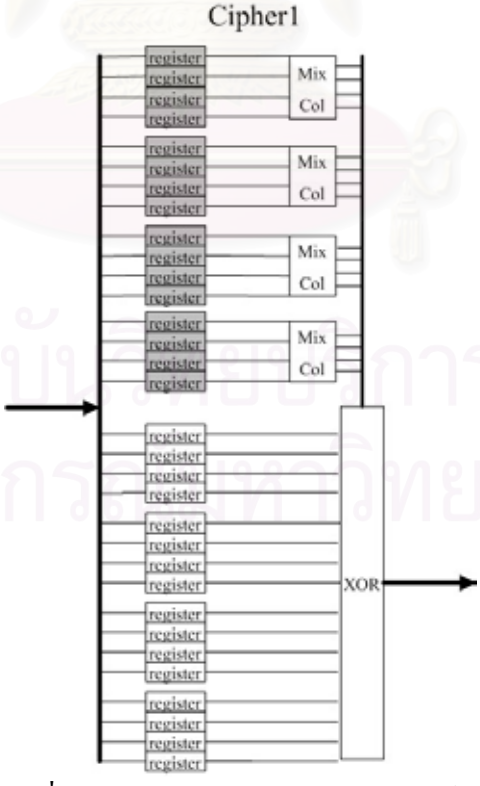


รูปที่ 5.1 วงจรไซเฟอร์เมื่อทำการสร้างเส้นแบ่งวงจรออกเป็นสองส่วนแล้ว

การแบ่งวงจรออกเป็นสองส่วนดังกล่าวจำเป็นต้องเพิ่มรีจิสเตอร์เพื่อส่งต่อข้อมูลข้ามรอยต่อของวงจรด้วยส่งผลให้วงจรทั้งสองเมื่อแยกกันโดยสมบูรณ์แล้วจะออกมาเป็นดังรูปที่ 5.2 และ 5.3

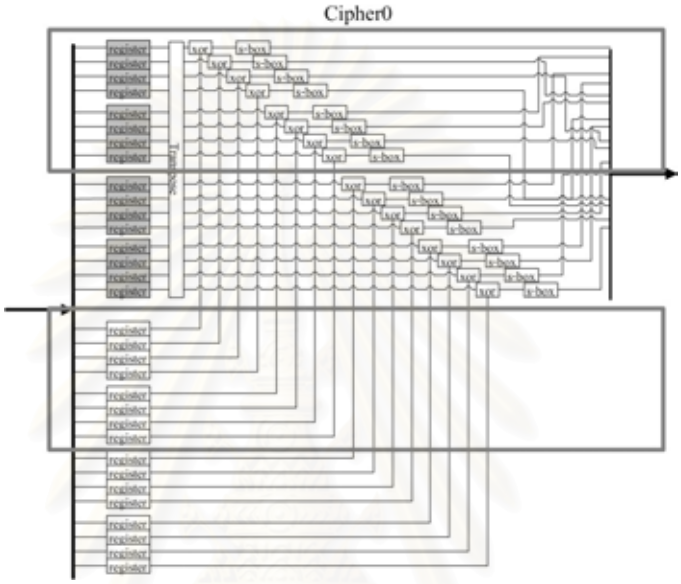


รูปที่ 5.2 โครงสร้างภายในวงจรไซเฟอร์ 0



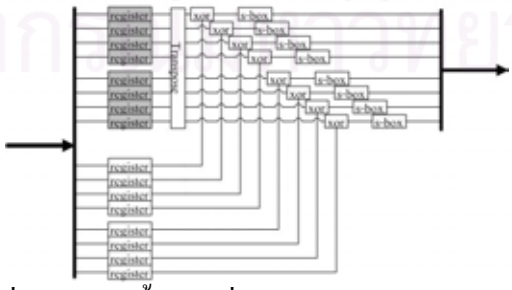
รูปที่ 5.3 โครงสร้างภายในวงจรไซเฟอร์ 1

เมื่อทำการแยกวงจรทั้งสองออกจากกันแล้วทำการดูขนาดของวงจร พบว่าตัววงจรไซเฟอร์ 0 ไม่ได้มีขนาดเล็กลงเท่าที่ควรโดยมีขนาดใหญ่กว่าวงจรไซเฟอร์ 1 ค่อนข้างมาก สาเหตุที่ทำให้วงจรไซเฟอร์ 0 ยังมีขนาดใหญ่อยู่เป็นเพราะว่าเอสบ็อกซ์ที่มีอยู่ในวงจร 16 ตัวด้วยกันซึ่งนับว่ามากเกินไปดังนั้นจึงจำเป็นต้องมีการลดขนาดของวงจรไซเฟอร์ 0 ลงไปอีกโดยทำการแบ่งออกเป็นสองวงจรซึ่งมีเอสบ็อกซ์แค่ 8 อันต่อวงจร ดังรูปที่ 5.4



รูปที่ 5.4 เส้นแบ่งของวงจรไซเฟอร์ 0

จากรูปที่ 5.4 แสดงเส้นแบ่งวงจรไซเฟอร์ 0 ส่วนของวงจรที่ถูกล้อมรอบด้วยสี่เหลี่ยมสีเทาจะถูกนำไปรวมกันเป็นวงจรเดียวกันส่วนที่อยู่นอกสี่เหลี่ยมสีเทาก็จะถูกแยกไปสร้างเป็นอีกวงจรเช่นเดียวกันแต่เนื่องด้วยว่าการแบ่งแบบนี้ไม่ได้แบ่งจากวงจรที่ทำงานต่อเนื่องกันจึงไม่จำเป็นต้องเพิ่มรีจิสเตอร์เพื่อจำค่าผลลัพธ์ชั่วคราวเอาไว้แต่สามารถจับวงจรสองอันนี้แยกออกจากกันได้เลย เมื่อแยกสองวงจรออกจากกันจะได้เป็นดังรูปที่ 5.5



รูปที่ 5.5 วงจรทั้งสองเมื่อถูกจับแยกออกจากกันแล้ว

วงจรทั้งสองเมื่อแยกออกมาจากกันแล้วจะมีหน้าตาที่ไม่ต่างกันมากนัก โดยส่วนที่ต่างกันมีเพียงส่วนที่เรียกว่าส่วนสลับเปลี่ยน (Transpose) เท่านั้น นอกนั้นจะมีโครงสร้างเหมือนกันหมด

ส่วนสลับเปลี่ยนต่างกันเพราะตัวสลับเปลี่ยนที่ใช้ในวงจรเป็นตัวสลับเปลี่ยนที่ทำการต่อสายตรงเอาไว้ไม่ได้มีการสลับเปลี่ยนเกิดขึ้นจริงๆ เราจะเรียกวงจรสองอันที่แยกออกมาใหม่นี้ว่าไซเฟอร์เอสบี 0 (CipherSB0) และ ไซเฟอร์เอสบี 1 (CipherSB1) แล้วเปลี่ยนชื่อวงจรย่อยไซเฟอร์ 1 เป็นไซเฟอร์เอ็มเอ็กซ์ (CipherMX) ในขณะนี้มีวงจรย่อยทั้งหมดในระบบสี่วงจรคือ ไซเฟอร์เอสบี 0 ไซเฟอร์เอสบี 1 ไซเฟอร์เอ็มเอ็กซ์ และ วงจรขยายคีย์ หน้าที่ของแต่ละวงจรสามารถสรุปได้คร่าวๆดังตารางที่ 5.3

ตารางที่ 5.3 หน้าที่ของวงจรไซเฟอร์เอสบี 0 ไซเฟอร์เอสบี 1 ไซเฟอร์เอ็มเอ็กซ์ และ วงจรขยายคีย์

วงจร	หน้าที่
ไซเฟอร์เอสบี 0	กระบวนการแทนที่ไบต์ของแถวลำดับข้อมูล ตำแหน่งที่ 00, 01, 02, 03, 10, 20, 30
ไซเฟอร์เอสบี 1	กระบวนการแทนที่ไบต์ของแถวลำดับข้อมูล ตำแหน่งที่ 11, 12, 13, 21, 22, 23, 31, 32, 33
ไซเฟอร์เอ็มเอ็กซ์	ทำการเลื่อนแถว* ผสมหลัก บวกคีย์แต่ละรอบ
วงจรขยายคีย์	หาคีย์แต่ละรอบ

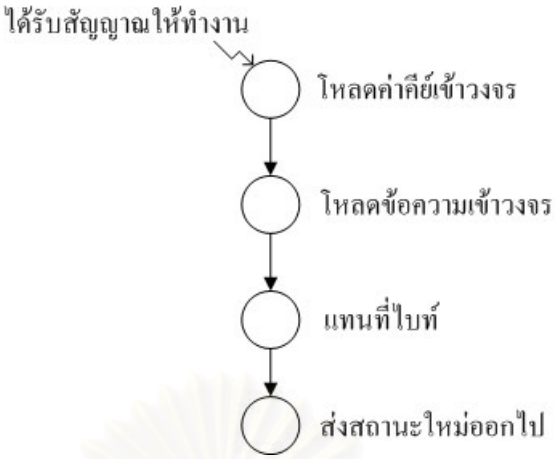
*ได้ทำการย้ายกระบวนการเลื่อนแถวจากเดิมไว้ในไซเฟอร์ 0 มาไว้ในไซเฟอร์เอ็มเอ็กซ์เพื่อความสะดวกในการเขียนโปรแกรม

เมื่อนำวงจรย่อยทั้งสี่มาทำงานรวมกันจะมีลำดับการใช้งานของวงจรย่อยเรียงลำดับตั้งแต่วงจรขยายคีย์ ไซเฟอร์เอสบี 0 ไซเฟอร์เอสบี 1 และ ไซเฟอร์เอ็มเอ็กซ์ จากนั้นจะวนไปเรื่อยจนกว่าจะทำงานเสร็จ หลังจากที่ได้แบ่งย่อยวงจรเสร็จสิ้นแล้วจึงทำการวิเคราะห์ด้านพฤติกรรมการทำงาน

พฤติกรรมของวงจรขยายคีย์: ไม่เปลี่ยนแปลงเนื่องจากไม่ได้มีการเปลี่ยนแปลงวงจรเกิดขึ้น

พฤติกรรมของไซเฟอร์เอสบี 0 และ *1*: สถานะภายในวงจรคือการโหลดค่าเข้า จากนั้นนำค่าข้อมูลที่ได้ไปประมวลด้วยการแทนที่ไบต์แล้วนำผลลัพธ์ไปเก็บดังรูปที่ 5.6

พฤติกรรมของไซเฟอร์เอ็มเอ็กซ์: คือกระบวนการภายในที่เหลือจากการทำไซเฟอร์ 0 และ 1 นั่นคือการเลื่อนแถว การผสมหลัก และ การบวกคีย์แต่ละรอบดังรูปที่ 5.7

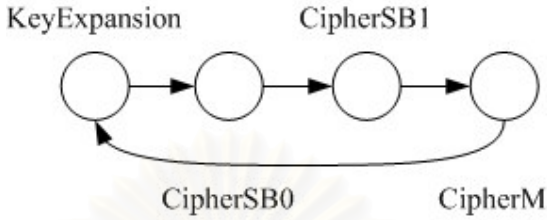


รูปที่ 5.6 แผนภาพสถานะการทำงานคร่าวๆของวงจรไซเฟอร์ 0 และ 1

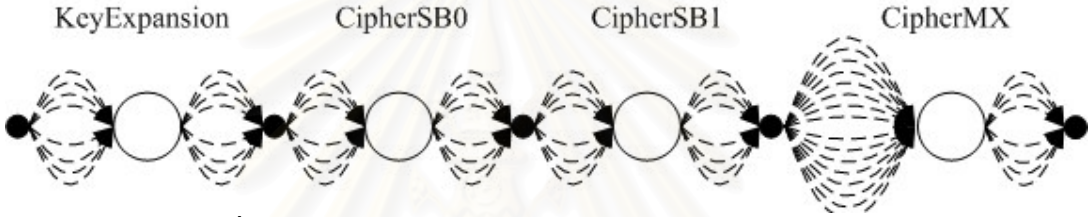
รูปที่ 5.7 แผนภาพสถานะการทำงานคร่าวๆของวงจรไซเฟอร์เอ็มเอ็กซ์

อย่างไรก็ตามรูปที่ 5.6 และ 5.7 เป็นแผนภาพสถานะที่ยังไม่ได้ลงรายละเอียดหรือตัดทอนสถานะที่สามารถทำงานพร้อมกันได้ออกไปเป็นเพียงแค่แนวความคิดในการทำงานเท่านั้น แต่เมื่อใส่สถานะต่างๆลงไปแล้วสถานการณ์ทำงานของไซเฟอร์เอ็มเอ็กซ์จะได้เป็นดังรูปที่ 4.33 (คือมีการโหลดข้อมูลทั้งหมด 16 สถานะ ประมวลผลทำงาน 1 สถานะ และ ส่งข้อมูลออกอีก 8 สถานะ) ส่วนวงจรไซเฟอร์เอสบี 0 และ 1 นั้นมีความแตกต่างตรงที่ใช้สถานะในการโหลดข้อมูลลดลงจากเดิมครึ่งหนึ่งนั่นคือมีสถานะโหลดแค่ 8 สถานะนอกนั้นจะเหมือนกันกับวงจรไซเฟอร์เอ็มเอ็กซ์

พฤติกรรมที่กล่าวข้างต้นอธิบายแค่พฤติกรรมโดดๆของแต่ละตัว ครานี้เรามาดูพฤติกรรมของทั้งสี่วงจรเมื่อมาทำงานในระบบเปลี่ยนโครงแบบได้บ้าง พฤติกรรมจะถูกนำเสนอผ่านทางกราฟการทำงานและกราฟการถ่ายโอนข้อมูล กราฟใหม่ที่จะเป็นดังรูปที่ 5.8 และ 5.9



รูปที่ 5.8 กราฟแสดงการทำงานของวงจรเข้ารหัสเออีเอเมื่อแบ่งย่อยวงจรใหม่



รูปที่ 5.9 กราฟแสดงการรับส่งข้อมูลระหว่างวงจรย่อยแต่ละอัน

ทั้งสองรูปสื่อความหมายการทำงานของวงจรเข้ารหัสแบบที่มีการแบ่งย่อยวงจรใหม่รูปที่ 5.8 อธิบายว่าวงจรมีการวางลำดับการเปลี่ยนโครงแบบก่อนหลังเป็นอย่างไรซึ่งถ้าลำดับดังกล่าวผิดไปการเข้ารหัสก็จะผิดไปด้วยเช่นกัน สังเกตได้ว่าเส้นเชื่อมแต่ละอันเป็นเส้นเชื่อมที่มีทิศทางนั่นคือเมื่อวงจรเปลี่ยนโครงแบบไปเป็นอีกวงจรหนึ่งแล้วจะไม่เปลี่ยนโครงแบบกลับมาที่วงจรเดิมจนกว่าจะครบรอบของการทำงานกราฟในรูปที่ 5.8 ไม่ได้บอกรายละเอียดเรื่องการถ่ายโอนข้อมูลที่ถูกออกออกแบบใหม่ซึ่งถูกกล่าวไว้ในรูปที่ 5.9 จากรูปที่ 5.9 เห็นได้ว่าการรับส่งข้อมูลกระจายตัวพอๆกันนั่นคือแต่ละวงจรจะใช้ขนาดของข้อมูลพอๆกัน มีเพียงแต่วงจรไซเฟอร์เอ็มเอ็กซ์เท่านั้นที่ใช้ข้อมูลมากกว่าวงจรอื่นเพราะนอกจากต้องโหลดค่าข้อความทั้งหมดแล้วยังจำเป็นต้องโหลดคีย์เข้ามาทำการบวกคีย์แต่ละรอบด้วยจึงนับได้ว่าเป็นคอขวดของระบบ

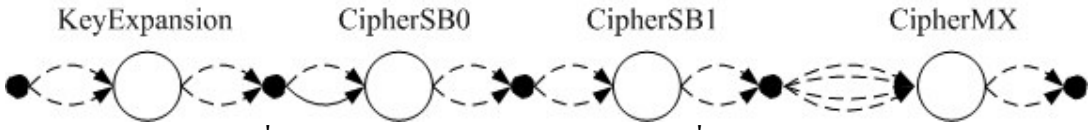
การเปลี่ยนพฤติกรรมการทำงานมาเป็นแบบนี้ทำให้ประสิทธิภาพในการประมวลผลหรือความเร็วนั้นยิ่งลดลงไปอีกซึ่งจากการประมาณแล้วเวลาการทำงานจะอยู่ที่กว่า 700 สัญญาณนาฬิกา แต่ทรัพยากรที่ใช้ลดลงอย่างเห็นได้ชัด (จะไปสรุปในหัวข้อประสิทธิภาพที่ปรับปรุงแล้ว) ด้วยเหตุนี้จึงน่าจะมีแนวทางการปรับปรุงให้ประสิทธิภาพความเร็วในการประมวลผลให้ดีกว่าที่เป็นอยู่

หลังจากที่เราวิเคราะห์และหาแนวทางลดการใช้ทรัพยากรแล้วปัญหาเรื่องที่รองลงมาคือความเร็ว ดังจะเห็นว่าวงจรแต่ละอันที่เราได้ออกแบบสิ้นเปลืองเวลากับการโหลดข้อมูลเข้ามาทำงานแต่เวลาที่ทำงานจริงๆภายในวงจรมีแค่ 1 สัญญาณนาฬิกาเท่านั้นดังจะเห็นได้จากตารางที่ 5.4

ตารางที่ 5.4 ตารางแสดงจำนวนสัญญาณนาฬิกาที่ใช้ในวงจรย่อยต่างๆ

วงจร	จำนวนสัญญาณนาฬิกาเพื่อโหลดค่า	จำนวนสัญญาณนาฬิกาเพื่อประมวลผล	จำนวนสัญญาณนาฬิกาเพื่อส่งค่าออกไป
ไซเฟอร์	16	1	8
ไซเฟอร์ 0 และ 1	16	1	8
ไซเฟอร์เอสบี 0 และ 1	8	1	8
ไซเฟอร์เอ็มเอ็กซ์	16	1	8
ขยายคีย์	8	1	8

จากตารางที่ 5.4 เมื่อคิดเป็นร้อยละของการทำงานแล้วการโหลดข้อมูลเข้าวงจรจะคิดเป็นร้อยละ 47-64 ของการเรียกใช้วงจรนั้นซึ่งนับว่ากินเวลามาก ลองมาดูที่กราฟรับส่งข้อมูลที่รูปที่ 5.9 ปัญหาที่เห็นได้เด่นชัดคือจำนวนเส้นเชื่อมที่มากเกินไปยิ่งโดยเฉพาะเส้นเชื่อมที่ต่อกับไซเฟอร์เอ็มเอ็กซ์ซึ่งมีด้วยกันถึง 16 เส้นด้วยกัน ดังนั้นหนทางที่จะเร่งประสิทธิภาพของระบบนี้คือต้องลดเส้นเชื่อมพวกนี้ลงให้น้อยที่สุด เส้นเชื่อมแต่ละเส้นเกิดจากการถ่ายโอนข้อมูลขนาด 16 บิต จากรีจิสเตอร์แบงก์ไปยังวงจรต่างๆ การจะลดเส้นเชื่อมก็ทำได้โดยลดจำนวนข้อมูลที่ใช้ลงหรือเพิ่มขนาดให้เส้นเชื่อมใหญ่ขึ้นเช่นจาก 16 บิต เป็น 32 บิตเป็นต้น แต่แนวทางในการลดจำนวนข้อมูลที่ส่งถ่ายนั้นเป็นเรื่องที่ทำได้ยากเพราะการทำงานมักเป็นไปแบบขนานคือต้องทำทีเดียวพร้อมๆกันไปหมด การแก้ปัญหาด้วยวิธีขยายขนาดของเส้นเชื่อมน่าจะเป็นทางออกที่ดีกว่า คราวนี้ก็ต้องมาพิจารณากันว่าเราจะเพิ่มจากเส้นละ 16 บิต เป็นขนาดเท่าไร จากรูปที่ 5.9 สังเกตได้ชัดว่าจำนวนเส้นเชื่อมที่ใช้ในแต่ละวงจรเป็น 8, 8, 8, 16 เมื่อคิดเป็นขนาดบิตแล้วจะได้เป็น 128, 128, 128, 256 ดังนั้นถ้าจะขยายขนาดของเส้นเชื่อมให้คุ้มค่าโดยไม่สิ้นเปลืองที่สุดน่าจะขยายเป็นขนาด 128 บิต (เพราะถ้าเป็น 256 บิตเส้นเชื่อมจะไม่ถูกใช้งานเต็มที่) เมื่อพิจารณาถึงความคุ้มค่าในการใช้งานก็ต้องจำเป็นต้องพิจารณาด้วยว่าแต่ละรอบของการทำงานมีการโหลดค่าเข้าไปใช้งานเต็มที่ 128 บิตหรือไม่ วงจรไซเฟอร์เอสบี 0 และ 1 เป็นวงจรที่เป็นข้อยกเว้นดังกล่าวคือวงจรย่อยนี้จะทำการโหลดข้อมูลเข้าไป 128 บิตแค่รอบแรกของการเข้ารหัสเท่านั้นส่วนรอบถัดๆไปจะใช้งานแค่ 64 บิตเท่านั้นทำให้แนวคิดเดิมที่ตั้งใจไว้ว่าจะใช้บัสขนาด 128 บิตต้องเปลี่ยนเป็นบัส 64 บิตแทน ส่งผลให้กราฟส่งถ่ายข้อมูลเป็นดังรูปที่ 5.10



รูปที่ 5.10 กราฟการรับส่งข้อมูลหลังจากเพิ่มขนาดบัสแล้ว

จากการประมาณสัญญาณนาฬิกาที่ใช้แล้วจะได้ประมาณ 200 ซึ่งนับว่าดีกว่าเดิมเกือบๆสี่เท่า ประกอบกับขนาดของวงจรที่เล็กกว่าวงจรต้นแบบถึงสามเท่าด้วยกันจึงนับว่าผลออกมาเป็นที่น่าพอใจ แต่วงจรเข้ารหัสตัวนี้ยังไม่ใช่วงจรอันสุดท้ายของงานวิจัยชิ้นนี้เนื่องจากยังมีความซ้ำซ้อนภายในระบบเกิดขึ้นจากวงจรไซเฟอร์เอสบี 0 และ 1 ซึ่งทำงานเหมือนกันเพียงแต่ทำการสลับเปลี่ยนต่างกันเท่านั้น เราจึงยังสามารถปรับปรุงวงจรดังกล่าวด้วยการสร้างวงจรสลับเปลี่ยนขึ้นมาอีกตัวหนึ่งแล้วรวมวงจรไซเฟอร์เอสบี 0 และ 1 เข้าด้วยกันทำให้วงจรทั้งหมดที่มีจะเหลือเป็น วงจรขยายคีย์ (KeyExpansion) วงจรไซเฟอร์เอสบี (CipherSB) วงจรไซเฟอร์เอ็มเอ็กซ์ (CipherMX) และวงจรสลับเปลี่ยน (Transposer) อย่างไรก็ตามขนาดและเวลาการทำงานก็ไม่แตกต่างจากเดิมนักพัฒนาการของวงจรเข้ารหัสในงานวิจัยชิ้นนี้จะแสดงในรูปที่ 5.11 ซึ่งจะแสดงในลักษณะกราฟการทำงานของระบบ

รูปที่ 5.11 ลำดับความเป็นมาของระบบสุดท้าย

5.3 ประสิทธิภาพที่ปรับปรุงแล้ว

วงจรสุดท้ายที่ได้จะมีขนาดและความถี่สูงสุดเป็นดังตารางที่ 5.5 และ 5.6

ตารางที่ 5.5 ตารางแสดงการใช้ทรัพยากรของวงจรต่างๆ

	สลับเปลี่ยน	ขยายคีย์	ไซเฟอร์เอสบี	ไซเฟอร์เอ็มเอ็กซ์
จำนวนเกตสมมูล	1,411	10,830	10,435	4,547

ตารางที่ 5.6 ตารางแสดงจำนวนสัญญาณนาฬิกาและความถี่สูงสุดของระบบ

จำนวนสัญญาณนาฬิกา	ความถี่สูงสุด(เมกะเฮิรตซ์)
202	75.6

สังเกตได้จากตารางที่ 5.5 เมื่อคำนวณดูแล้วจะพบว่าระบบนี้ใช้เกตสมมูลทั้งหมด 10,830 เกตสมมูล สาเหตุที่ในระยะแรกยังได้ประสิทธิภาพไม่ดีย่อมเกิดมาจากขนาดของบัสและขนาดของวงจรซึ่งยังไม่ได้คำนึงถึงประสิทธิภาพมากเท่าที่ควร

บทที่ 6

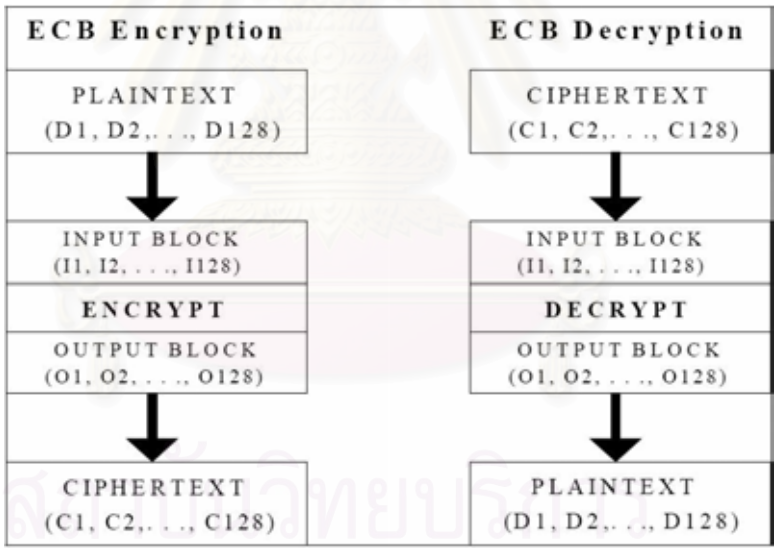
การตรวจสอบความถูกต้อง

การตรวจสอบความถูกต้องของวงจรเข้ารหัสทำโดยการใช้เวกเตอร์ทดสอบ (Test Vector) เวกเตอร์ทดสอบนี้เป็นเวกเตอร์ทดสอบชุดเดียวกันกับที่เอ็นไอเอสที (NIST: National Institute Standards and Technology) [14] ได้กำหนดไว้เพื่อใช้ทดสอบระเบียบวิธีการเข้ารหัสที่จะมาเป็นเออีเอส การทดสอบมีทั้งหมดแบ่งเป็นสองลักษณะประกอบด้วย

1.การทดสอบแบบรู้คำตอบ (KAT: Known Answer Tests) คือชุดของเวกเตอร์ทดสอบเพื่อประสิทธิภาพในแต่ละด้านของการเข้ารหัสใช้ทดสอบทีละ 128 เวกเตอร์

2.การทดสอบแบบมอนติคาร์โล (MCT: Monte Carlo Tests) คือเวกเตอร์ทดสอบที่ใช้ทดสอบทีละ 400 เวกเตอร์แต่ละเวกเตอร์ต้องเข้ารหัสย่อยๆต่อเนื่องกันอีกเป็นจำนวน 10,000 รอบ นอกจากนี้การทดสอบในแต่ละแบบจะแยกย่อยออกเป็นอีกสองโหมดคือ

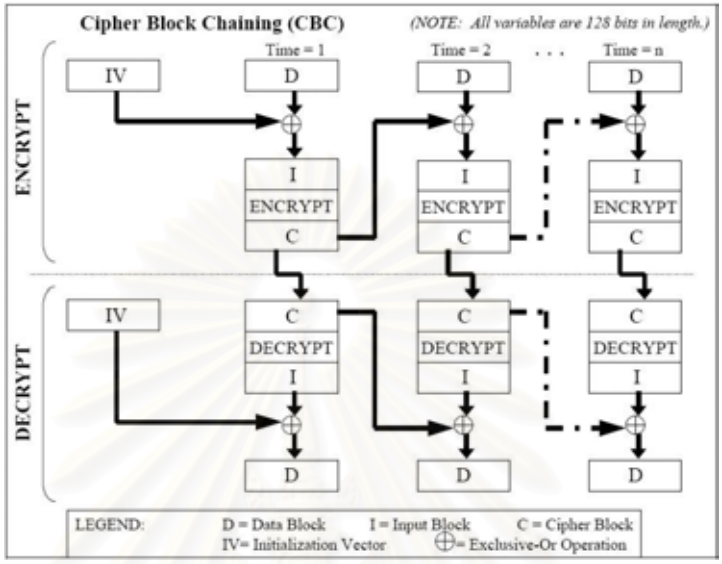
1.โหมดอีซีบี (ECB: Electronic Codebook) เป็นโหมดที่ใส่ข้อความเข้าไปโดยตรงดังในรูปที่ 6.1



รูปที่ 6.1 แผนภาพแสดงการทดสอบโหมดอีซีบี

จากรูปที่ 6.1 การเข้ารหัสจะนำอินพุตต์ (I1,I2, …,I128) จากข้อความทั้งหมด (Plain Text) 128 บิต (D1,D2, …,D128) เข้ามาประมวลผลโดยตรงแล้วส่งออกไปเป็นเอ้าท์พุตต์ (O1,O2, …,O128) ซึ่งจะถ้าเป็นการเข้ารหัสผลที่ได้คือข้อความที่เข้ารหัส (C1,C2, …,C128) เรียบร้อยแล้ว ส่วนการถอดรหัสนำอินพุตต์ (I1,I2, …,I128) จากข้อความที่เข้ารหัสแล้ว (C1,C2, …,C128) เข้ามาประมวลผลได้เป็นเอ้าท์พุตต์ (O1,O2, …,O128) แล้วส่งออกเป็นข้อความที่ถูกถอดรหัส (D1,D2, …,D128)

2.โหมดซีบีซี (CBC: Cipher Block Chaining) คือโหมดที่นำเอาอินพุตต์ที่เป็นข้อความธรรมดามาทำการเอ็กซ์ออร์กับเวกเตอร์ค่าเริ่มต้น (Initialization Vector) ก่อนเข้ารหัสดังแสดงในรูปที่ 6.2



รูปที่ 6.2 แผนภาพแสดงการทดสอบโหมดซีบีซี

จากรูปที่ 6.2 ในกระบวนการเข้ารหัสข้อมูล (D) ที่เวลาที่หนึ่งทำการเอ็กซ์ออร์กับเวกเตอร์ค่าเริ่มต้นก่อนที่ไปเป็นอินพุตต์ (I) เพื่อทำการเข้ารหัสแล้วนำข้อความที่เข้ารหัส (C) ไปใช้เป็นตัวกระทำเอ็กซ์ออร์ในรอบถัดไปและเป็นเช่นเดียวกันกับกระบวนการถอดรหัส

กระบวนการทดสอบที่ได้กล่าวมาต้องอาศัยชุดของเวกเตอร์ทดสอบซึ่งถูกจัดเก็บลงเป็นไฟล์ที่มีชื่อต่างกันเพื่อความสะดวกในเรียกการใช้งานและแต่ละไฟล์จะใช้งานทดสอบในกรณีที่ต่างกันตามตารางที่ 6.1 และ 6.2

ตารางที่ 6.1 ตารางแสดงการใช้ไฟล์ชุดเวกเตอร์ทดสอบแบบรู้คำตอบ

Filename	Mode	Test	Key Sizes *(bits)*
ecb_vk.txt	ECB	Variable Key KAT	128, 192, 256
ecb_vt.txt	ECB	Variable Text KAT	128, 192, 256
ecb_tbl.txt *(if applicable)*	ECB	Tables KAT	128, 192, 256
? (possibly multiple files) *(if applicable)*	ECB	Intermediate Values KAT	128, 192, 256

ตารางที่ 6.2 ตารางแสดงการใช้ไฟล์ชุดเวกเตอร์ทดสอบแบบมอนติคาร์โล

Filename	Mode	Test	Key Sizes *(bits)*
ecb_e_m.txt	ECB	Encrypt MCT	128, 192, 256
ecb_d_m.txt	ECB	Decrypt MCT	128, 192, 256
cbc_e_m.txt	CBC	Encrypt MCT	128, 192, 256
cbc_d_m.txt	CBC	Decrypt MCT	128, 192, 256

จากตารางที่ 6.1 จะเห็นได้ว่าเป็นการทดสอบในโหมดอีซีบีเพียงอย่างเดียวเท่านั้น

Variable Key KAT คือการทดสอบที่ใช้ชุดของเวกเตอร์ทดสอบที่มีข้อความเดียวกันทุกเวกเตอร์ทดสอบมีเพียงคีย์เท่านั้นที่แต่ละเวกเตอร์ทดสอบมีค่าแตกต่างกัน

Variable Text KAT คือการทดสอบที่ใช้ชุดของเวกเตอร์ทดสอบที่มีคีย์เดียวกันทุกเวกเตอร์ทดสอบมีเพียงข้อความเท่านั้นที่แต่ละเวกเตอร์ทดสอบมีค่าแตกต่างกัน

Table KAT คือชุดของเวกเตอร์ทดสอบที่ใช้เพื่อทดสอบตารางเอสบ็อกซ์ (S-Box) เวกเตอร์ทดสอบภายในจะถูกบังคับให้ใช้งานเอสบ็อกซ์ทุกๆค่าที่เป็นไปได้

Intermediate Value KAT คือการทดสอบค่าของคำตอบชั่วคราวที่เกิดขึ้นระหว่างการเข้ารหัส

ดังนั้นแต่ละไฟล์ใช้เพื่อทำการทดสอบดังนี้

ecb_vk.txt – เป็นไฟล์ที่เก็บชุดของเวกเตอร์ทดสอบในโหมดอีซีบีเพื่อทดสอบการเข้ารหัสแบบข้อความคงที่

ecb_vt.txt – เป็นไฟล์ที่เก็บชุดของเวกเตอร์ทดสอบในโหมดอีซีบีเพื่อทดสอบการเข้ารหัสแบบคีย์คงที่

ecb_tbl.txt – เป็นไฟล์ที่เก็บชุดของเวกเตอร์ทดสอบในโหมดอีซีบีเพื่อทดสอบตารางแทนค่าที่ใช้เข้ารหัส (ชุดของเวกเตอร์ทดสอบนี้มีไว้สำหรับการเข้ารหัสที่มีการใช้ตาราง)

ส่วนไฟล์สุดท้ายที่ไม่มีชื่อมีไว้สำหรับทดสอบคำตอบระหว่างการเข้ารหัสซึ่งผู้ทดสอบเป็นคนตั้งขึ้นมาเองซึ่งเอ็นไอเอสทีไม่ได้กำหนดมาตรฐานไว้

จากตารางที่ 6.2 จะเห็นได้ว่าการทดสอบแบบมอนติคาร์โลแยกเป็นสองประเภทคือโหมดอีซีบีและโหมดซีบีซี สองโหมดนี้ยังแยกย่อยระหว่างการเข้ารหัสและถอดรหัสด้วย ภายในไฟล์ประกอบด้วยเวกเตอร์ทดสอบจำนวน 400 เวกเตอร์ เมื่อนำมาใช้ทดสอบแบบมอนติคาร์โลแต่ละเวกเตอร์จะถูกเข้ารหัสอย่างต่อเนื่องทั้งหมด 10,000 ครั้ง การทดสอบแบบมอนติคาร์โลถูกแบ่งแยกอย่างชัดเจนว่าเข้ารหัสหรือถอดรหัสดังนั้นไฟล์ที่นำมาทดสอบจึงมีสองไฟล์ด้วยกันคือ ecb_e_m.txt และ cbc_e_m.txt ไฟล์แรกเพื่อทดสอบการเข้ารหัสด้วยมอนติคาร์โลโหมดอีซีบี ส่วนอีกอันเพื่อทดสอบการเข้ารหัสด้วยมอนติคาร์โลโหมดซีบีซี

6.1 รายละเอียดการทดสอบ

เริ่มแรกทดสอบวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้นี้โดยวิธีรู้คำตอบแบบอีซีบีด้วยไฟล์ที่มีชื่อดังต่อไปนี้

ecb_vk.txt

ecb_vt.txt

ecb_tbl.txt

ecb_iv.txt (เป็นไฟล์ที่รินเดลล์แนบเพิ่มมาด้วยเพื่อทดสอบคำตอบชั่วคราว)

การทดสอบด้วยไฟล์ทั้งสี่นี้เป็นไปในลักษณะที่ใส่อินพุตต์เข้าไปแล้วรอรับคำตอบแบบตัวต่อตัวจึงไม่ต้องดัดแปลงวงจรให้เหมาะกับการทดสอบแต่อย่างใด

ต่อจากนั้นทำการทดสอบแบบมอนติคาร์โลโดยไฟล์

ecb_e_m.txt

cbc_e_m.txt

ดังที่ได้กล่าวมาแล้วข้างต้นเกี่ยวกับการทดสอบแบบมอนติคาร์โลซึ่งมีทั้งในแบบอีซีบีและซีบีซีมีรายละเอียดดังรูปที่ 6.3 และ 6.4

```
Initialize KEY0, PT0

FOR i = 0 TO 399
 {
        Record i, KEYi, PT0
        FOR j = 0 TO 9,999
         {
                IBj = PTj
                Perform algorithm in encrypt state, resulting in CTj
                PTj+1 = CTj
         }
        Record CTj

        KEYi+1 = KEYi ⊕ last n bits of CT, where n=128, 192, or 256 (depending on key size)
        PT0 = CT9999
 }
```

รูปที่ 6.3 การทดสอบมอนติคาร์โลแบบอีซีบี

จากรูปที่ 6.3 การทดสอบจะเริ่มต้นจากรับค่าคีย์ตั้งต้น KEY_0 และข้อความตั้งต้น PT_0 เข้ามาต่อจากนั้นจะเข้าวงวนอันนอก เก็บค่าคีย์และข้อความจากนั้นเข้าสู่วงวนอันในแล้วส่งข้อความเข้าไปเป็นพารามิเตอร์สำหรับเข้ารหัส ($IB_j = PT_j$) เมื่อเข้ารหัสเสร็จแล้วก็นำข้อความเข้ารหัสที่ได้ไปใช้เป็นอินพุตต์ในครั้งต่อไป ($PT_{j+1} = CT_j$) วงวนด้านในก็จะวนไปเรื่อยๆโดยใช้ข้อความที่ถูกเข้ารหัสในครั้งก่อนมาเป็นอินพุตต์ ส่วนคีย์จะใช้คีย์เดียวกันตลอดจนจบวงวนด้านใน เมื่อวงวนด้านในเสร็จสิ้นหนึ่งรอบจะมีการเปลี่ยนคีย์โดยนำเอาคีย์เดิมมาเอ็กซ์ออร์กับข้อความสุดท้ายที่เข้ารหัสเสร็จ ($KEY_{i+1} = KEY_i$ xor CT) หลังจากนั้นส่งต่อข้อความสุดท้ายที่ได้จากวงวนด้านในมาเป็นอินพุตต์

ของวงวนด้านในรอบต่อไป ($PT_0 = CT_{9999}$) ทำเช่นนี้ไปเรื่อยๆจนหมดวงวนด้านนอกก็ถือเป็นการทดสอบเสร็จสิ้น

อย่างไรก็ตามเนื่องจากการทดสอบดังกล่าวทำเป็นวงวนไปเรื่อยๆทั้งหมด 4,000,000 ครั้ง ซึ่งไม่เหมาะกับวงจรที่ได้ออกแบบมาตอนต้นจึงจำเป็นต้องดัดแปลงวงจรเพื่อให้ทดสอบได้ การดัดแปลงดังกล่าวไม่ได้ส่งผลต่อการทำงานโดยรวมของวงจรแต่เป็นการปรับแต่งอินพุตต์ที่เข้ามาให้มีลักษณะเหมือนอย่างที่การทดสอบต้องการ วงจรจะถูกดัดแปลงให้สามารถเข้ารหัสอย่างต่อเนื่องได้โดยใช้คีย์ตัวเดิมนั่นคือสามารถทำงานของวงวนด้านในได้ แต่งานของวงวนด้านนอก (เช่น การหาคีย์ต่อไปให้วงวนด้านใน) จำเป็นต้องใช้มือเข้ามาช่วยทำเนื่องจากการดัดแปลงมากไปจะส่งผลให้เกิดความซ้ำซ้อนของงานขึ้น

```
Initialize KEY0, IV, PT0

FOR i = 0 TO 399
 {
        If (i==0) CV0 = IV
        Record i, KEYi, CV0, PT0
        FOR j = 0 TO 9,999
         {
                IBj = PTj ⊕ CVj
                Perform algorithm in encrypt state, resulting in CTj
                IF j=0
                 PTj+1 = CV0
                ELSE
                 PTj+1 = CTj-1
                CVj+1 = CTj
         }
        Record CTj

        KEYi+1 = KEYi ⊕ last n bits of CT, where n=128, 192, or 256 (depending on key size)
        PT0 = CT9998
        CV0 = CT9999
}
```

รูปที่ 6.4 การทดสอบมอนติคาร์โลแบบซีบีซี

การทดสอบในแบบซีบีซีจะมีหลักการคล้ายๆกับอีซีบีแตกต่างกันเพียงแค่การส่งอินพุตต์ไปยังวงจรเข้ารหัสต้องมีการนำข้อความไปเอ็กซ์ออร์กันข้อความที่ถูกเข้ารหัสในรอบที่แล้ว ($IB_j = PT_j$ xor CV_j) ส่วนของโค้ดที่เพิ่มมาเป็นเพียงรายละเอียดปลีกย่อยที่ในการหาข้อความที่ถูกเข้ารหัสในรอบก่อนจึงไม่ขอกล่าวถึง

การทดสอบวิธีนี้จำเป็นต้องดัดแปลงวงจรที่ซับซ้อนกว่าเดิมเนื่องจากมีการเปลี่ยนแปลงอินพุตต์ระหว่างวงวนรอบเล็กและการเปลี่ยนแปลงดังกล่าวต้องมีความเกี่ยวข้องกับข้อความในครั้งที่ผ่านมาด้วย ดังนั้นการทดสอบในแบบซีบีซีต้องเพิ่มส่วนของวงจรเข้าไปเพื่อตั้งค่าตั้งต้นใหม่ในทุกๆรอบผลการทดสอบแบบมอนติคาร์โลของทั้งสองกรณีแสดงไว้ในตารางที่6.3

6.2 ผลการทดสอบ

การทดสอบทั้งสี่เป็นไปอย่างถูกต้อง สามารถสรุปได้เป็นตารางที่ 6.3

ตารางที่ 6.3 ตารางแสดงความถูกต้องในการทดสอบการเข้ารหัส

ไฟล์ที่ใช้ทดสอบ	ความถูกต้อง(ร้อยละ)
ecb_vk.txt	100
ecb_vt.txt	100
ecb_tbl.txt	100
ecb_iv.txt	100
ecb_e_m.txt*	100
cbc_e_m.txt*	100

*เป็นการสุ่มทดสอบ เนื่องจากมีเวกเตอร์ทดสอบเป็นจำนวนมาก

จากตารางที่ 6.3 ทำให้สามารถสรุปได้ว่าวงจรเข้ารหัสแบบเปลี่ยนโครงแบบได้ทำงานได้อย่างถูกต้องตรงตามความต้องการของการเข้ารหัสแบบเออีเอสทุกประการ

บทที่ 7
สรุปผลการวิจัย

7.1 ผลการวิจัย

วงจรเข้ารหัสเออีเอสแบบเปลี่ยนโครงแบบได้เป็นแนวทางหนึ่งที่สามารถนำไปใช้กับงานประเภทที่ต้องการทรัพยากรอย่างจำกัดเช่น นำไปใช้กับระบบฝังตัว เป็นต้น เนื่องจากมีขนาดที่เล็กและมีคุณสมบัติที่เป็นข้อได้เปรียบจากการเปลี่ยนโครงแบบได้จึงสามารถเปลี่ยนโครงแบบตัวเองไปเป็นวงจรย่อยแต่ละส่วนเพื่อทำงานย่อยๆ ดังนั้นขนาดของวงจรรวม ณ ขณะใดขณะหนึ่งย่อมมีขนาดไม่เกินวงจรย่อยที่มีขนาดใหญ่ที่สุดในระบบ ด้านความเร็วในการประมวลผลถึงแม้งานวิจัยชิ้นนี้จะไม่เน้นจุดนั้นก็ตามแต่ก็ได้พยายามให้อยู่ในเกณฑ์ที่พอเป็นที่น่าพอใจ

7.2 ข้อเสนอแนะ

เนื่องด้วยเทคโนโลยีในตอนนี้ยังไม่สนับสนุนการใช้วิธีเปลี่ยนโครงแบบด้วยตัวเอง การทำงานวิจัยชิ้นนี้จำได้อาศัยการจำลองวงจรเพียงเท่านั้น อย่างไรก็ตามคาดว่าอนาคตอันใกล้นี้เทคโนโลยีดังกล่าวจะถูกพัฒนาให้สามารถใช้งานได้อย่างสมบูรณ์ และหวังว่างานวิจัยชิ้นนี้ก็จะเป็นประโยชน์ต่อวงจรเปลี่ยนโครงแบบดังกล่าว

รายการอ้างอิง

[1] Fernando, S. and H. Yajun. Design of Networked Reconfigurable Encryption Engine. The 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machine (2005)

[2] McMillan, S. and C. Patterson. JBits™ Implementations of The Advance Encryption Standard (Rijndael). Lecture Notes in Computer Science, 162-171 Berlin/Heldeiberg : Springer-Verlag, 2001.

[3] Edson, L.H., W.L. John, and T.K. Se'rgio. Using PARBIT to Implement Partial Run-Time Reconfigurable Systems. Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream 12th International Conference (2002).

[4] Katherine, C. and H. Scott. Reconfigurable computing: a survey of systems and software. ACM Computer. Survey. (2002): 171-210.

[5] Chaves, R., G. Kuzmanov, S. V. and L. Sousa. Reconfigurable Memory Based AES Co-Processor. Parallel and Distributed Processing Symposium 20th International (2006).

[6] Kwok, T. T. and Y. Kwok. On the Design of a Self-Reconfigurable SoPC Based Cryptographic Engine. The 24th International Conference on Distributed Computing Systems Workshops (2004).

[7] Su, C., C. Horng, C. Huang and C. Wu, A configurable AES Processor for Enhanced for Security. The 12th Asia and South Pacific Design Automation Conferrence (2005): 361-366.

[8] Piromsopa, K., P. Bavonparadon and P.Chongstitvatana. Hardware multiplexing: towards a resource efficient reconfigurable processor. The 3rd International Symposium on Communications and Information Technologies (2003).

[9] NIST. Annoucing the advanced encryption standard (AES), FIPS 197. National Institute of Standards and Technology Technical report (2001).

[10] Jian, X., L. Yuan-feng, D. Zi-bin and S. Yi. Design and Implementation of reconfigurable AES IP Core using FPGAs. The 6th International Conference On ASIC (2005): 765-765.

[11] Pionteck, T., T. Staake, T. Stiefmeier, L.D. Kabulepa, M. Glesner. Design of a Reconfigurable AES encryption/decryption engine for mobile terminals. The 2004 International Symposium onCircuits and Systems (2004): 545-548.

[12] Usselmann, R.. http://www.opencores.org. opencores, R. Usselmann, 2002

[13] Purna, K.M.G,. and D. Bhatia. Temporal partitioning and scheduling data flow graphs for reconfigurable computers. IEEE Transactions on Computers (1999): 579-590.

[14] NIST, http://csrc.nist.gov/CryptoToolkit/aes/. Computer Security Resource Center (CSRC), 1998.

ประวัติผู้เขียนวิทยานิพนธ์

นายเจนโชติ ศรีพรประเสริฐ เกิดเมื่อวันที่ 28 มกราคม พ.ศ. 2526 ที่จังหวัดกรุงเทพฯ สำเร็จการศึกษาระดับประถมศึกษา ถึงมัธยมศึกษาตอนปลายจากโรงเรียนกรุงเทพคริสเตียนวิทยาลัย สำเร็จการศึกษาระดับปริญญาบัณฑิต ในสาขาวิชาวิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยในปีการศึกษา 2548