การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับ

CryptoChip Application Prototype

นายทรงวิทย์ เตชะราชันย์
นางสาวทรงศิริ โตเลี้ยง
นายทวีศักดิ์ เจริญรุ่งสกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2548

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับ

CryptoChip Application Prototype

โดย

นายทรงวิทย์ เตชะราชันย์

นางสาวทรงศิริ โตเลี้ยง

นายทวีศักดิ์ เจริญรุ่งสกุล

อาจารย์ที่ปรึกษา

อาจารย์อัครเดช วัชระภูพงษ์

ผศ.ธนา หงส์สุวรรณ

อาจารย์ธนัญชัย ตรีภาค

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับ

CryptoChip Application Prototype

ผู้จัดทำ

1. นายทรงวิทย์ เตชะราชันย์ รหัสนักศึกษา 45010291
2. นางสาวทรงศิริ โตเลี้ยง รหัสนักศึกษา 45010293
3. นายทวีศักดิ์ เจริญรุ่งสกุล รหัสนักศึกษา 45010294

_____ อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชระภูพงษ์)

_____ อาจารย์ที่ปรึกษา

(ผศ.ธนา หงษ์สุวรรณ)

_____ อาจารย์ที่ปรึกษา

(อาจารย์ธนัญชัย ตรีภาค)

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับ

นายทรงวิทย์	เตชะราชันย์	45010291
นางสาวทรงศิริ	โตเลี้ยง	45010293
นายทวีศักดิ์	เจริญรุ่งสกุล	45010294
อาจารย์ อัครเดช	วัชระภูพงษ์	อาจารย์ที่ปรึกษา
ผศ. ธนา	หงษ์สุวรรณ	อาจารย์ที่ปรึกษาร่วม
อาจารย์ ธนัญชัย	ตรีภาค	อาจารย์ที่ปรึกษาร่วม

ปีการศึกษา 2548

บทคัดย่อ

การสื่อสารข้อมูลในปัจจุบันนี้จำเป็นต้องพึ่งพากรรมวิธีการเข้าและถอดรหัสลับอันสลับซับซ้อนยิ่งขึ้นเรื่อยๆ ฉะนั้นการเข้าและถอดรหัสลับด้วยฮาร์ดแวร์จึงเป็นสิ่งที่หลีกเลี่ยงไม่ได้ แม้ว่ากรรมวิธีการเข้าและถอดรหัสลับจะเป็นที่ทราบและมีซอร์สโค้ดทั่วไปก็ตาม หากแต่เมื่อแปลงมาใช้งานกับฮาร์ดแวร์ต้องพิจารณาถึงข้อเด่นและข้อด้อยของแพลตฟอร์มฮาร์ดแวร์นั้นด้วย เพื่อให้มีประสิทธิภาพการทำงานสูงสุด

โครงการนี้อาศัยโค้ดเอชดีแอลและอัลกอริทึม Secret Key Cryptography เป็นการเข้าและถอดรหัสลับพื้นฐานบนฮาร์ดแวร์ที่กำหนดพร้อมพัฒนาซอฟต์แวร์ไดรเวอร์หรือไลบรารีที่จำเป็น จากนั้นจึงพัฒนางานตัวอย่างเพื่อพิสูจน์ความสามารถในการใช้งานได้จริง

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับ

นายทรงวิทย์	เตชะราชันย์	45010291
นางสาวทรงศิริ	โตเลี้ยง	45010293
นายทวีศักดิ์	เจริญรุ่งสกุล	45010294
อาจารย์ อัครเดช	วัชระภูพงษ์	อาจารย์ที่ปรึกษา
ผศ. ธนา	หงษ์สุวรรณ	อาจารย์ที่ปรึกษาร่วม
อาจารย์ ธนัญชัย	ตรีภาค	อาจารย์ที่ปรึกษาร่วม

ปีการศึกษา 2548

บทคัดย่อ

การสื่อสารข้อมูลในปัจจุบันนี้จำเป็นต้องพึ่งพากรรมวิธีการเข้าและถอดรหัสลับอันสลับซับซ้อนยิ่งขึ้นเรื่อยๆ ฉะนั้นการเข้าและถอดรหัสลับด้วยฮาร์ดแวร์จึงเป็นสิ่งที่หลีกเลี่ยงไม่ได้ แม้ว่ากรรมวิธีการเข้าและถอดรหัสลับจะเป็นที่ทราบและมีซอร์สโค้ดทั่วไปก็ตาม หากแต่เมื่อแปลงมาใช้งานกับฮาร์ดแวร์ต้องพิจารณาถึงข้อเด่นและข้อด้อยของแพลตฟอร์มฮาร์ดแวร์นั้นด้วย เพื่อให้มีประสิทธิภาพการทำงานสูงสุด

โครงการนี้อาศัยโค้ดเอชดีแอลและอัลกอริทึม Secret Key Cryptography เป็นการเข้าและถอดรหัสลับพื้นฐานบนฮาร์ดแวร์ที่กำหนดพร้อมพัฒนาซอฟต์แวร์ไดรเวอร์หรือไลบรารีที่จำเป็น จากนั้นจึงพัฒนางานตัวอย่างเพื่อพิสูจน์ความสามารถในการใช้งานได้จริง

CryptoChip Application Prototype

Mr. Songwit	Techarachan	45010291
Miss Songsiri	Tolieng	45010293
Mr. Taweesak	Charoenrungsakul	45010294
Mr. Akkradach	Watcharapupong	Advisor
Asst.Prof. Thana	Hongsuwan	Co-Advisor
Mr. Thananchai	Treepark	Co-Advisor

Academic Year 2005

ABSTRACT

Nowadays, the data communication necessarily depends on more complicated Cryptography in order to meet the most effectiveness of secure data communication. This causes Hardware cryptography is incontrovertible. Although the process of cryptography is pervasive, and there are ubiquitous source codes, developers or programmers should consider advantages and drawbacks of hardware platform when they adapt cryptography to operate on the hardware.

This project is conducted by using HDL codes and Secret Key Cryptography algorithms in process of encryption and decryption on the designated hardware, and software drivers and necessary libraries are simultaneously developed on the designated hardware before testing the practical implement.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงมิอาจเสร็จสมบูรณ์ไปได้หากไม่ได้รับความช่วยเหลือจากหลาย ๆ ฝ่ายด้วยกัน

ขอบคุณอาจารย์อัครเดช วัชระภูพงษ์ - พี่เค้ก สำหรับคำปรึกษา แนวทางการแก้ไขปัญหาต่าง ๆ การเอนเตอร์เทนต่าง ๆ ที่มอบให้เช่น คำว่า "Create" และน้ำเปล่าสวีเดนกับชาเขียวยี่ห้อต่าง ๆ ซึ่งมีส่วนช่วยในโปรเจ็คเป็นอย่างมาก

ขอขอบคุณเพื่อน ๆ พี่ ๆ น้อง ๆ สมาชิกห้องวิจัย ISAG ทุกคนสำหรับสิ่งดี ๆ ต่าง ๆ ไม่ว่าจะเป็นความรู้ ความสนุกสนาน ขอขอบคุณพี่ก่อ สำหรับเครื่องคอมพิวเตอร์ (ของพี่ตั้ม) ที่มอบให้ในสำหรับทำโปรเจ็ค ขอบคุณพี่บอมบ์สำหรับคำสั่งสอน ถึงแม้ว่าจะเป็นช่วงเวลาสั้น ๆ

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ สำหรับไฟฟ้า น้ำประชา ที่พักอาศัย ที่ศึกษาความรู้ต่าง ๆ และอินเตอร์เน็ต

ขอบคุณพ่อ แม่ และบุคคลทางบ้านทุกคน ที่ช่วยส่งกำลังใจ ความห่วงใย กำลังทรัพย์ มาให้อย่างสม่ำเสมอ

สุดท้ายนี้ ขอขอบคุณคณะผู้จัด ผู้ที่ร่วมทุกข์ ร่วมสุข มาด้วยกันตลอด

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นายทรงวิทย์ เตชะราชันย์
นางสาวทรงศิริ โตเลี้ยง
นายทวีศักดิ์ เจริญรุ่งสกุล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

เนื่องจากปัจจุบันการใช้งานในเครือข่ายนั้นมีความไม่ปลอดภัยสูง การส่งข้อมูลใดๆผ่านเครือข่ายย่อมมีความเสี่ยงที่จะถูกดักฟังข้อมูลหรือทำการเปลี่ยนแปลงข้อมูล ดังนั้นการเข้ารหัสลับจึงเป็นอีกวิธีหนึ่งที่นำมาใช้ในการส่งข้อมูลผ่านเครือข่าย เพื่อให้ผู้ที่ดักฟังข้อมูลไม่สามารถเข้าใจได้ พร้อมทั้งต้องมีการตรวจสอบข้อมูลว่าการส่งข้อมูลนั้นไม่ได้ถูกปลอมแปลงด้วยการใช้การเข้ารหัสทางเดียว

การเข้าและถอดรหัสลับนั้นนับวันก็เพิ่มความซับซ้อนมากขึ้นเรื่อยๆเพื่อเป็นการเพิ่มความมั่นใจในความปลอดภัย ดังนั้นการเข้ารหัสลับด้วยซอฟต์แวร์ที่แพร่หลายทั่วไปทางอินเตอร์เน็ตนั้นทีความง่าย แต่ก็มีข้อเสียที่สำคัญคือเรื่องของเวลาที่การใช้ซอฟต์แวร์นั้นใช้เวลาในการเข้าและถอดรหัสลับนานมากต่างจากการใช้ฮาร์ดแวร์จึงเป็นหลักสำคัญในการทำโปรเจ็คนี้ขึ้นมา และการที่เราใช้ฮาร์ดแวร์ยังสามารถเพิ่มความสามารถต่างๆได้มากยิ่งขึ้น

1.2 วัตถุประสงค์ของโครงงาน

1. สามารถทำการเข้าและถอดรหัสลับข้อมูลผ่านทางฮาร์ดแวร์ได้อย่างถูกต้อง
2. ศึกษาการเขียนโปรแกรมดีไวซ์ไดรเวอร์บนระบบปฏิบัติการลีนุกซ์
3. ศึกษาและพัฒนาต้นแบบชิปเข้าและถอดรหัสลับร่วมกับอุปกรณ์
4. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบคอมพิวเตอร์บุคคลได้
5. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบปฏิบัติการได้
6. ทำการสุ่มเลขเทียมโดยฮาร์ดแวร์ได้อย่างมีประสิทธิภาพ
7. ทำการเข้ารหัสทางเดียวโดยฮาร์ดแวร์ได้

1.3 ขอบเขตของโครงงาน

1. สามารถทำการเข้าและถอดรหัสลับข้อมูลโดยฮาร์ดแวร์ได้อย่างถูกต้อง
2. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบคอมพิวเตอร์บุคคลได้
3. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบปฏิบัติการได้
4. สามารถการสุ่มเทียมโดยฮาร์ดแวร์ได้อย่างมีประสิทธิภาพ

5. สามารถทำการเข้ารหัสทางเดียวโดยฮาร์ดแวร์ได้

1.4 วิธีการดำเนินการ

1. กำหนดขอบเขตของโครงงาน โดยดูจากข้อกำหนดต่าง ๆ
2. ออกแบบระบบโดยรวมก่อนว่าต้องมีอะไรมาประกอบกันบ้าง
3. แบ่งส่วนงานต่าง ๆ ออกเป็น 3 ส่วนหลัก ๆ ที่สามารถทำงานไปพร้อม ๆ กันได้ โดยประกอบด้วย ส่วนของฮาร์ดแวร์ ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์ และส่วนของอัลกอริทึมที่ใช้
4. ส่วนของฮาร์ดแวร์นั้นจะดูเรื่องของการเขียนโปรแกรมด้วยภาษาวีเอชดีเอลและการเลือกการ์ดที่จะนำมาใช้รวมถึงการศึกษาการใช้งานการ์ดนั้น ๆ
5. ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์นั้นจะดูเริ่มจากการเขียนภาษาซีบนลีนุกซ์ก่อนแล้วจึงเริ่มศึกษาการเขียนซอฟต์แวร์ดีไวซ์ไดรเวอร์ที่เป็นคาแร็กเตอร์ดีไวซ์และพีซีไอไดรเวอร์
6. ศึกษาอัลกอริทึมต่าง ๆ ที่จะนำมาใช้งาน โดยเริ่มต้นจากการสุ่มเลขเทียม ซึ่งต้องดูว่าแต่ละตัวนั้นมีข้อดูข้อเสียอย่างไรและสามารถนำมาใช้กับฮาร์ดแวร์ได้จริงไหม
7. เขียนโค้ดเอชดีเอลส่วนของการสุ่มเลขเทียมลงบนการ์ด
8. เขียนโค้ดส่วนของดีไวซ์ไดรเวอร์ของการสุ่มเลขเทียม
9. ศึกษาอัลกอริทึมเรื่องของการทำแฮช แล้วอัลกอริทึมที่สามารถใช้งานได้มาใช้
10. เขียนโค้ดเอชดีเอลส่วนของแฮชฟังก์ชัน
11. เขียนโค้ดส่วนของดีไวซ์ไดรเวอร์ของแฮช
12. ศึกษาอัลกอริทึมเรื่องของการเข้าและถอดรหัสลับ แบบต่าง ๆ แล้วเลือกที่สามารถใช้ได้มาทำ
13. เขียนโค้ดเอชดีเอลส่วนของการเข้าและถอดรหัสลับ
14. เขียนโค้ดส่วนของดีไวซ์ไดรเวอร์ของการเข้าและถอดรหัสลับ
15. เขียนโปรแกรมขึ้นมาเพื่อทดสอบการทำงานของระบบส่วนต่าง ๆ
16. ตรวจสอบส่วนต่าง ๆ ของระบบว่ามีข้อผิดพลาดหรือต้องปรับปรุงไหม
17. แก้ไขข้อผิดพลาดต่าง ๆ เพื่อให้ระบบมีเสถียรภาพในการทำงาน

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ ความเข้าใจเกี่ยวกับกระบวนการทำการสุ่มเลขเทียม
2. ได้รับความรู้ ความเข้าใจเกี่ยวกับกระบวนการเข้ารหัสทางเดียว

3. ได้รับความรู้ ความเข้าใจเกี่ยวกับกระบวนการเข้าและถอดรหัสลับ
4. ได้รับความรู้ ความเข้าใจเกี่ยวกับการเขียนดีไวซ์ไดรเวอร์
5. สามารถสร้างเป็นชิพที่สามารถทำการเข้าและถอดรหัสลับได้โดยมีการสุ่มเลขเทียม, การสร้างเลขสุ่มเทียม และมีการทำการเข้ารหัสทางเดียวด้วย

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 8 บทด้วยกันคือ

บทที่ 1 กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับ และส่วนประกอบของปริญญานิพนธ์

บทที่ 2 ได้อธิบายถึงวิธีการทำงานของการสุ่มเลขเทียม การแปลงข้อมูลและเข้ารหัสลับในวิธีต่าง ๆ รวมถึงการนำไปใช้งาน

บทที่ 3 กล่าวถึงเทคโนโลยีเอฟพีจีเอ ว่ามีแบบไหนบ้าง สถาปัตยกรรมภายใน การทำงาน คุณสมบัติต่าง ๆ

บทที่ 4 กล่าวถึงการเขียนลีนุกซ์ดีไวซ์ไดรเวอร์ ซึ่งอธิบายถึงไดรเวอร์ชนิดต่าง ๆ และการเขียนโปรแกรมดีไวซ์ไดรเวอร์ที่เป็นคาแร็กเตอร์ดีไวซ์ ฟังก์ชันต่าง ๆ ที่จำเป็นต้องใช้

บทที่ 5 กล่าวถึงการเขียนไดรเวอร์สำหรับพีซีไอการ์ดบนระบบปฏิบัติการลีนุกซ์ ประกอบไปด้วยการโครงสร้างของพีซีไอไดรเวอร์ ขั้นตอนต่าง ๆ ที่จำเป็นต้องทำในการเขียนโปรแกรมดีไวซ์ไดรเวอร์

บทที่ 6 กล่าวถึงการออกแบบ จะแสดงให้เห็นถึงภาพรวมของระบบ แล้วอธิบายลงลึกในแต่ละส่วนทั้งในส่วนของฮาร์ดแวร์และซอฟต์แวร์

บทที่ 7 เป็นผลการทดสอบระบบ นำภาพผลลัพธ์จากการทำงานของระบบมาเปรียบเทียบกับผลลัพธ์ที่ควรจะได้ตามทฤษฎี

บทที่ 8 เป็นบทวิจารณ์และสรุป ซึ่งกล่าวถึงบทสรุปของโครงงาน วิจารณ์สิ่งที่ได้รับจากโครงงาน และข้อเสนอแนะสำหรับเป็นแนวทางในการพัฒนาต่อ

บทที่ 2
การสุ่มเลขเทียม การแปลงข้อมูลและเข้ารหัสลับ

ในการทำโปรเจ็คนี้ต้องอาศัยความรู้จากสามส่วนด้วยกันคือส่วนของทฤษฎีการเข้าและถอดรหัสลับ ซึ่งก็มีสามส่วนคือส่วนเลขสุ่มเทียม ส่วนของการเข้ารหัสทางเดียว และส่วนของเข้าและถอดรหัสลับ ทำการศึกษาดูว่าทฤษฎีนี้มีข้อดีและข้อเสียอย่างไรเพื่อหาว่าทฤษฎีไหนที่เหมาะสมกับโปรเจ็คของเรา ถัดจากส่วนของการเข้าและถอดรหัสก็มีส่วนของซอร์ฟแวร์ดีไวซ์ไดรเวอร์ซึ่งเราต้องเขียนขึ้นมาใหม่ให้สามารถใช้กับฮาร์ดแวร์ที่เราสร้างขึ้นมาเอง ต้องมีการศึกษาว่าจะติดต่อกับฮาร์ดแวร์อย่างไร ไม่ว่าจะเป็นเรื่องของการรับส่งข้อมูลระหว่างฮาร์ดแวร์กับซอฟต์แวร์แอพพลิเคชัน และสุดท้ายส่วนของฮาร์ดแวร์เป็นส่วนศึกษาการเขียนภาษาเอชดีเอลเพื่อให้สามารถทำให้การ์ดทำงานได้และยังต้องศึกษาการทำงานของพีซีไอด้วย

2.1 ทฤษฎีทางด้านการสุ่มเลขเทียม

การสุ่มเลขนั้นมีประโยชน์ในการนำมาเพิ่มความปลอดภัยให้กับกุญแจที่เราใช้ในการเข้ารหัสลับ ซึ่งโดยปกติแล้วเป็นอัลกอริธึมที่นำมาใช้ เพื่อเพิ่มประสิทธิภาพด้านความปลอดภัยให้กับส่วนต่างๆ เช่น กุญแจที่ใช้ในการเข้ารหัสลับของเรา ค่าเวกเตอร์ตั้งต้น หมายเลขเซสชั่น (session id) โดยจะเป็นการเพิ่มความยากให้กับผู้ที่ไม่หวังดี (ที่อาจอยากเปลี่ยนแปลงข้อมูลหรือขโมยข้อมูล) ในการคาดเดาค่าต่างๆ โดยการสุ่มตัวเลข (random number) นั้นมีทั้งแบบการสุ่มเลขแบบแท้จริง (truly- random number) และการสุ่มเลขเทียม (pscudo random number) โดยการสุ่มเลขแบบแท้จริงนั้นเป็นการสุ่มที่ไม่มีแบบแผนใด ๆ ที่จะทำให้สามารถคาดเดาเลขต่อไปได้ว่าจะเป็นค่าอะไร ซึ่งทำให้เราก็ไม่อาจแน่ใจได้ว่าเลขที่ได้ออกมาจากการสุ่มนั้นจะเป็นค่าการสุ่มตัวเลขที่เหมาะสมหรือเปล่า(เลขที่ออกมาอาจมีค่าซ้ำกัน) ซึ่งโดยปกติทั่วไปแล้วการสุ่มอย่างแท้จริงนั้นจะเป็นการเอาค่ามาจากส่วนของฮาร์ดแวร์ ตัวอย่างเช่น การสุ่มเลขที่ได้จากเวลาของจังหวะในการพิมพ์ (key stroke) ค่าจากหมายเลขของกระบวนการที่กำลังทำงานอยู่ (running process id) ค่าจากนาฬิกา โดยค่าที่ได้มาเหล่านี้เราก็ไม่อาจแน่ใจได้อยู่ดีว่าจะสามารถหาแหล่งที่จะให้ค่าที่เป็นการสุ่มอย่างแท้จริงตามที่ได้กล่าวมาแล้ว ซึ่งบางทีการเลือกค่าตั้งต้นในการสุ่มเลขเทียมนั้นจะได้จากค่าการสุ่มเลขจริง

คุณสมบัติของการสุ่มเลขแบบแท้จริงคือ

1. ต้องไม่มีความโน้มเอียง คือตัวเลขทุกๆตัวมีความเป็นไปได้ในการเป็นผลลัพธ์ตัวต่อไปเท่าๆกัน

2. ต้องไม่สามารถคาดเดาได้ คือ ไม่สามารถคาดเดาผลลัพธ์ตัวต่อไปได้ว่าจะเป็นค่าใดแม้ว่าจะรู้ผลลัพธ์ที่ผ่านมาก็ไม่สามารถรู้ค่าต่อไปได้

3. ต้องไม่สามารถให้ผลลัพธ์ในรูปแบบเดียวกันคือหากว่าเรามีการสุ่มตัวเลขสองชุด โดยมีการใช้เงื่อนไขในการเริ่มต้นที่เหมือนกัน ก็จะได้ค่าตัวเลขที่ออกมาต่างกัน

เนื่องจากข้อจำกัดที่ว่าการสุ่มอย่างแท้จริงนั้นไม่อาจแน่ใจได้ว่าเป็นค่าที่ได้ออกมาเป็นค่าที่ควรได้จากการสุ่มที่ดีหรือเปล่า ดังนั้นจึงมีการใช้การสุ่มเลขเทียม (pseudo random number) เข้ามาแทน โดยการสุ่มเลขเทียมนั้นเป็นอัลกอริธึมที่สามารถรู้ค่าต่อไปได้ (deterministic algorithm) ซึ่งแน่นอนว่าเลขที่ได้ออกมานั้นเป็นการใช้การคำนวณค่าทางคณิตศาสตร์ออกมา

คุณสมบัติของการสุ่มเลขเทียมคือ

1. จะต้องมีค่าตั้งต้น (seed) เป็นอินพุทในการทำงานถึงจะให้ค่าผลลัพธ์ออกมา

2. ลำดับของตัวเลขที่ได้ออกมานั้นเป็นผลต่อเนื่องที่ได้จากค่าตั้งต้น ดังนั้นถ้าเราใช้ค่าตั้งต้นเป็นค่าเดิม ลำดับของเลขที่เป็นผลลัพธ์จะต้องเป็นค่าเดิมอยู่ในลำดับเดิม

3. เนื่องจากเป็นผลลัพธ์ที่ได้มาจากการคำนวณดังนั้นย่อมรู้ได้ว่าค่าที่ออกมาตัวต่อไปจะเป็นค่าใด

ค่าตั้งต้น(seed) คือ ค่าที่เป็นเหมือนอินพุทให้กับการทำงาน ซึ่งเราจะกำหนดค่านี้ในครั้งแรกที่เริ่มใช้งานเท่านั้นหลังจากนั้นก็จะได้ผลลัพธ์เป็นตัวเลขออกมาเรื่อย ๆ จนกว่าการสุ่มเลขออกมานั้นจะครบจำนวนเท่าที่อัลกอริธึมนั้นรองรับได้ หรือเมื่อต้องการให้มีการกำหนดค่าเริ่มต้นให้ใหม่

ลักษณะของการสุ่มเลขเทียมที่ดีคือ

1. ถ้าเราพิจารณาจากลำดับของตัวเลขที่เป็นผลลัพธ์ออกมานั้น จะต้องไม่สามารถหาความสัมพันธ์ของผลลัพธ์นั้นเพื่อที่จะเอาไปใช้ในการคาดเดาตัวเลขที่เป็นผลลัพธ์ตัวต่อไปได้

2. ตัวเลขซึ่งคือผลลัพธ์ที่ออกมานั้นจะต้องมีค่าไม่ซ้ำกันในช่วงที่กำหนดซึ่งช่วงจะกว้างขนาดไหนขึ้นอยู่กับอัลกอริธึมที่เลือกใช้

3. ช่วงความยาวที่ให้ค่าไม่ซ้ำกันจะต้องมีความยาวมากพอสมควรเพื่อทำให้ค่าที่ออกมาไม่ต้องมีการเริ่มใส่ค่าตั้งต้นใหม่ ไม่ต้องกลัวว่าจะเกิดค่าที่ซ้ำกัน หากว่าช่วงความยาวของรอบสั้นก็ต้องเริ่มใหม่อาจให้ค่าที่ซ้ำ ๆ เดิมออกมา ตัวอย่างลักษณะที่ไม่ดีที่จะทำให้ข้อผิดพลาดในการทำเลขสุ่มเทียม

ตัวอย่างลักษณะที่ไม่ดีที่จะทำให้ข้อผิดพลาดในการทำเลขสุ่มเทียม

1. การเลือกใช้ค่าตั้งต้นที่มีขนาดเล็กไป(จำนวนบิตน้อยไป) เช่นการเลือกค่าตั้งต้นที่มีขนาด 16 บิตเพื่อนำมาใช้เป็นกุญแจในการเข้ารหัสลับตามอัลกอริธึมที่เลือก ปัญหาคือ ค่ากุญแจ ที่เป็นไปได้เพียงแค่ 65536 ค่าเท่านั้นที่สามารถเป็นไปได้ ซึ่งเป็นค่าที่ถือได้ว่ามีจำนวนน้อยในการที่จะค้นหา

ค่าที่เป็นไปได้(เนื่องจากค่าที่เป็นไปได้มีจำนวนน้อยก็หมายความว่าสามารถหาค่าที่เป็นไปได้ในช่วงเวลาที่สั้นลง)

2. การเลือกค่าตั้งต้นโดยใช้ค่าจากนาฬิกา ซึ่งหากพิจารณาว่าหน่วยที่ใช้คือหน่วยของวินาที ดังนั้นค่าที่เป็นไปได้ก็จะมีทั้งหมด 60x60x60 เท่ากับ 216000 ค่าที่เป็นไปได้ซึ่งเป็นจำนวนที่ไม่มากเลย

3. การเลือกค่าตั้งต้นที่มีความยาวน้อยไปย่อมส่งผลให้มีจำนวนค่าที่มีความเป็นไปได้น้อยลง สามารถคำนวณหรือคาดเดาได้ง่ายขึ้น

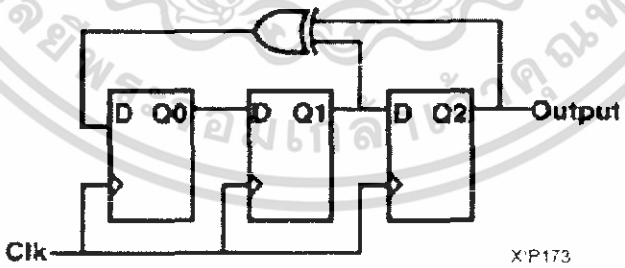
อัลกอริธึมสำหรับทำการสุ่มเทียมนั้นมีอยู่มากมายหลายอัลกอริธึมเช่น linear congruential generators, lagged Fibonacci generators, Linear feedback shift registers, generalised feedback shift registers, Blum Blum Shub, Mersenne twister, Fortuna

2.2 อัลกอริธึม Linear feedback shift registers

อัลกอริธึมแบบสุ่มเลขเทียมที่เป็นชิพรีจิสเตอร์โดยบิตที่เป็นค่าอินพุทนั้นเป็นค่าที่ถูกบังคับจากฟังก์ชันลิเนียร์ (linear function) โดยฟังก์ชันลิเนียร์ส่วนใหญ่ที่ใช้ในการทำงานของอัลกอริธึมนี้จะเป็น XORหรือ XNOR จากตำแหน่งแท็ปของค่าเริ่มต้น ค่าเริ่มต้นที่ใช้จะเรียกว่าค่าตั้งต้น(seed) ซึ่งมีข้อยกเว้นคือไม่ควรใช้ค่าตั้งต้นเป็น 0 เนื่องจากจะให้ค่าผลลัพธ์ออกมาเป็น 0 ทำให้สามารถคาดเดาค่าตั้งต้นได้ อัลกอริธึม LFSR นี้สามารถทำงานได้อย่างรวดเร็ว

อัลกิริธึมที่เลือกใช้ในโครงการได้ทำเป็น LFSR ขนาด 128 บิต ซึ่ง LFSR สามารถแบ่งออกได้เป็น 2 แบบคือแบบ Fibonacci และแบบ Galois โดยแบบ

1. Fibonacci ลักษณะการทำงานคือ ค่าอินพุทจะได้จากการเอ็กซ์คลูซีพออร์ค่าเอาท์พุทฟีดแบคกลับไป ดังตัวอย่างข้างล่างเป็นรูปการทำงานของ Fibonacci LFSR ขนาด 3 บิต



รูปที่ 2.1 แสดง LFSR แบบ Fibonacci ขนาด 3 บิต

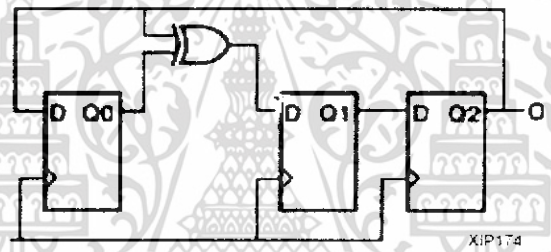
ซึ่งขั้นตอนการทำงานนั้น จะทำการโหลดค่าตั้ง ซึ่งข้อยกเว้นคือห้ามเป็นค่าศูนย์ ตัวอย่างของผลลัพธ์ที่ได้ออกมาเป็นค่า Q2 จากตาราง ตัวอย่างผลลัพธ์เป็นดังตารางนี้

Q2	Q1	Q0
1	1	1
1	1	0
1	0	0
0	0	1
0	1	0
1	0	1
0	1	1
1	1	1

รูปที่ 2.2 แสดงค่าผลลัพธ์ที่ได้จาก LFSR แบบ Fibonacci ขนาด 3 บิต

ซึ่งลำดับการสุ่มตัวเลขที่ได้ออกมานั้นสามารถใช้เอ็กซ์คลูซีฟนอร์ได้ด้วยซึ่งในกรณีนี้ค่าตั้งต้น (seed) ไม่ควรจะเป็น 1 ทุกบิต

2. Galois ลักษณะการทำงานเป็นดังรูปข้างล่าง ซึ่งเป็น LFSR ขนาด 3 บิตเช่นกัน



รูปที่ 2.3 แสดง LFSR แบบ Galois ขนาด 3 บิต

ตารางข้างล่างนี้แสดงค่าเอาท์พุทที่ได้จากการชิพ 1 รอบ

Q2	Q1	Q0
1	1	1
1	0	1
0	0	1
0	1	0
1	0	0
0	1	1
1	1	0
1	1	1

รูปที่ 2.4 แสดงค่าผลลัพธ์ที่ได้จาก LFSR แบบ Galois ขนาด 3 บิต

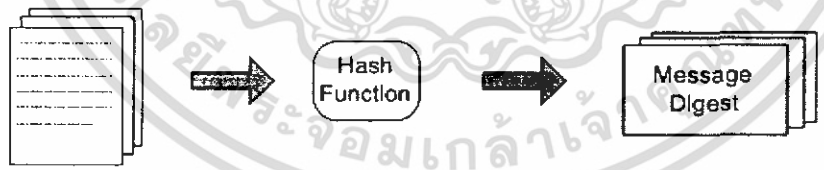
ค่าแท็ปคือ ตำแหน่งที่จะส่งค่ากลับคืนเพื่อนำมาใช้ในการทำเอ็กซ์คลูซีพออร์(XOR) เป็นค่าที่เอามาใช้เป็นค่าอินพุท ซึ่งตำแหน่งของค่าแท็ปนั้นจะมีความแตกต่างกันขึ้นอยู่กับจำนวนของรีจิสเตอร์ที่ต้องการ ตารางที่จะแสดงในภาคผนวกจะแสดงค่าแท็ปเพื่อสร้างเอาท์พุทออกมาให้มีช่วงมากที่สุดซึ่งดูค่าแท็ปได้จากตารางในภาคผนวก ก. นี้จะแสดงค่าแท็ปเพื่อสร้างออกมาให้มีลำดับค่ามากที่สุด

ขนาดของ LFSR ที่เป็นไปได้ คือตั้งแต่ขนาด 2-168 บิต

ช่วงของผลลัพธ์ที่มากที่สุดที่ได้ออกมาโดยใช้อัลกอริธึม LFSR นั้นจะมีค่าเป็น 2n-1 โดยค่า n นั้นเป็นจำนวนของรีจิสเตอร์ที่ใช้ ดังนั้นหากใช้รีจิสเตอร์ขนาด 32 บิต จะให้ค่าผลลัพธ์ออกมาถึง 4 ล้านค่า(2^{32}-1)

2.3 การแปลงข้อมูล

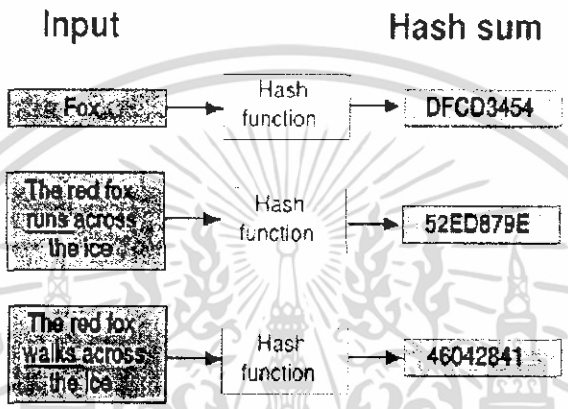
การแปลงข้อมูลหรือการทำเมสเซสไดเจสต์เป็นวิธีที่สำคัญที่สามารถใช้ในการตรวจสอบว่าไฟล์ในระบบที่ใช้งานมีการเปลี่ยนแปลงแก้ไขหรือไม่ (ไม่ว่าจะโดยเจตนาหรือไม่ก็ตาม) บางครั้งการเปลี่ยนแปลงแก้ไขอาจถูกกระทำโดยผู้ที่ไม่มีสิทธิ์ เช่น ผู้บุกรุก เป็นต้น วิธีการใช้เมสเซสไดเจสต์เพื่อตรวจสอบไฟล์ในระบบหรือการส่งข้อมูลหากันผ่านทางเครือข่าย คือให้เลือกใช้อัลกอริทึมใดอัลกอริธึมหนึ่ง เพื่อสร้างเมสเซสไดเจสต์ของไฟล์ที่จะส่งหากันแล้วทำการส่งไป เมื่อผู้รับได้รับข้อความก็จะทำการหาแมสเซสไดเจสต์แล้วเอาค่าที่ได้มาเปรียบเทียบกับค่าแมสเซสไดเจสต์ที่ส่งมาถ้าได้เหมือนกันก็จะสามารถมั่นใจได้ว่าข้อความไม่ได้ถูกเปลี่ยนแปลง แต่หากไม่เท่ากันก็แสดงว่าข้อความถูกเปลี่ยนแปลง นอกจากนี้เมสเซสไดเจสต์ยังเป็นส่วนหนึ่งของการลงลายมือชื่ออิเล็กทรอนิกส์ กล่าวคือการลงลายมือชื่ออิเล็กทรอนิกส์ในปัจจุบันจะใช้การลงลายมือชื่อกับ เมสเซสไดเจสต์ของข้อความตั้งต้นแทนการลงลายมือชื่อกับข้อความตั้งต้นทั้งข้อความ



รูปที่ 2.5 แสดงตัวอย่างการทำแฮชฟังก์ชัน

ลักษณะของอัลกอริทึมในการทำเมสเซสไดเจสต์คือ การสร้างข้อความสรุปที่สามารถใช้เป็นตัวแทนของข้อความตั้งต้นได้ โดยทั่วไปเมสเซสไดเจสต์ที่ได้ออกมาจะมีความยาวตั้งแต่ 128 ขึ้นไปขึ้นอยู่กับอัลกอริธึมที่เลือกใช้และจะไม่ขึ้นกับขนาดความยาวของข้อความตั้งต้น ในการทำแฮช (hash) หรือ เมสเซสไดเจสต์นั้นเป็นการทำฟังก์ชันทางเดียว ซึ่งลักษณะของอัลกอริธึมที่ดีคือ

1. ถ้าเปลี่ยนแปลงอักษรไปแม้แต่บิตเดียว ต้องให้ได้ผลลัพธ์ที่ไม่เหมือนเดิมเลย ไม่งั้นคนจะจับ รูปแบบการทำงานได้ เช่น f(x) = 12345, f(x+1) = 99999 เป็นต้น ทำให้หาความสัมพันธ์ได้ยาก ทุกๆ บิตของไดเจสต์จะมีโอกาสร้อยละ 50 ที่จะแปรเปลี่ยนค่าไปด้วย ซึ่งหมายถึงว่า 0 เปลี่ยนค่าเป็น 1 และ 1 เปลี่ยนเป็น 0 คุณสมบัติข้อนี้สามารถอธิบายได้ว่าการเปลี่ยนแปลงแก้ไขข้อความตั้งต้นโดยผู้ไม่ประสงค์ดีแม้ว่าอาจแก้ไขเพียงเล็กน้อยก็ตาม ก็จะส่งผลให้ผู้รับข้อความทราบว่าข้อความที่ตนได้รับไม่ใช่ข้อความตั้งต้น (โดยการนำข้อความที่ตนได้รับเข้าอัลกอริทึมเพื่อทำการคำนวณหาไดเจสต์ออกมา แล้วจึงเปรียบเทียบไดเจสต์ที่คำนวณได้กับไดเจสต์ที่ส่งมาให้ด้วย ถ้าต่างกัน แสดงว่าข้อความที่ได้รับนั้นถูกเปลี่ยนแปลงแก้ไข)



รูปที่ 2.6 ตัวอย่างการทำงานของแฮช

2. ต้องไม่ใช่ฟังก์ชันแบบ 1:1 เพราะสามารถทำการจับคู่ได้ง่าย ฟังก์ชันที่ใช้ส่วนใหญ่จะเป็น modulo เพราะ มีคุณสมบัติที่ดี X mod 5 == 3 ตัวเลขอะไรบ้างที่หาร 5 แล้วเหลือเศษ 3 คำตอบมีได้เป็นล้าน ๆ คำตอบเลย

3. มีการชนกัน (collision) ที่ต่ำ การชนกันหมายถึงว่าการที่มีข้อความตั้งต้นสองข้อความที่แตกต่างกันแต่เมื่อคำนวณเมสเซสไดเจสต์แล้วได้ออกมาค่าเดียวกัน อย่างตัวอย่างข้อ 2 เกิดการชนกันที่สูงมากทุก ๆ 5 ตัวจะได้คำตอบที่ใช้ได้ ถึงจะไม่ใช่คำตอบ แต่ผลลัพธ์เท่ากัน ก็ถือว่าถูกอย่างนี้ไม่ควรใช้ โอกาสที่ข้อความตั้งต้น 2 ข้อความใดๆ ที่มีความแตกต่างกัน จะสามารถคำนวณได้ค่าไดเจสต์เดียวกันมีโอกาสน้อยมาก คุณสมบัติข้อนี้ทำให้แน่ใจได้ว่า เมื่อผู้ไม่ประสงค์ดีทำการแก้ไขข้อความตั้งต้น ผู้รับข้อความที่ถูกแก้ไขไปแล้วนั้นจะสามารถตรวจพบได้ถึงความผิดปกติที่เกิดขึ้นอย่างแน่นอน

อย่างไรก็ตามในทางทฤษฎีแล้ว มีโอกาสที่ข้อความ 2 ข้อความที่แตกต่างกันจะสามารถคำนวณแล้วได้ค่าไดเจสต์เดียวกัน แต่ก็มีข้อแม้ในเรื่องของเวลานั่นเองที่ทำให้สามารถใช้วิธีนี้ได้ อัลกอริทึมสำหรับสร้างไดเจสต์ที่ดีควรจะมีโอกาสน้อยมากๆ ที่จะก่อให้เกิดปัญหาการชนกันของไดเจสต์

โดยปัญหานี้ทำให้วิธีการตรวจสอบความมั่นคงปลอดภัยของข้อมูลที่อาศัยความเป็นหนึ่งเดียวของเมสเซสไดเจสต์ว่าไม่น่าเชื่อถือ เช่น ทำให้ไม่สามารถยืนยันความครบถ้วนแท้จริงของซอฟต์แวร์สำหรับดาวน์โหลดด้วยเมสเซสไดเจสต์ของซอฟต์แวร์นั้นได้ หรือ เป็นการเพิ่มโอกาสให้บุคคลที่ฉ้อโกงสามารถเขียน E-mail เพื่อจะแจ้งความจำนงในการขอปิดบัญชีของเจ้าของบัญชีได้โดยใช้ลายมือชื่อดิจิทัลปลอม โดยเมสเซสไดเจสที่ทำออกมานั้นเป็นตัวตรวจสอบความถูกต้องของข้อความว่าข้อความที่ส่งไปนั้นมีการเปลี่ยนแปลงไปรึเปล่า

อัลกอริทึมสำหรับสร้างไดเจสต์ที่เป็นที่รู้จักกันดีมีดังนี้

อัลกอริทึม MD2

ผู้พัฒนาคือ Ronald Rivest ข้อเสียของอัลกอริทึมนี้คือใช้เวลามากในการคำนวณไดเจสต์หนึ่งๆ MD2 จึงไม่ค่อยได้มีการใช้งานกันมากนัก MD2 สร้างไดเจสต์ที่มีความยาว 128 บิต

อัลกอริทึม MD4

อัลกอริทึมนี้พัฒนาขึ้นมาเพื่อแก้ปัญหาความล่าช้าในการคำนวณของ MD2 อย่างไรก็ตามในภายหลังได้พบว่าอัลกอริทึมมีข้อบกพร่อง คือ มีปัญหาการชนกันของไดเจสต์มีโอกาสเกิดขึ้นได้ไม่น้อย ซึ่งผู้บุกรุกอาจใช้ประโยชน์จากจุดอ่อนนี้เพื่อทำการแก้ไขข้อความตั้งต้นที่ส่งมาให้ได้ MD4 ผลิตไดเจสต์ที่มีขนาด 128 บิต

อัลกอริทึม MD5

พัฒนาต่อจาก MD4 เพื่อให้มีความปลอดภัยที่สูงขึ้น ถึงแม้จะเป็นที่นิยมใช้งานกันอย่างแพร่หลาย แต่ก็มีจุดบกพร่องของ MD5 (เช่นเดียวกับ MD4) จึงทำให้ความนิยมเริ่มลดลง MD5 ผลิตไดเจสต์ที่มีขนาด 128 บิต

อัลกอริทึม SHA

SHA ย่อจาก Secure Hash Algorithm อัลกอริทึม SHA ได้รับแนวคิดในการพัฒนามาจาก MD4 และได้รับการพัฒนาขึ้นมาเพื่อใช้งานร่วมกับอัลกอริทึม DSS (ซึ่งใช้ในการลงลายมือชื่ออิเล็กทรอนิกส์) หลังจากที่ได้มีการตีพิมพ์เผยแพร่อัลกอริทึมนี้ได้ไม่นาน NIST ก็ประกาศตามมาว่าอัลกอริทึมจำเป็นต้องได้รับการแก้ไขเพิ่มเติมเล็กน้อยเพื่อให้สามารถใช้งานได้อย่างเหมาะสม SHA สร้างไดเจสต์ที่มีขนาด 160 บิต

อัลกอริทึม SHA-1

SHA-1 เป็นอัลกอริทึมที่แก้ไขเพิ่มเติมเล็กน้อยจาก SHA การแก้ไขเพิ่มเติมนี้เป็นที่เชื่อกันว่าทำให้อัลกอริทึม SHA-1 มีความปลอดภัยที่สูงขึ้น SHA-1 สร้างไดเจสต์ที่มีขนาด 160 บิต

อัลกอริทึม SHA-256, SHA-384 และ SHA-512

เป็นอัลกอริธึมที่แก้ไขขนาดให้มีความยาวมากขึ้น ทำให้มีความปลอดภัยมากขึ้นกว่าเดิม เนื่องจากขนาดของเมสเซสไดเจสต์ที่ออกมามีขนาดที่ยาวขึ้นก็ต้องใช้เวลามากขึ้น

2.4 การแปลงข้อมูลโดยใช้การคำนวณแบบ SHA1

ข้อจำกัดของ SHA1 คือ ข้อความที่เป็นอินพุทเข้ามาจะต้องมีขนาดน้อยกว่า 2^{64} ซึ่งค่าเมสเซสไดเจสต์ที่ได้ออกมานั้นจะมีขนาด 160 บิต โดยค่าที่ได้ออกมาเป็นการคำนวณค่าถึง 80 รอบ แล้วได้ผลลัพธ์ออกมาเป็น A B C D E ซึ่งแต่ละตัวมีค่าตัวละ 32 บิต 5 ตัวมาต่อกันจะได้เป็นเมสเซสไดเจสต์ที่มีขนาด 160 บิต

ขั้นตอนการทำงานคือ

1. **การเติมเต็มบิตต่อท้าย(Padding Bit)**

ทำการแบ่งข้อความที่รับเข้ามาเป็นบล๊อกๆ โดยมีขนาดบล๊อกละ 512 บิต และในบล๊อกสุดท้ายจะทำการเติมจำนวนบิตให้ครบ 448 บิต และอีก 64 บิตที่เหลือใช้ในการเติมความยาวของข้อความ และการเติมบิตต่อท้ายนั้นเริ่มต้นด้วยการเติมบิต "1" ก่อน แล้วจึงตามด้วยบิต "0" ต่อท้ายไปจนกว่าความยาวของบิตข้อความในบล๊อกนั้นจะครบ 448 บิต ขั้นตอนดังตัวอย่างข้างล่างนี้

สมมติให้ข้อความคือ ABC ก็จะมองเป็นแอสกีได้เป็น 616263

แปลงเป็นเลขฐานสองได้ 01100001 01100010 01100011

- เติม 1 หลังข้อความนั้น

  01100001 01100010 01100011 1

- เติม 0 ให้มีขนาด 448 บิต(เติม 0 ไปอีก 423 บิตโดยแสดงเป็นเลขฐาน16)

  61626380 00000000 00000000 00000000 00000000

  00000000 00000000 00000000 00000000 00000000

  00000000 00000000 00000000 00000000

2. **การเติมความยาวบิตของข้อความ**

ค่าความยาวของข้อความที่เป็นอินพุทนั้นจะเก็บไว้ใน 64 บิตที่เหลือไว้ เนื่องจากเป็นข้อกำหนดว่าความยาวของอินพุทที่สามารถใช้อัลกอริธึม SHA1 นั้นมีขนาดไม่เกิน 2^{64} บิตดังนั้นพื้นที่ 64 บิตที่เหลือไว้จึงเพียงพอที่จะใช้ในการเก็บค่าความยาวบิตของข้อความ โดยเติมเวิร์ด(word) 32 บิตที่มีลำดับต่ำก่อนแล้วจึงตามด้วยเวิร์ดที่มีลำดับสูง ดังนั้นข้อความที่ได้ออกมาจะมีขนาดเป็นหลายเท่าของ 512 บิต หรือมีขนาดเป็นหลายเท่าของ 16 เวิร์ด (1 เวิร์ด มีขนาด 32 บิต) ให้ M[0.....N-1] แทรเวิร์ดของข้อความที่ผ่านกระบวนการต่อเติมแล้ว โดยที่ N คือ จำนวนเท่าของ 16 ตัวอย่างเช่น

- เติม ความยาวของข้อความนั้นไปในส่วนท้ายสุดของบล็อคนั้น

ความยาวของข้อความคือ 24 ดังนั้นเติมไปในส่วนท้ายของบล็อคนี้ (แสดงเป็นเลขฐานสิบหก)

61626380 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000 00000000

00000018

3. การกำหนดค่าเริ่มต้นของบัฟเฟอร์ของแมสเซสไดเจสต์

บัฟเฟอร์นั้นมีขนาด 160 บิต เพื่อเก็บผลลัพธ์ของการแฮช(Hash) โดยจะใช้บัฟเฟอร์ที่มีขนาด 32 บิตทั้งหมด 5 ตัวคือ A B C D และ E โดยบัฟเฟอร์เหล่านี้มีค่าเริ่มต้นและใช้ในการเก็บค่าที่ได้จากการคำนวณในแต่ละรอบด้วย ซึ่งเรียงลำดับจากไบต์ที่มีลำดับต่ำสุดก่อน

4. กระบวนการหาแมสเซสไดเจสต์

ฟังก์ชันที่ใช้คือ f0 , f1 , f2 , ……., f79 ที่ใช้ในการคำนวณ 80 รอบ

คือสำหรับ f_t , 0<=t<=79 โดยมีการกำหนดค่าดังนี้

$F_t(B,C,D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D)$; (0 <= T <= 19)

$F_t(B,C,D) = B \text{ XOR } C \text{ XOR } D$; (20 <= T <= 39)

$F_t(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D)$; (40 <= T <= 59)

$F_t(B,C,D) = B \text{ XOR } C \text{ XOR } D$; (60 <= T <= 79)

โอเปอร์เรชันที่ใช้

เนื่องจากเป็นพจน์หนึ่งที่ใช้ในการคำนวณคือ

$S^n(X) = (X << n) \text{ OR } (X >> 32-n)$

ค่าคงที่ที่ใช้ในการคำนวณ คือ

K_t = 5A827999 (0 <= t <= 19)

K_t = 6ED9EBA1 (20 <= t <= 39)

K_t = 8F1BBCDC (40 <= t <= 59)

K_t = CA62C1D6 (60 <= t <= 79)

การคำนวณจะมีการคำนวณเพื่อให้ได้ Temp ออกมา

โดยมีการกำหนดค่าเริ่มต้นในการคำนวณ คือ $\{H_i\}$ ซึ่งก็คือ A B C D และ E เพื่อใช้ในการคำนวณดังนี้ (แสดงในรูปแบบเลขฐาน 16)

H_0 = 67452301

H_1 = EFCDAB89

H_2 = 98BADCFE

H_3 = 10325476

H_4 = C3D2E1F0

- แบ่งบล็อคออกเป็นเวิร์ด(word) เวิร์ดหนึ่งมี 32 บิตจะได้ทั้งหมด 16 เวิร์ดคือแทนด้วย W[0] ถึง W[15] คือ

W[0] = 61626380 W[1] = 00000000 W[2] = 00000000

W[3] = 00000000 W[4] = 00000000 W[5] = 00000000

W[6] = 00000000 W[7] = 00000000 W[8] = 00000000

W[9] = 00000000 W[10] = 00000000 W[11] = 00000000

W[12] = 00000000 W[13] = 00000000 W[14] = 00000000

W[15] = 00000000

และสำหรับค่า t = 16 ถึง 79 จะหาค่า W_t ได้จาก $W_t = S1(W_t\text{-}3 \text{ XOR } W_t\text{-}8 \text{ XOR } W_t\text{-}14 \text{ XOR } W_t\text{-}16)$

- จากนั้นกำหนดให้ A = H_0 , B = H_1 , C = H_2, D = H_3 , E = H_4
- จากนั้นทำการคำนวณโดยกำหนดว่าจาก t=0 ถึง 79 ให้ทำ

- $TEMP = S^5(A) + F_t(B,C,D) + E + W_t + K_t$

- กำหนดให้ E = D ; D = C ; $C = S^{30}(B)$; B = A ; A = TEMP

- เมื่อคำนวณครบ 80 รอบจะได้

H0 = H0+ A , H1 = H1 + B , H2 = H2 + C , H3 = H3 +D , H4 = H4 +E

ก็จะได้เมสเซสไดเจสต์ขนาด 160 บิตที่ได้จากการเรียงกัน 5 words คือ H0 H1 H2 H3 H4

หมายเหตุ หากข้อความที่จะทำเมสเซสไดเจสต์นั้นมีความยาวเกิน 512 บิต ค่า H0 H1 H2 H3 H4 ที่ได้จากบล็อคแรกก็จะไปใช้เป็นค่า H0 H1 H2 H3 H4 ในบล็อกต่อๆไป

2.5 การเข้ารหัสลับ

การเข้ารหัสลับข้อมูล คือ การทำให้ข้อมูลเป็นความลับ โดยอาศัยการทำการเข้ารหัสลับและการถอดรหัสลับซึ่งเป็นส่วนสำคัญของระบบข้อมูลในปัจจุบันนี้ จึงมองเป็น 2 ส่วนหลักๆคือ เรื่องการเข้ารหัสลับ และการถอดรหัสลับ

การเข้ารหัสลับ(Encryption) เป็นการเปลี่ยนรูปข้อมูลโดยใช้กระบวนการต่างๆในการเปลี่ยนรูปข้อมูลเพื่อให้ไม่สามารถอ่านหรือตีค่าแปลความหมายได้ เพื่อทำให้ข้อมูลนั้นเป็นความลับ โดยวิธีในการเข้ารหัสนั้นโดยพื้นฐานมีหลักการที่สำคัญ 2 อย่างคือ

1. การแทนที่ (Substitution) เป็นการแทนที่บิตนั้นๆด้วยข้อมูลอื่น ทำให้ข้อมูลมีความสับสนยากต่อการถอดรหัส (โดยตำแหน่งบิตนั้นๆ ยังคงเรียงกันเหมือนเดิม)

2. การสับเปลี่ยนตำแหน่ง (Permutation) เป็นการสับเปลี่ยนตำแหน่งใดๆของข้อมูล เมื่อมีการสับเปลี่ยนตำแหน่งมากๆ ทำให้ข้อมูลมีความซับซ้อนยากต่อการถอดรหัส

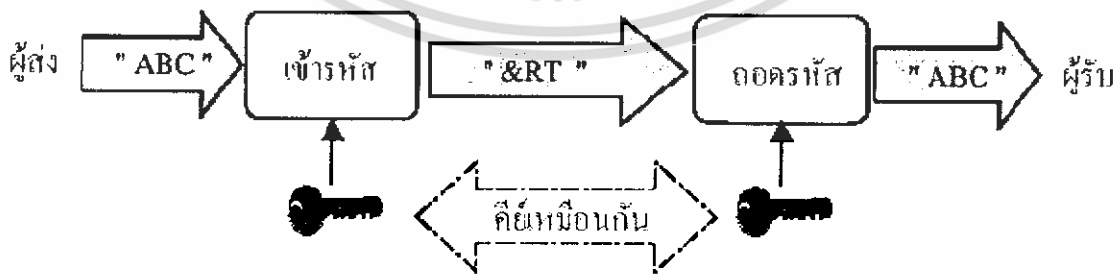
การถอดรหัส (Decryption) เป็นการแปลงข้อมูลที่ได้ผ่านการเข้ารหัสให้กลับมาอยู่ในสภาพเดิมที่สามารถอ่านและตีความหมายได้ซึ่งการเข้าและถอดรหัสนั้น ได้ถูกควบคุมโดยกุญแจ (Key) โดยได้แบ่งระบบของการเข้ารหัสตามลักษณะของกุญแจที่ใช้ เป็น 2 แบบ คือ

- ระบบการเข้ารหัสโดยใช้กุญแจเดียว (Single Key Encryption)
- ระบบการเข้ารหัสโดยใช้กุญแจสาธารณะ (Public Key Encryption)

2.5.1 ระบบการเข้ารหัสโดยใช้กุญแจเดียว

อัลกอริทึมแบบนี้จะใช้กุญแจที่เรียกว่า กุญแจลับ (Secret key) ซึ่งมีเพียงหนึ่งเดียวเพื่อใช้ในการเข้าและถอดรหัสข้อความที่ส่งไป อัลกอริทึมยังสามารถแบ่งย่อยออกเป็น 2 ประเภท ได้แก่ แบบบล็อค (Block Algorithms) ซึ่งจะทำการเข้ารหัสทีละบล็อค (1 บล็อคประกอบด้วยหลายไบต์ เช่น 64 ไบต์ เป็นต้น) และแบบสตรีม (Stream Algorithms) ซึ่งจะทำการเข้ารหัสทีละไบต์อัลกอริทึมแบบนี้จะใช้กุญแจที่เรียกว่า กุญแจลับ (Secret key) ซึ่งมีเพียงหนึ่งเดียวเพื่อใช้ในการเข้าและถอดรหัสข้อความที่ส่งไป อัลกอริทึมยังสามารถแบ่งย่อยออกเป็น 2 ประเภท ได้แก่ แบบบล็อค (Block Algorithms) ซึ่งจะทำการเข้ารหัสทีละบล็อค (1 บล็อคประกอบด้วยหลายไบต์ เช่น 64 ไบต์ เป็นต้น) และแบบสตรีม (Stream Algorithms) ซึ่งจะทำการเข้ารหัสทีละไบต์

ตัวอย่างของการเข้ารหัสด้วยระบบนี้ ได้แก่ DES, 3DES, IDEA เป็นต้น



รูปที่ 2.7 การเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว

2.5.2 ระบบการเข้ารหัสโดยใช้กุญแจสาธารณะ

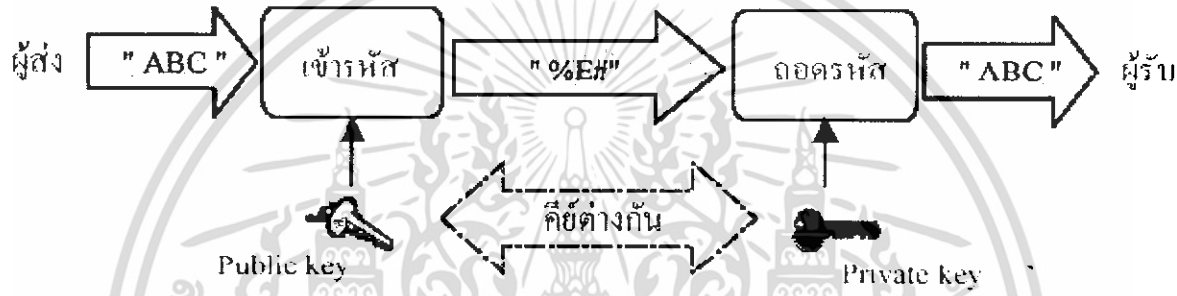
การเข้ารหัสโดยใช้กุญแจสาธารณะ ได้นำกุญแจมาประกอบการเข้ารหัส 2 อัน คือ

1. กุญแจส่วนตัว (Private Key) เป็นกุญแจที่จะต้องเก็บเป็นความลับ

2. กุญแจสาธารณะ (Public Key) เป็นกุญแจที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งใช้กุญแจที่เรียกกันว่า กุญแจสาธารณะ (Public keys) ในการเข้ารหัสและใช้กุญแจที่เรียกกันว่า กุญแจส่วนตัว (Private keys) ในการถอดรหัสข้อมูลนั้น กุญแจสาธารณะนี้สามารถส่งมอบให้กับผู้อื่นได้ เช่น เพื่อนร่วมงานที่เราต้องการติดต่อด้วย หรือแม้กระทั่งวางไว้บนเว็บไซต์เพื่อให้ผู้อื่นสามารถดาวน์โหลดไปใช้งานได้ สำหรับกุญแจส่วนตัวนั้นต้องเก็บไว้กับผู้เป็นเจ้าของกุญแจส่วนตัวเท่านั้นและห้ามเปิดเผยให้ผู้อื่นทราบโดยเด็ดขาด

ซึ่งกุญแจทั้งสองอันนี้ เป็นกุญแจที่ต่างกัน โดยมีวิธีการเข้าและถอดรหัส ดังรูปนี้



รูปที่ 2.8 การเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ

2.5.3 ความแข็งแกร่งของอัลกอริธึมสำหรับการเข้ารหัส

ความแข็งแกร่งของอัลกอริทึมหมายถึงความยากในการที่ผู้บุกรุกจะสามารถถอดรหัสข้อมูลได้โดยปราศจากกุญแจที่ใช้ในการเข้ารหัส ซึ่งจะขึ้นอยู่กับปัจจัยดังนี้

- การเก็บกุญแจเข้ารหัสไว้อย่างเป็นความลับ ผู้เป็นเจ้าของกุญแจลับหรือส่วนตัวต้องระมัดระวังไม่ให้กุญแจสูญหายหรือล่วงรู้โดยผู้อื่น

- ความยาวของกุญแจเข้ารหัส ปกติกุญแจเข้ารหัสจะมีความยาวเป็นบิต ยิ่งจำนวนบิตของกุญแจยิ่งมาก ยิ่งทำให้การเดาเพื่อสุ่มหากุญแจที่ถูกต้องเป็นไปได้ยากยิ่งขึ้น (เช่น กุญแจขนาด 1 บิต จะสามารถแทนตัวเลขได้ 2 ค่าคือ 0 กับ 1 กุญแจขนาด 2 บิต จะเป็นไปได้ 4 ค่าคือ 0, 1, 2, 3 เป็นต้น)

- ความไม่เกรงกลัวต่อการศึกษาหรือดูอัลกอริทึมเพื่อหารูปแบบของการเข้ารหัส อัลกอริทึมที่ดีต้องเปิดให้ผู้รู้ทำการศึกษาในรายละเอียดได้โดยไม่เกรงว่าผู้ศึกษาจะสามารถจับรูปแบบของการเข้ารหัสได้

- การมีช่องทางลับในอัลกอริทึม อัลกอริทึมที่ดีต้องไม่แฝงไว้ด้วยช่องทางลับที่สามารถใช้เป็นทางเข้าไปสู่อัลกอริทึม แล้วอาจใช้เพื่อทำการถอดรหัสข้อมูลได้ ช่องทางลับนี้ทำให้ไม่จำเป็นต้องใช้กุญแจในการถอดรหัส

- ความพยายามในการหาความสัมพันธ์ในข้อมูลที่ได้รับ กล่าวคือเมื่อผู้บุกรุกทราบข้อมูลบางอย่างที่เป็นข้อมูลตั้งต้นซึ่งยังไม่ได้เข้ารหัส รวมทั้งมีข้อมูลที่เข้ารหัสแล้ว (ของข้อมูลตั้งต้นนั้น) ผู้บุกรุกอาจจะสามารถหาความสัมพันธ์ระหว่างข้อความทั้งสองนั้นได้ ซึ่งจะเป็นวิธีการในการถอดรหัสข้อมูลได้ ปัญหานี้เรียกกันว่า Known plaintext attack (คำว่า plaintext หมายถึงข้อความตั้งต้นที่ยังไม่ได้ผ่านการเข้ารหัส)

- คุณสมบัติของข้อความตั้งต้น คุณสมบัตินี้อาจใช้เป็นช่องทางในการถอดรหัสข้อมูลได้ อัลกอริทึมที่ดีต้องไม่ใช้คุณสมบัติของข้อความเป็นกลไกในการเข้ารหัสข้อมูล

คำแนะนำในการเลือกใช้อัลกอริทึมคือให้ใช้อัลกอริทึมที่ได้มีการใช้งานมาเป็นระยะเวลานานแล้ว ทั้งนี้เนื่องจากหากปัญหาของอัลกอริทึมนี้มีจริง ก็คงเกิดขึ้นมานานแล้วและก็คงเป็นที่ทราบกันแล้ว นั่นคืออย่างน้อยที่สุดจวบจนกระทั่งถึงปัจจุบัน ก็ยังไม่มีการบุกรุกที่ทำให้อัลกอริทึมนั้นไม่สามารถใช้งานได้อย่างปลอดภัยเป็นที่ประจักษ์ ดังนั้นจึงไม่ควรใช้อัลกอริทึมใหม่ๆ ที่เพิ่งได้มีการนำเสนอกันสู่สาธารณะ เพราะอาจมีช่องโหว่แฝงอยู่และยังไม่เป็นที่ทราบในขณะนี้

2.5.4 ความยาวของกุญแจที่ใช้ในการเข้ารหัส

ความยาวของกุญแจเข้ารหัสมีหน่วยนับเป็นบิต หนึ่งบิตในคอมพิวเตอร์เป็นตัวเลขฐานสองที่ประกอบด้วยค่า 0 และ 1 กุญแจที่มีความยาว 1 บิต ตัวเลขที่เป็นไปได้เพื่อแทนกุญแจนั้น จึงอาจมีค่าเป็น 0 หรือ 1 กุญแจที่มีความยาว 2 บิต ตัวเลขที่เป็นไปได้จึงเป็น 0, 1, 2 และ 3 ตามลำดับ กุญแจที่มีความยาว 3 บิต ตัวเลขที่เป็นไปได้จะอยู่ระหว่าง 0 ถึง 7 ดังนั้นเมื่อเพิ่มความยาวของกุญแจทุกๆ 1 บิต ค่าที่เป็นไปได้ของกุญแจจะเพิ่มขึ้นเป็นสองเท่าตัว หรือจำนวนกุญแจที่เป็นไปได้จะเพิ่มขึ้นเป็น 2 เท่าตัวนั่นเอง

ฉะนั้นจะเห็นได้ว่ากุญแจยิ่งมีความยาวมาก โอกาสที่ผู้บุกรุกจะสามารถคาดเดากุญแจที่ตรงกับหมายเลขที่ถูกต้องของกุญแจจะยิ่งยากมากขึ้นตามลำดับ ในการที่ผู้บุกรุกลองผิดลองถูกกับกุญแจโดยใช้กุญแจที่มีหมายเลขต่างๆ กัน เพื่อหวังที่จะพบกุญแจที่ถูกต้องและสามารถใช้ถอดรหัสข้อมูลได้ การลองผิดลองถูกนี้เราเรียกกันว่าหรือการค้นหากุญแจ (Key search) นั่นเอง ทฤษฎีได้กล่าวไว้ว่าการลองผิดลองถูกนี้โดยเฉลี่ยจะต้องทดลองกับกุญแจเป็นจำนวนครึ่งหนึ่งของกุญแจทั้งหมดก่อนที่จะพบกุญแจที่ถูกต้อง

ความยาวของกุญแจที่มีขนาดเหมาะสมจึงขึ้นอยู่กับความเร็วในการค้นหากุญแจของผู้บุกรุกและระยะเวลาที่ต้องการให้ข้อมูลมีความปลอดภัย ตัวอย่างเช่น ถ้าผู้บุกรุกสามารถลองผิดลองถูก

กับกุญแจเป็นจำนวน 10 กุญแจภายในหนึ่งวินาทีแล้ว กุญแจที่มีความยาว 40 บิต จะสามารถป้องกันข้อมูลไว้ได้ 3,484 ปี ถ้าผู้บุกรุกสามารถลองได้เป็นจำนวน 1 ล้านกุญแจในหนึ่งวินาที (เทคโนโลยีปัจจุบันสามารถทำได้) กุญแจที่มีความยาว 40 บิตจะสามารถป้องกันข้อมูลไว้ได้เพียง 13 วันเท่านั้น (ซึ่งอาจไม่เพียงพอสำหรับในบางลักษณะงาน) ด้วยเทคโนโลยีในปัจจุบันหากผู้บุกรุกสามารถทดลองได้เป็นจำนวน 1,000 ล้านกุญแจในหนึ่งวินาที กุญแจขนาด 128 บิตจะสามารถป้องกันข้อมูลไว้ได้ 1022 ปี ดังนั้นด้วยลักษณะงานทั่วไปกุญแจขนาด 128 บิตจะพอเพียงต่อการรักษาความลับของข้อมูลเอาไว้ได้

ตัวอย่างอัลกอริธึมที่รู้จักกันดี

อัลกอริทึม DES

DES ย่อมาจาก Data Encryption Standard อัลกอริทึมนี้ได้รับการรับรองโดยรัฐบาลสหรัฐอเมริกาในปี ค.ศ. 1977 ให้เป็นมาตรฐานการเข้ารหัสข้อมูลสำหรับหน่วยงานของรัฐทั้งหมด ในปี 1981 อัลกอริทึมยังได้รับการกำหนดให้เป็นมาตรฐานการเข้ารหัสข้อมูลในระดับนานาชาติตามมาตรฐาน ANSI (American National Standards) อีกด้วย DES เป็นอัลกอริทึมแบบบล็อกซึ่งใช้กุญแจที่มีขนาดความยาว 56 บิตและเป็นอัลกอริทึมที่มีความแข็งแกร่ง แต่เนื่องด้วยขนาดความยาวของกุญแจที่มีขนาดเพียง 56 บิต ซึ่งในปัจจุบันถือได้ว่าสั้นเกินไป ผู้บุกรุกอาจใช้วิธีการลองผิดลองถูกเพื่อค้นหากุญแจที่ถูกต้องสำหรับการถอดรหัสได้

ในปี 1998 ได้มีการสร้างเครื่องคอมพิวเตอร์พิเศษขึ้นมาซึ่งมีมูลค่า 250,000 เหรียญสหรัฐเพื่อใช้ในการค้นหากุญแจที่ถูกต้องของการเข้ารหัสข้อมูลหนึ่งๆ ด้วย DES และพบว่าเครื่องคอมพิวเตอร์นี้สามารถค้นหากุญแจที่ถูกต้องได้ภายในระยะเวลาไม่ถึงหนึ่งวัน

อัลกอริทึม Triple-DES

Triple-DES เป็นอัลกอริทึมที่เสริมความปลอดภัยของ DES ให้มีความแข็งแกร่งมากขึ้นโดยใช้อัลกอริทึม DES เป็นจำนวนสามครั้งเพื่อทำการเข้ารหัส แต่ละครั้งจะใช้กุญแจในการเข้ารหัสที่แตกต่างกัน ดังนั้นจึงเปรียบเสมือนการใช้กุญแจเข้ารหัสที่มีความยาวเท่ากับ 56*3 = 168 บิต Triple-DES ได้ถูก ใช้งานกับสถาบันทางการเงินอย่างแพร่หลาย รวมทั้งใช้งานกับโปรแกรม Secure Shell (ssh) ด้วย

อัลกอริทึม Blowfish

Blowfish เป็นอัลกอริทึมที่มีความรวดเร็วในการทำงาน มีขนาดเล็กกระทัดรัด และใช้การเข้ารหัสแบบบล็อก ผู้พัฒนาคือ Bruce Schneier อัลกอริทึมสามารถใช้กุญแจที่มีขนาดความยาวตั้งแต่ไม่มากนักไปจนถึงขนาด 448 บิต ซึ่งทำให้เกิดความยืดหยุ่นสูงในการเลือกใช้กุญแจ รวมทั้งอัลกอริทึมยังได้รับการออกแบบมาให้ทำงานอย่างเหมาะสมกับหน่วยประมวลผลขนาด 32 หรือ 64

บิต Blowfish ได้เปิดเผยสู่สาธารณะและไม่ได้มีการจดสิทธิบัตรใดๆ นอกจากนั้นยังใช้ในโปรแกรม SSH และอื่นๆ

อัลกอริทึม IDEA

IDEA ย่อมาจาก International Data Encryption Algorithm อัลกอริทึมนี้ได้รับการพัฒนาในประเทศสวิสเซอร์แลนด์ที่เมือง Zarich โดย James L. Massey และ Xuejia Lai และได้รับการตีพิมพ์เผยแพร่ในปี ค.ศ. 1990 อัลกอริทึมใช้กุญแจที่มีขนาด 128 บิต และได้รับการใช้งานกับโปรแกรมยอดฮิตสำหรับการเข้ารหัสและลงลายมือชื่ออิเล็กทรอนิกส์ในระบบอีเมล์ที่มีชื่อว่า PGP ต่อมา IDEA ได้รับการจดสิทธิบัตรทางด้านซอฟต์แวร์โดยบริษัท Ascom-Tech AG ในประเทศสวิสเซอร์แลนด์ ซึ่งทำให้การนำไปใช้ในงานต่างๆ เริ่มลดลง ทั้งนี้เนื่องจากติดปัญหาเรื่องลิขสิทธิ์นั่นเอง

อัลกอริทึม RC4

อัลกอริทึมนี้เป็นอัลกอริทึมแบบสตรีม (ทำงานกับข้อมูลทีละไบต์) ซึ่งได้รับการพัฒนาขึ้นมาโดย Ronald Riverst และถูกเก็บเป็นความลับทางการค้าโดยบริษัท RSA Data Security ในภายหลังอัลกอริทึมนี้ได้รับการเปิดเผยใน Usenet เมื่อปี ค.ศ. 1994 และเป็นที่ทราบกันว่าเป็นอัลกอริทึมที่มีความแข็งแกร่งโดยสามารถใช้ขนาดความยาวของกุญแจที่มีขนาดตั้งแต่ 1 บิตไปจนกระทั่งถึงขนาด 2048 บิต

อัลกอริทึม Rijndael (หรืออัลกอริทึม AES)

อัลกอริทึมนี้ได้รับการพัฒนาโดย Joan Daemen และ Vincent Rijmen ในปี 2000 อัลกอริทึมได้รับการคัดเลือกโดยหน่วยงาน National Institute of Standard and Technology (NIST) ของสหรัฐอเมริกาให้เป็นมาตรฐานในการเข้ารหัสชั้นสูงของประเทศ อัลกอริทึมมีความเร็วสูงและมีขนาดกะทัดรัดโดยสามารถใช้กุญแจที่มีความยาวขนาด 128, 192 และ 256 บิต

อัลกอริทึม One-time Pads

อัลกอริทึมนี้ได้รับการยอมรับว่าเป็นอัลกอริทึมที่ไม่มีใครสามารถเจาะความแข็งแกร่งของอัลกอริทึมได้ อัลกอริทึมใช้กุญแจที่มีขนาดความยาวซึ่งอาจจะมากกว่าขนาดความยาวของข้อความที่ต้องการเข้ารหัส กุญแจจะถูกสร้างออกมาแบบสุ่มและโดยปกติจะถูกใช้งานแค่เพียงครั้งเดียวแล้วทิ้งไป แต่ละไบต์ของข้อความที่ต้องการส่งไปจะถูกเข้าและถอดรหัสโดยหนึ่งไบต์ (ชนิดไบต์ต่อไบต์) ของกุญแจที่ถูกสร้างขึ้นมาใช้งาน เนื่องจากกุญแจที่ถูกใช้งานแต่ละครั้งจะไม่ซ้ำกันและถูกสร้างขึ้นมาแบบสุ่ม จึงเป็นการยากที่จะค้นหากุญแจที่ถูกต้องได้

ข้อจำกัดของอัลกอริทึมนี้ คือขนาดของกุญแจที่อาจมีขนาดยาวกว่าข้อความที่ต้องการส่ง ซึ่งส่งผลให้การส่งมอบกุญแจที่มีขนาดใหญ่ทำได้ไม่สะดวกนัก รวมทั้งการสร้างกุญแจให้มีความสุ่มสูงไม่ใช่เป็นสิ่งที่ทำได้ง่ายนัก อย่างไรก็ตามอัลกอริทึมนี้ก็ยังมีการใช้งานในระบบเครือข่ายที่ต้องการความปลอดภัยสูง

อัลกอริทึมสำหรับการเข้ารหัสแบบกุญแจสาธารณะ (หรือการเข้ารหัสแบบอสมมาตร) อัลกอริทึมที่เป็นที่รู้จักกันดีมีดังนี้

อัลกอริทึม RSA

อัลกอริทึม RSA ได้รับการพัฒนาขึ้นที่มหาวิทยาลัย MIT ในปี 1977 โดยศาสตราจารย์ 3 คน ซึ่งประกอบด้วย Ronald Rivest, Adi Shamir และ Leonard Adleman ชื่อของอัลกอริทึมได้รับการตั้งชื่อตามตัวอักษรตัวแรกของนามสกุลของศาสตราจารย์ทั้งสามคน อัลกอริทึมนี้สามารถใช้ในการเข้ารหัสข้อมูลรวมทั้งการลงลายมือชื่ออิเล็กทรอนิกส์ด้วย

อัลกอริทึม DSS

DSS ย่อมาจาก Digital Signature Standard อัลกอริทึมนี้ได้รับการพัฒนาขึ้นมาโดย National Security Agency ในประเทศสหรัฐอเมริกาและได้รับการรับรองโดย NIST ให้เป็นมาตรฐานกลางสำหรับการลงลายมือชื่ออิเล็กทรอนิกส์ในประเทศสหรัฐอเมริกา

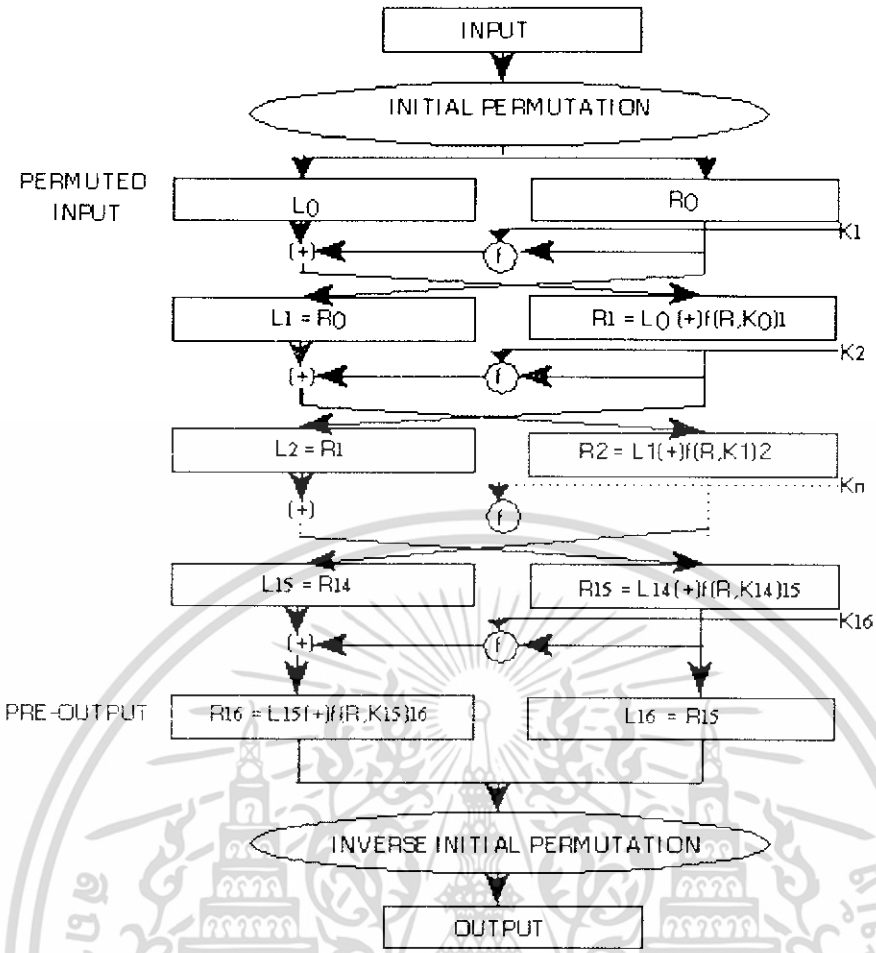
2.6 การเข้ารหัสลับแบบ DES

การทำงานของอัลกอริธึมแบบ DES นั้นเป็นพื้นฐานของการทำงานแบบ 3DES ดังนั้นในเบื้องต้นต้องเข้าใจการทำงานของ DES ซึ่งเป็นการเข้ารหัสลับที่เป็นแบบสมมาตร ใช้ขนาดข้อมูล 64 บิต และขนาดของกุญแจ 56 บิต การทำงานของอัลกอริทึม DES ได้แบ่งการทำงานออกเป็น 3 ส่วนที่สำคัญ คือ

1. การเรียงบิตใหม่ (Initial Permutation) เพื่อผลิต "permuted input" ซึ่งเป็นข้อมูลที่มีการสลับตำแหน่งจากเดิมไป

2. การเข้าฟังก์ชันการคำนวณ โดยมีกระบวนการทำงานแบบเดิมซ้ำกัน 16 ครั้ง ซึ่งผลลัพธ์ที่ได้จะได้เป็นข้อมูลขนาด 64 บิตอยู่ชุดหนึ่งที่เรียกว่า PREOUTPUT

3. การเรียงบิตกลับ (Inverse Initial Permutation) ซึ่งจะได้ผลลัพธ์ออกมาเป็นข้อมูลที่เข้ารหัสแล้ว

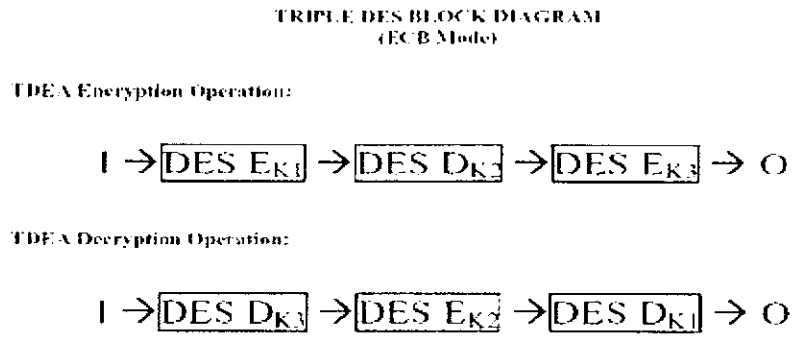


รูปที่ 2.9 กระบวนการเข้ารหัสด้วยอัลกอริธึม DES

สำหรับกระบวนการถอดรหัสนั้น ทำเช่นเดียวกันกับกระบวนการเข้ารหัส คือ ใส่ข้อมูลที่ถูกเข้ารหัสข้อมูลมาแล้วที่มีขนาด 64 บิต พร้อมกับกุญแจตัวเดิมที่ใช้ในการเข้ารหัส แล้วทำตามขั้นตอนต่างๆ เหมือนกับการเข้ารหัสข้อมูล แต่ต้องทำการเปลี่ยนลำดับของกุญแจใหม่ โดยใช้กุญแจจากลำดับท้ายสุด เรียงมาจนถึงลำดับแรกสุด คือ K16, K15,..., K1

2.7 การเข้ารหัสแบบ 3DES (Tripie Data Encryption Standard)

การเข้ารหัสแบบ 3DES นั้น เป็นการเข้ารหัสอีกแบบหนึ่ง โดยอิงมาตรฐานมาจากการเข้ารหัสแบบ DES ซึ่งเข้าและถอดรหัสแบบ DES เป็นจำนวน 3 ครั้ง และใช้กุญแจเป็นจำนวน 3 ตัว ในการทำงาน ดังนั้นเปรียบเสมือนว่าใช้กุญแจที่มีความยาว 56*3 คือ 168 บิต



รูปที่ 2.10 กระบวนการเข้าและถอดรหัสด้วย 3DES

โดยมีขั้นตอนในการเข้ารหัสข้อมูล ดังนี้คือ

1. เข้ารหัสข้อมูลโดยใช้กุญแจตัวแรก
2. ถอดรหัสข้อมูลโดยใช้กุญแจตัวที่สอง และ
3. เข้ารหัสข้อมูลโดยใช้กุญแจตัวที่สาม

ในส่วนของขั้นตอนของการถอดรหัสข้อมูล มีดังนี้คือ

1. ถอดรหัสข้อมูลโดยใช้กุญแจตัวแรก
2. เข้ารหัสข้อมูลโดยใช้กุญแจตัวที่สอง และ
3. ถอดรหัสข้อมูลโดยใช้กุญแจตัวที่สาม

2.8 Mode of operation

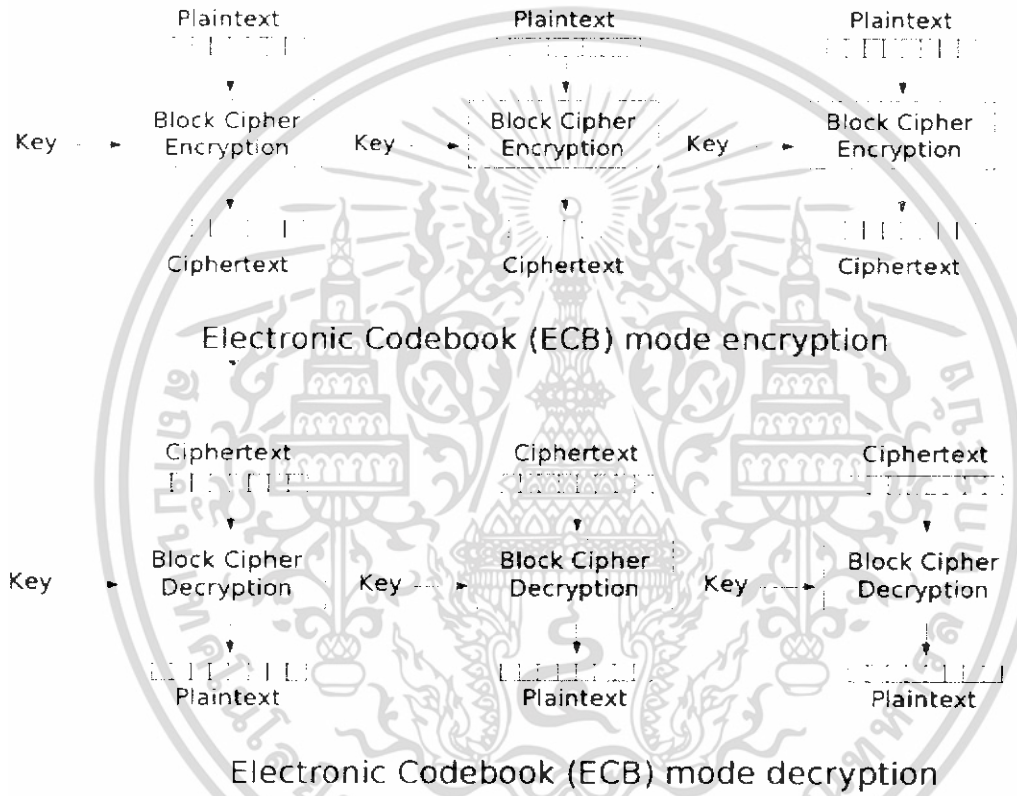
การเข้าและถอดรหัสลับนั้นอัลกอริธึมที่ใช้จะมองข้อความที่จะเข้าและถอดรหัสเป็นบล็อก ซึ่งความยาวของบล็อกมักเป็นค่าคงที่ขนาด 64 หรือ128 บิต ดังนั้นหากข้อความที่ต้องการเข้ารหัสลับนั้นมีค่าความยาวมากกว่านี้ก็จะต้องใช้ Mode of operation เข้ามาช่วยแบ่งข้อมูลที่ยาว ๆ นั้นให้เป็นหลายๆบล็อก ซึ่งจะมีหลาย operation ดังนี้

1. Electronic Codebook(ECB)
2. Cipher Block Chaining (CBC)
3. k-Bit Output Feedback Mode (OFB)
4. k-Bit Cipher Feedback Mode (CFB)
5. Counter Mode (CTR)

ขั้นตอนการทำงานของแต่ละวิธี

1. Electronic Codebook (ECB)

เป็นวิธีการที่ง่ายที่สุด โดยหลักการทำงานคือ นำเอาข้อความ(plaintext) มาแบ่งเป็นบล็อก เช่นขนาดบล็อกละ 64 บิตถ้าเหลือก็จะทำการเติมเต็มให้ครบ 64 บิต แล้วนำมาเข้ารหัสลับกับกุญแจลับที่เลือกไว้ทีละบล็อก โดยทุกบล็อกนั้นใช้กุญแจตัวเดียวกัน และในการถอดรหัสลับก็ทำในลักษณะเดียวกัน คือนำไซเฟอร์(cipher)ที่ได้จากการเข้ารหัสมาถอดรหัสด้วยกุญแจตัวเดียวเดิมซึ่งเหมือนทุกบล็อก ลักษณะการทำงานเป็นดังรูปที่ 2.11 แสดงไว้ข้างล่างนี้



รูปที่ 2.11 แสดงการเข้าและถอดรหัสลับแบบ ECB

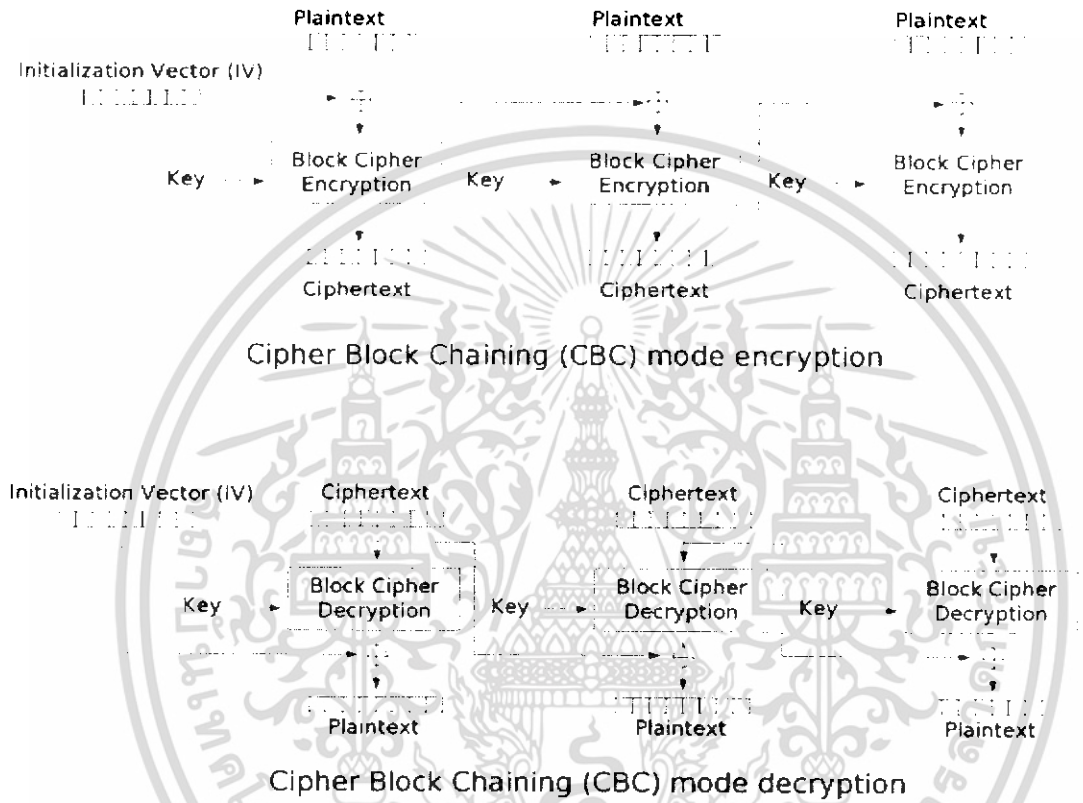
ข้อเสียของวิธีการนี้มีอยู่ 2 ประการคือ

1. เนื่องจากทุกบล็อกนั้นใช้กุญแจเดิม ดังนั้นหากข้อความเดิมที่เหมือนเดิมก็จะทำให้ได้ไซเฟอร์ออกมาเหมือนกันทำให้สามารถนำไปสู่การคาดเดาที่ง่ายขึ้นได้

2. ปัญหาด้านการตรวจสอบความถูกต้อง เพราะหากทำการสลับไซเฟอร์กันก็สามารถทำได้เพราะแต่ละบล็อกไม่ได้มีความต่อเนื่องกัน

2. Cipher Block Chaining (CBC)

เป็นการนำ ECB มาพัฒนา แก้ไขในจุดที่ยังเป็นปัญหาของ ECB โดยมีการนำ initial vector เข้ามาช่วย ป่วนให้เกิดความวุ่นวายมากขึ้นซึ่งเป็นค่าที่ได้จากการสุ่ม โดย IV ที่ได้นั้นเอามา XOR กับบล็อกข้อความ แล้วทำการเข้ารหัสลับเพื่อให้ได้เป็นไซเฟอร์ออกมาแล้วเอาไซเฟอร์นั้นไปเป็น IV สำหรับข้อความบล็อกต่อไปตามลำดับ



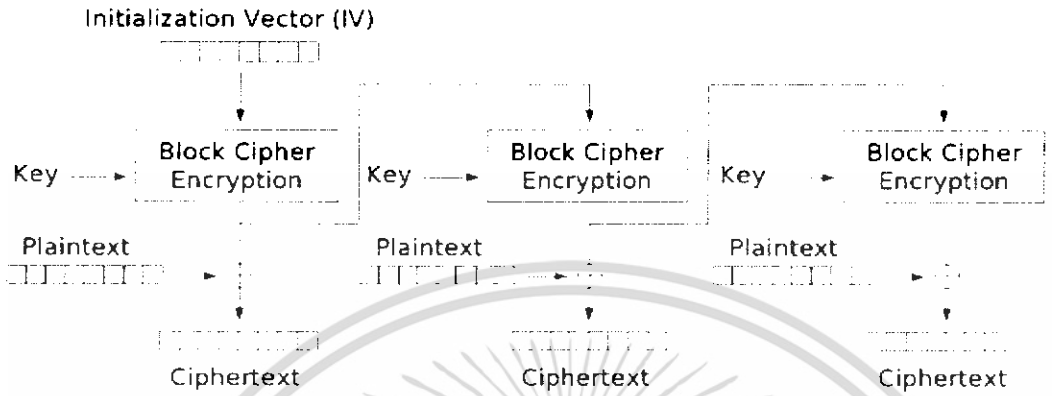
รูปที่ 2.12 แสดงการเข้าและถอดรหัสลับแบบ CBC

ข้อดีของวิธีการนี้คือ แม้ว่าข้อความจะเป็นข้อความเดิมแต่จะได้ Cipher ที่ไม่เหมือนเดิม เพราะว่าใช้ค่า IV ที่ต่างไปจากเดิมและแก้ไขการสลับบล็อกได้ในระดับหนึ่งเพราะหากบล็อกเปลี่ยนที่ค่าที่ได้จากการถอดรหัสออกมาจะเปลี่ยนไป

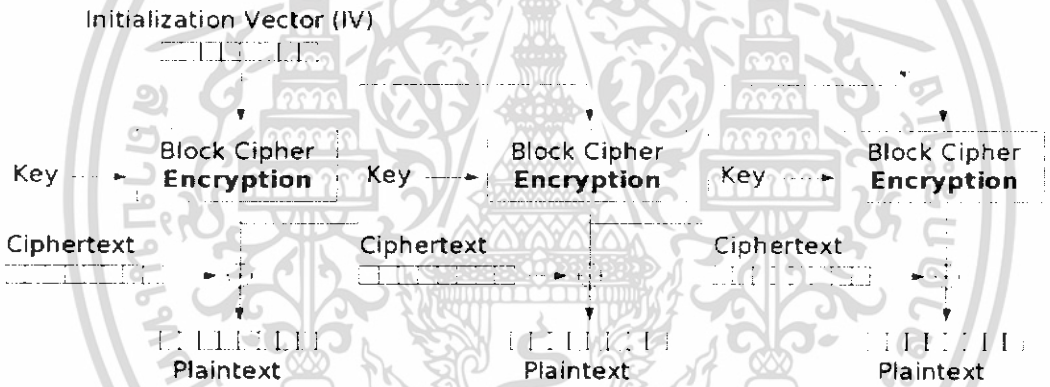
ข้อเสียของวิธีการนี้คือ เปลืองแบนด์วิธ เพราะเวลาส่งต้องส่งทั้ง Cipher และ IV ไปด้วย และยังสามารถทำการเปลี่ยนแปลงบิตได้อยู่ดีทำให้ความหมายที่ออกมาผิดไป

3. k-Bit Output Feedback Mode (OFB)

เป็น Stream cipher ที่ XOR message แบบ one time padding ทำให้เกิดบล็อกข้อมูลขนาดไหนก็ได้ไม่จำเป็นต้องเป็น 64 หรือ 128 บิต ทำให้ไม่จำเป็นต้องมีการเติมเต็มบิตหากข้อมูลไม่เต็มบล็อก



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

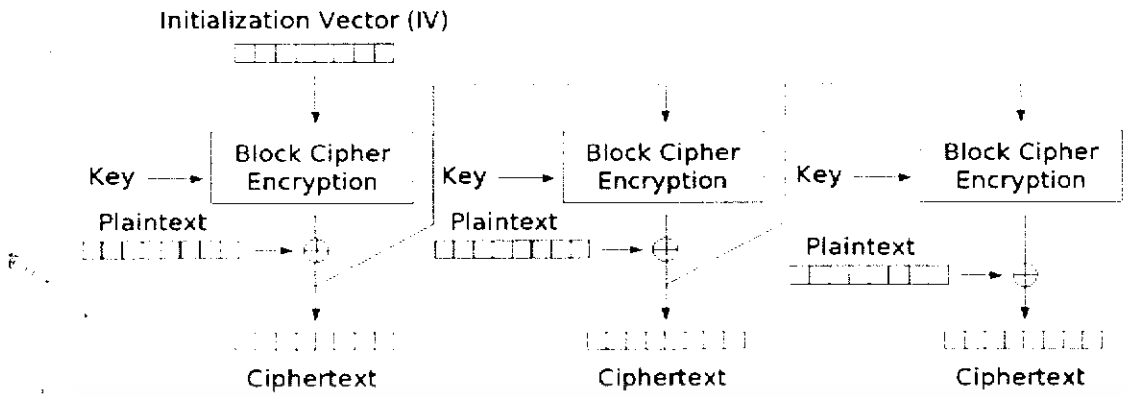
รูปที่ 2.13 แสดงการเข้าและถอดรหัสลับแบบ OFB

ข้อดี

1. สามารถหา subkey ไว้ก่อนได้ทำให้สามารถทำได้เร็ว
2. ข้อมูลไม่จำเป็นต้องเต็มบล็อก ทำให้ไม่ต้องทำการเติมเต็มบิตข้อมูล

4. k-Bit Cipher Feedback Mode (CFB)

คล้ายกับการทำงานของ OFB แต่ค่าที่ใช้ในการเข้ารหัสลับเก็บบล็อกต่อไปนั้นใช้ค่าที่ได้จากการเข้ารหัสลับของบล็อกที่ผ่านมา



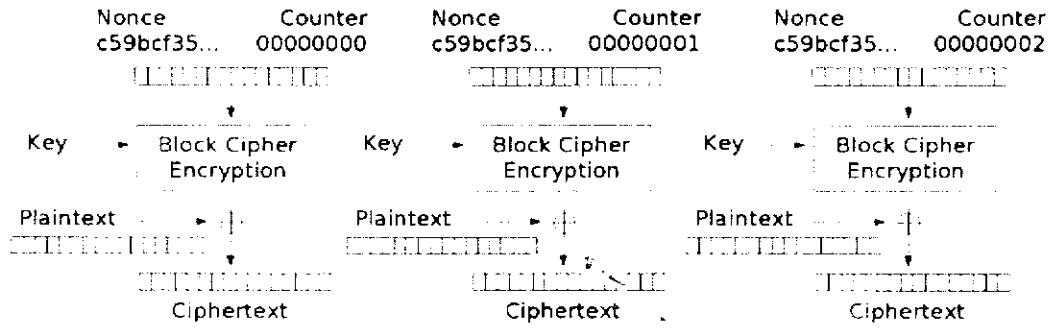
รูปที่ 2.14 แสดงการเข้าและถอดรหัสลับแบบ CFB

ข้อดีของวิธีการนี้คือ ไม่สามารถเปลี่ยนแปลง Cipher ได้เพราะ Cn มีผลต่อการคิดในบล็อกต่อไป

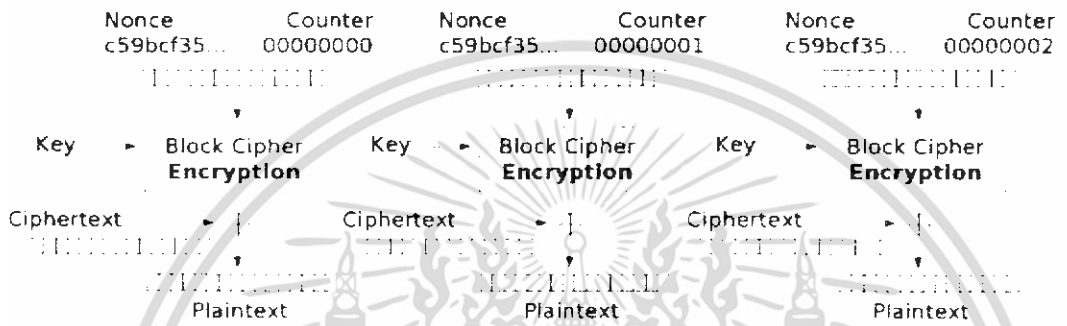
ข้อเสียของวิธีการนี้คือ ซับซ้อนและยุ่งยาก ไม่สามารถทำงานแบบขนาน(Parallel) ไม่สามารถทำการสร้างกุญแจ (generate key) ไว้ล่วงหน้าได้

5. Counter Mode (CTR)

คล้ายกับการทำงานของ feedback mode แต่จะเปลี่ยนค่าตรง register ไปเป็นค่าเวกเตอร์ตั้งต้น (initial vector) แทน ข้อดีของวิธีการนี้คือ สามารถแยกงานทำได้คือ สามารถใช้ thread แยกงานกันทำบล็อกไหนก็ได้ ถ้าทำการสลับบล็อกไซเฟอร์แล้วจะทำให้การถอดรหัสลับนั้นมีปัญหาเพราะว่าค่าของเวกเตอร์ตั้งต้นเป็นตัวบังคับลำดับของบล็อก



Counter (CTR) mode encryption



Counter (CTR) mode decryption

รูปที่ 2.15 แสดงการเข้าและถอดรหัสลับแบบ CTR

สำหรับการเข้าและถอดรหัสลับแบบ 3DES นั้น จะใช้ CBC ซึ่งสามารถทำได้ 2 แบบ คือ CBC Outside และ CBC Inside โดยสองแบบนี้จะมีลักษณะที่แตกต่างกันคือ

ตารางที่ 2.1 เปรียบเทียบคุณสมบัติของ CBC inside และ CBC outside

CBC inside	CBC outside
เป็นวิธีการที่มีความซับซ้อนมาก	เป็นที่นิยมมากกว่าเพราะว่ามองแยกเป็นส่วนๆ
ดีกว่า outside ในเชิงด้านความปลอดภัย	มีความสะดวกในการทำงานมากกว่า

2.9 อัลกอริธึม AES

การเข้ารหัสแบบ AES (Advanced Encryption Standard)

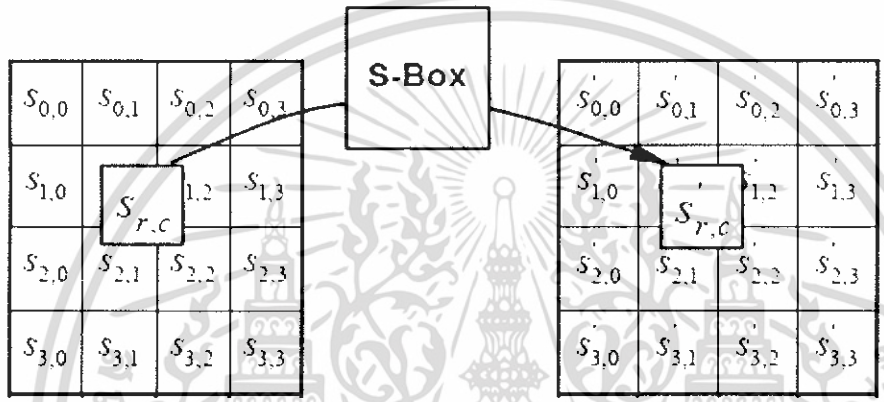
AES หรือที่รู้จักว่า Rijndael คือการเข้ารหัสข้อมูลโดยจะแบ่งขนาดบล็อกเป็นบล็อกละ 128 บิต และมีการใช้กุญแจขนาด 128,192 หรือ 256 บิตแต่ถ้าตรงตามมาตรฐานจะใช้กุญแจขนาด 128 บิตด้วย ซึ่งการเข้าและถอดรหัสลับนั้นเป็นการเข้ารหัสแบบสมมาตร (Symmetric Cryptosystem)

คือใช้กุญแจตัวเดียวกันในการเข้าและถอดรหัส โดยจะมองเป็นเมตริก 2 มิติขนาด 4*4 โดยบล็อกละ 8 บิต โดยทำงานตามสี่ขั้นตอนนี้ให้ครบ 10 รอบ

การทำงานของอัลกิริธึม AES แบ่งเป็น 2 ส่วนคือ ส่วนของข้อความและส่วนของกุญแจที่ใช้

ลำดับการทำงานของส่วนข้อความแบ่งออกเป็น 4 ขั้นตอนที่สำคัญคือ

1. การเทียบเปลี่ยนค่า (SubBytes) นำเอาข้อความที่ต้องการนำมาเข้ารหัสลับ มาแบ่งเป็นบล็อกขนาดบล็อกละ 8 บิตทั้งหมดได้ความยาว 128 บิต แล้วนำเอาค่าที่อยู่ในข้อความตั้งต้นเทียบค่ากับตาราง s-box แล้วได้เป็นค่าใหม่ออกมา

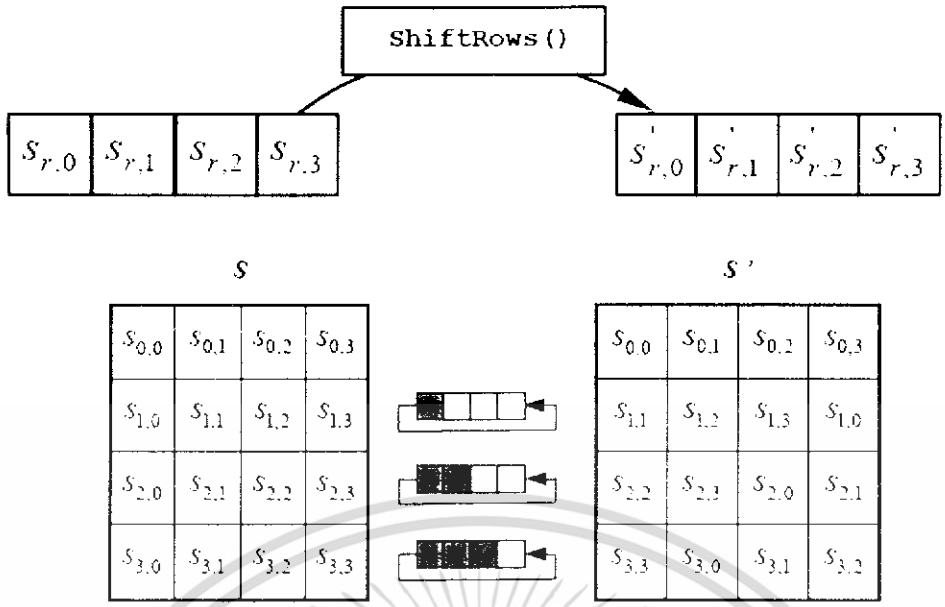


รูปที่ 2.16 แสดงค่าที่ได้ออกมาโดยใช้ s-box ในการเทียบแต่ละไบต์

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

รูปที่ 2.17 แสดง s-box ได้ค่าโดยเทียบไบต์ xy (แสดงออกมาในรูปแบบเลขฐาน 16)

2. การเลื่อนแถว (ShiftRows) เป็นการเปลี่ยนแปลงข้อความที่ถูกเข้ารหัสโดยการเลื่อน 3 แถวล่างเนื่องจากมี 4 แถว เริ่มจาก 0-3 เริ่มเลื่อนแถวที่ 1 เลื่อนซ้ายไป 1 บล็อก แถวที่ 2 เลื่อนไป 2 บล็อก แถวที่ 3 เลื่อนไป 3 บล็อก เป็นการเลื่อนวนไปทางซ้ายดังรูป



รูปที่ 2.18 แสดงการเลื่อนวนทางซ้ายที่ 3 แถวล่าง

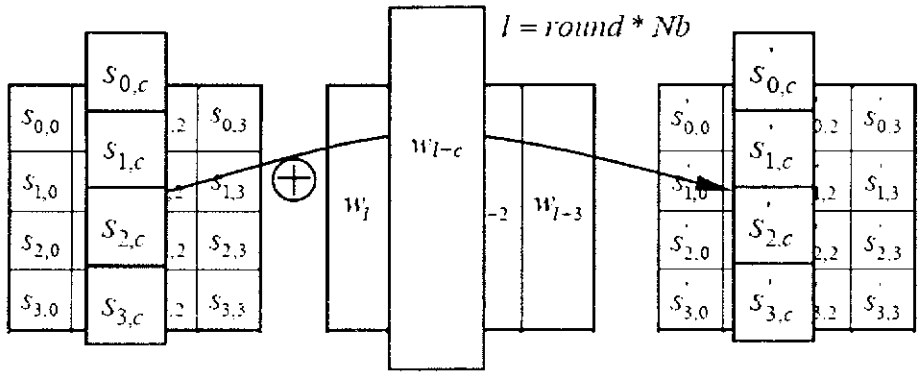
3. การผสมคอลัมน์ (MixColumns) เป็นการคิดเทียบต่อคอลัมน์โดยคิดแบบโพลิโนเมียลผ่านฟังก์ชัน GF^8 ซึ่งจะแสดงเป็นการคำนวณทางเมตริกซ์ตามรูปข้างล่างนี้ เพื่อให้ค่าที่ออกมาได้จากการผสมแถวให้มีค่าเปลี่ยนไป

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

สำหรับ 0 <= c <= 4

รูปที่ 2.19 แสดงการผสมคอลัมน์

4. การนำกุญแจมาผสมกับข้อความ (AddRoundKey) เป็นการ XOR แต่ละบล็อกของค่าที่ได้จากการขั้นตอนที่ผ่านมากับกุญแจที่ได้จากการคำนวณในแต่ละรอบ



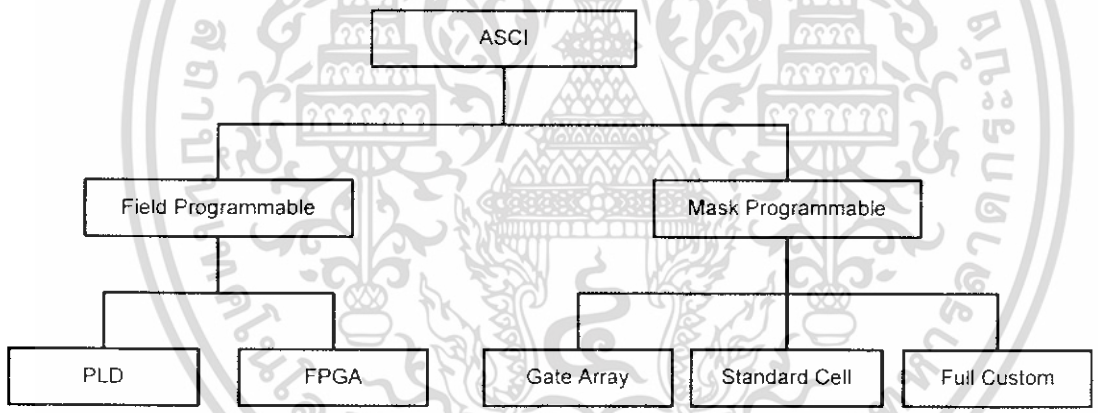
รูปที่ 2.20 แสดงผลลัพธ์ที่เกิดจากการ XOR กุญแจกับค่าแต่ละบล็อก

บทที่ 3
เทคโนโลยีเอฟพีจีเอ

ในด้านของอุตสาหกรรมอิเล็กทรอนิกส์มีความก้าวหน้ามากขึ้นจากเดิม เป็นผลให้เกิดการพัฒนา ความสามารถของอุปกรณ์ต่างๆมากกมาย ซึ่งสามารถลดค่าใช้จ่ายต่างๆ ลดการสิ้นเปลืองพลังงาน และลดขนาดของอุปกรณ์ลง แต่ในขณะเดียวกันก็ได้ทำการเพิ่มประสิทธิภาพและระดับความน่าเชื่อถือของอุปกรณ์ให้สูงขึ้นด้วย

นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตโดยทำการเพิ่มความหนาแน่นและจำนวนของฟังก์ชันโลจิก (Logic Function) ให้เหมาะสม ซึ่งผลิตเป็นปริมาณมาก และทำการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ขึ้น โดยแบ่งวงจรรวมตามการสร้างออกได้เป็น 2 ชนิด คือ

- Field Programmable
- Mark Programmable



รูปที่ 3.1 ผังการแบ่งกลุ่มของวงจรรวม ASIC

3.1 Field Programmable

ผู้ใช้สามารถออกแบบและสร้างวงจรที่ต้องการใช้ได้เอง โดยไม่ต้องไปโรงงานเพื่อผลิต และในปัจจุบันนี้มีเครื่องมือที่ใช้ช่วยในการออกแบบและสร้างวงจรรวมที่มีความสามารถสูง ตั้งแต่ขั้นตอนการออกแบบ การจำลองการทำงาน จนถึงการจัดสร้างวงจรลงในอุปกรณ์ ซึ่งอุปกรณ์ Field programmable สามารถหาซื้อได้ง่าย เป็นผลให้การสร้างวงจรอิเล็กทรอนิกส์จนถึงระบบไมโครโพรเซสเซอร์ หันมาใช้อุปกรณ์จำพวกนี้เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อยๆแยกชิ้น (Discrete component) กันมากขึ้น

3.1.1 พีแอลดี (PLD: Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านโลจิกต่อกันอยู่เป็นกลุ่มมีทั้งวงจรคอมบิเนชัน (Combination) และซีเควนต์เชียล (Sequential) อุปกรณ์พีแอลดีทุกชนิดล้วนมีหลักการพื้นฐานของวงจรภายในที่เหมือนกัน โดยมีวงจรหลักเป็นวงจรคอมบิเนชันที่ให้ผลเป็น ผลคูณร่วมบวก (Sum of product) ซึ่งประกอบไปด้วยชุดของแอนด์เกตที่ต่อร่วมกับออเกต การโปรแกรม คือ การเลือกว่าจะให้มีการต่ออินพุตภายในของแอนด์เกตกับสัญญาณอินพุตใดบ้าง อินพุตต่างๆของอุปกรณ์ทุกตัว จะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใด จะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว

3.1.2 แอลซีเอ (LCA: Logic Cell Array)

อุปกรณ์ชนิดนี้ถูกสร้างเป็นอาร์เรย์ที่ประกอบด้วยเกตจำนวน 1,200 – 1,800 เกต มีลักษณะของสถาปัตยกรรมที่ใกล้เคียงกับเกตอาร์เรย์ (Gate Array) โดยใช้โปรเซสแบบซีมอส 1.6 ไมครอน ขึ้นโลหะคู่ (CMOS 1.6 microns double-layer metal) สามารถโปรแกรมและลบได้โดย ใช้กระแสไฟฟ้า (Static RAM based) ภายในจัดเรียงเป็นเมทริกซ์ (Matrix) ของโลจิกเซลล์ (Logic Cell) ล้อมรอบด้วยอินพุต เอาต์พุตเซลล์ แต่ปัจจุบันได้พัฒนามาเป็น เอฟพีจีเอ (FPGA: Filed Programmable Gate Array) ซึ่งมีประสิทธิภาพความจุของเกตสูงมากขึ้น โดยสร้างออกมาเป็นอนุกรม (Series) ต่างๆ เช่น ตระกูล XC3000 และตระกูล XC4000 เป็นต้น

3.1.3 เอฟพีจีเอ (FPGA: Field Programmable Gate Array)

เป็นอุปกรณ์ที่พัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทไซรินซ์ (XILINX Inc.) ซึ่งสามารถจะกำหนดฟังก์ชันการทำงานได้ตามความต้องการของผู้ใช้โดยผ่านการกำหนดฟังก์ชันการทำงานของวงจร และเนื่องจากเอฟพีจีเอมีประสิทธิภาพการทำงานและปริมาณความหนาแน่นของเกตสูง ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอจึงสามารถออกแบบและทดสอบเสร็จภายในเวลาเพียง 2-3 วันเท่านั้น และการเปลี่ยนแปลงแก้ไข ก็ใช้เวลาไม่นาน ซึ่งประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์ และลดค่าเอ็นอาร์อี (NRE: Non-Recurring Engineering cost) ลงไปด้วย

3.2 Mark Programmable

สำหรับการใช้งานวงจรรวม ASIC ในเชิงพาณิชย์นั้น จำเป็นต้องใช้วงจรรวม ASIC แบบ Mark Programmable เนื่องจากว่า หากมีปริมาณการผลิตที่สูงเป็นหมื่นตัวนั้น จะมีราคาต้นทุนต่ำกว่าแบบ Field Programmable ดังนั้น การใช้งานวงจรรวมแบบ Mark Programmable จึงมีบทบาทสำคัญในการผลิตสินค้า อิเล็กทรอนิกส์ในเชิงพาณิชย์อย่างมากในปัจจุบัน

3.3 เทคโนโลยีของเอฟพีจีเอ

เป็นชิปที่สามารถทำการโปรแกรมได้ โดยการกำหนดุดเชื่อมต่อเพื่อสร้างจุดเชื่อมต่อภายในเพื่อ
ประกอบเป็นวงจรได้ตามต้องการ ซึ่งสามารถแบ่งลักษณะของจุดเชื่อมต่อได้ดังนี้

3.3.1 Physical Changing

- Fused คือ สามารถโปรแกรมได้เพียงครั้งเดียว หลังจากโปรแกรมแล้ว จุดเชื่อมต่อจะขาดจากกัน
- Anti-fused คือ สามารถโปรแกรมได้เพียงครั้งเดียว หลังจากโปรแกรมแล้ว จุดเชื่อมต่อจะต่อถึงกัน

3.3.2 Memory-Based

เป็นการใช้หน่วยความจำเป็นตัวเก็บข้อมูล เพื่อควบคุมจุดเชื่อมต่อต่างๆ

- ROM based คือ สามารถโปรแกรมได้หลายครั้ง ซึ่งข้อมูลจะคงอยู่ได้โดยไม่ต้องใช้ไฟเลี้ยงมีความจุต่ำ ส่วนใหญ่ไม่เกิน 20,000 เกต
- RAM based คือ สามารถโปรแกรมได้ตลอดไม่จำกัด แต่จำเป็นที่จะต้องใช้ไฟเลี้ยง มีความจุสูง ประมาณ 10,000 – 1,000,000 เกต

เหตุผลที่สามารถทำการออกแบบเอฟพีจีเอได้ง่ายและรวดเร็ว เพราะ

- ไม่ต้องรู้โครงสร้างของชิป เพียงแค่ทำการออกแบบโลจิกที่ต้องการเท่านั้น
- ใช้ภาษาเอชดีแอล (HDL) ในการออกแบบ หรือใช้ซอฟต์แวร์ มากำหนดลักษณะการทำงาน โดยภาษาที่เป็นที่นิยมใช้กันมากที่สุด คือ ภาษาวีเอชดี ซึ่งสามารถทำให้ใช้ได้กับชิปทุกค่าย
- สามารถโปรแกรมให้ชิปทำงานได้โดยตรง โดยไม่จำเป็นต้องถอดตัวชิปออกมาข้างนอก หรือจะใช้เป็นบอร์ดโปรแกรมมาทำงานก็ได้ และสามารถทำการโปรแกรมได้หลายครั้ง ทำให้สามารถพัฒนาและปรับปรุงความมีประสิทธิภาพได้ง่าย
- IC FPGA ที่นิยมใช้และพอหาได้ ในบ้านเรา มีอยู่ 2 ค่ายใหญ่ด้วยกัน คือ ALTERA กับ XILINX

3.4 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล Spartan 3

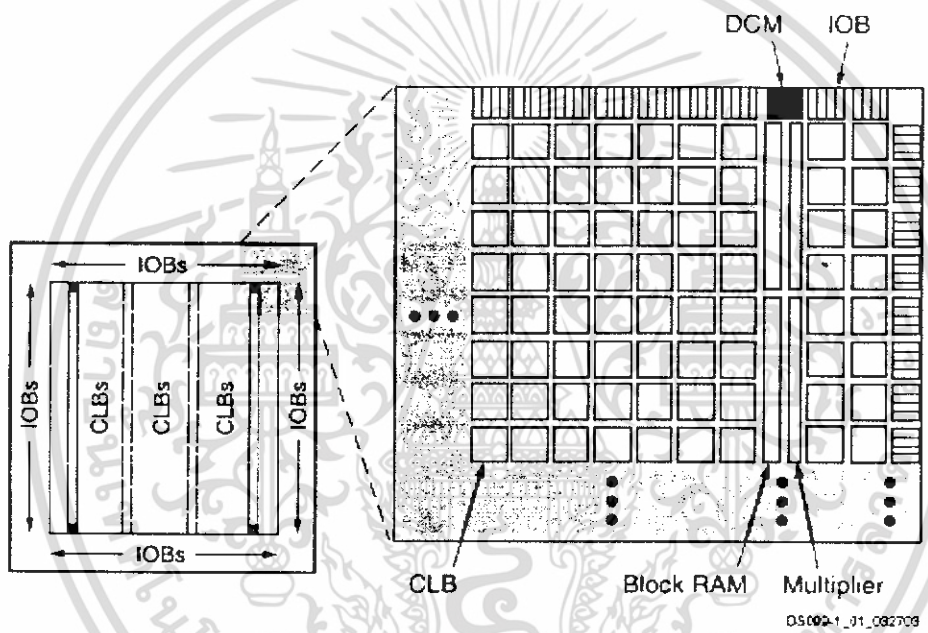
เอฟพีจีเอเป็นชิปที่มีสนามวงจรขนาดใหญ่อยู่ภายใน มีความเร็วสูง และมีสถาปัตยกรรมการออกแบบคล้ายซีพีแอลดี (CPLD: Complex Programmable Logic Device) แต่มีส่วนประกอบที่

ซับซ้อนและมีประสิทธิภาพมากกว่า โดยสามารถนำเอฟพีจีเอมาใช้และออกแบบวงจรต่างๆได้ตามที่เราต้องการโดยกระทำผ่านโปรแกรมที่ใช้ในการออกแบบ ซึ่งสามารถทำได้ง่าย ดังนั้น เอฟพีจีเอจึงเหมาะสมสำหรับการออกแบบวงจรเป็นอย่างมาก

สถาปัตยกรรมภายในของเอฟพีจีเอ สามารถแบ่งออกเป็น 3 ส่วน คือ

- ซีแอลบี (CLB: Configuration Logic Block)
- ไอโอบี (IOB: Input/Output Block) และ
- การเชื่อมต่อภายใน (Interconnect)

ภายในสถาปัตยกรรมของเอฟพีจีเอ จะมีลักษณะเป็นตารางโลจิกบล็อก (Logic Block) และล้อมรอบไปด้วยบล็อกของไอโอ (I/O Block) และการเชื่อมต่อระหว่างซีแอลบี กระทำโดยผ่านช่องว่างระหว่างแถวและคอลัมน์ (Column)



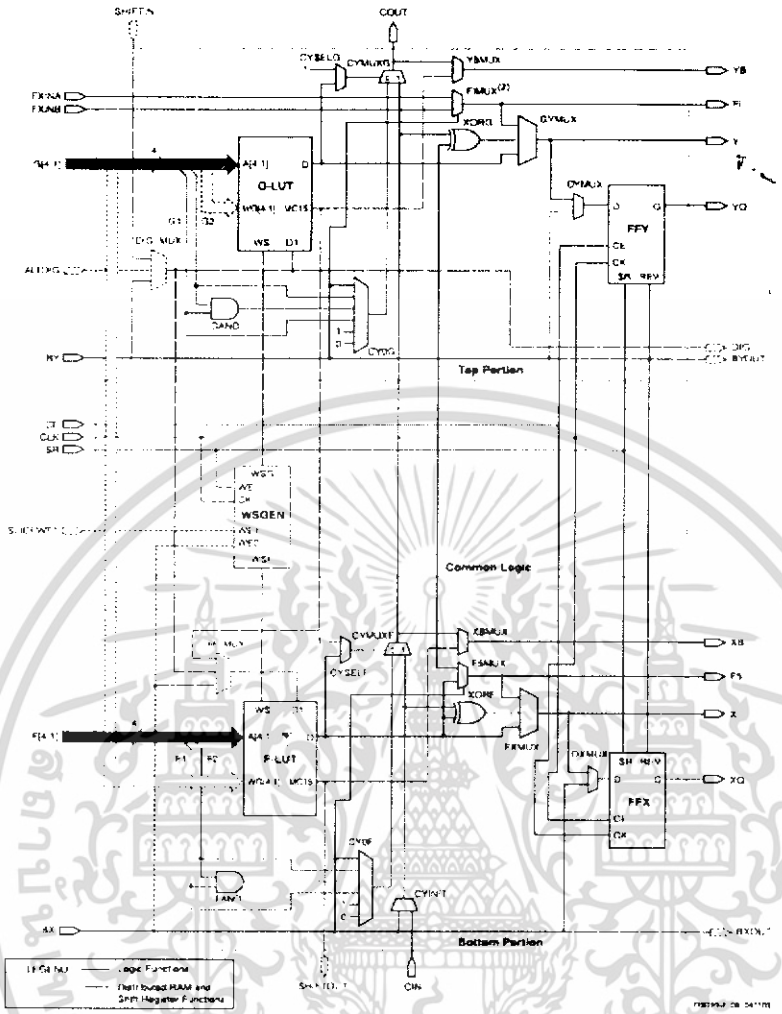
รูปที่ 3.2 วงจรพื้นฐานภายในเอฟพีจีเอ

อุปกรณ์เอฟพีจีเอทำงานได้โดยใช้ Program-driven logic device ซึ่งทำหน้าที่เป็นตัวระบุการทำงานของทั้ง 3 ส่วน ซึ่งค่าต่างๆได้ถูกกำหนดไว้ในโปรแกรมคอนฟิกกูเรชัน (Configuration Program)หรือเก็บไว้ในอีพรอม ซึ่งอยู่ภายในเอฟพีจีเอ โดยเอฟพีจีเอจะเริ่มต้นทำงานเมื่อตัวโปรแกรมถูกโหลดเข้าสู่เอฟพีจีเอแล้วประสิทธิภาพของเอฟพีจีเอถูกกำหนดโดยความเร็วของโลจิกส่วนประกอบของหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ

ซีแอลบี(CLB)

แต่ละตัวประกอบด้วยหน่วยของคอมบิเนชันโลจิก (Combination Logic) ที่สามารถโปรแกรมได้ โดยสามารถนำมาใช้สร้างวงจรทางด้านฟังก์ชันบูลีน (Boolean Function) ของอินพุต

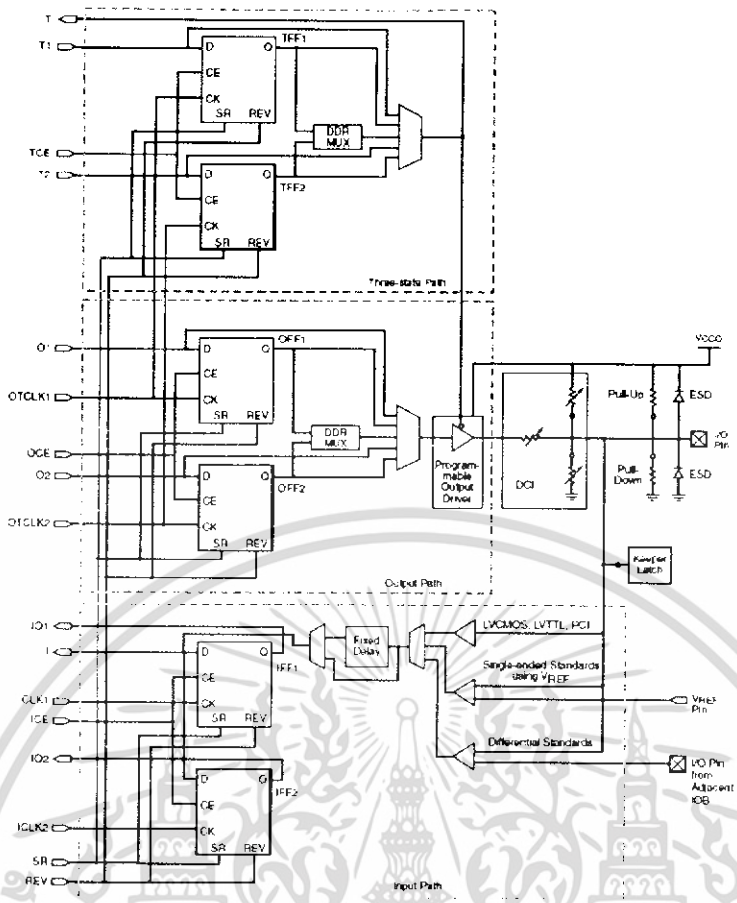
ได้ และรีจีสเตอร์ (Register) ที่ทำหน้าที่เก็บข้อมูล ซึ่งรับค่าจากส่วนคอมบิเนชัน(Combination) หรือรับค่าโดยตรงจากเอาต์พุตของซีแอลบี



รูปที่ 3.3 ผังวงจรภายในของไอโอบีของเอฟพีจีเอ

ไอโอบี(IOB)

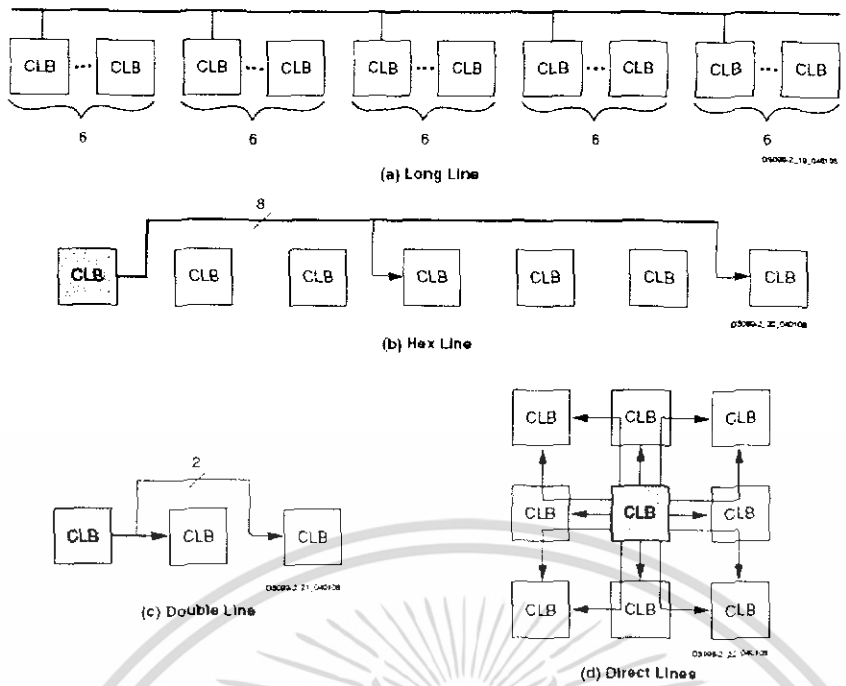
เป็นส่วนที่ติดต่อกับวงจรภายนอกของเอฟพีจีเอ ซึ่งมาจากส่วนของอุปกรณ์อินพุต และเอาต์พุตที่สามารถโปรแกรมได้ (Programmable I/O device) โดยแต่ละตัวโปรแกรมได้อย่างอิสระ สามารถให้เป็นอินพุตและเอาต์พุตแบบ 3 สถานะ หรือไอโอ (I/O: Input/Output) แบบสองทิศทางก็ได้ โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอล (TTL) และซีมอสเทรดโฮลของไอโอบี ซึ่งแต่ละตัวล้วนมีฟลิปฟลอป (flip-flop) ที่สามารถใช้เป็นบัฟเฟอร์ (Buffer)สำหรับอินพุตและเอาต์พุตได้



รูปที่ 3.4 ผังวงจรภายในของซีแอลบีของเอฟพีจีเอ

การเชื่อมต่อภายใน

การเชื่อมต่อภายในเอฟพีจีเอ ประกอบด้วยเน็ตเวิร์ค (Network) 2 ทิศทาง คือ ทางแถวและทางคอลัมน์ ซึ่งวางอยู่ระหว่าง CLB programmable switch และจะทำการเชื่อมต่ออินพุตและเอาต์พุตของไอโอบี และซีแอลบีที่จุดเชื่อมต่อร่วมระหว่างแถวกับคอลัมน์ ซึ่งสามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่างๆได้



รูปที่ 3.5 ผังการเชื่อมต่อภายในของเอฟพีจีเอ

บทที่ 4
การเขียนดีไวซ์ไดรเวอร์

4.1 การเขียนไดรเวอร์สำหรับลีนุกซ์

อุปกรณ์แต่ละชนิดย่อมมีลักษณะการทำงานที่แตกต่างกันออกไป ซึ่งโอเอส (OS) มีหน้าที่ควบคุมและติดต่อกับอุปกรณ์ต่าง ๆ ทุกประเภท ดังนั้นโอเอสจึงจำเป็นต้องรู้ลักษณะการทำงานของอุปกรณ์ทุกประเภท เพื่อจะได้สามารถควบคุมการทำงานของอุปกรณ์แต่ละประเภทได้ถูกต้อง แต่หากเรานำเอาโปรแกรมที่เป็นส่วนควบคุมอุปกรณ์ทั้งหมดมาใส่รวมไว้ในตัวโอเอส จะทำให้โอเอสมีขนาดใหญ่มาก ผู้ออกแบบโอเอสจึงมีความคิดที่จะแยกเอาส่วนที่ควบคุมอุปกรณ์ต่าง ๆ ออกจากตัวโอเอส ซึ่งโปรแกรมที่แยกออกมานี้ เรียกว่า ดีไวซ์ไดรเวอร์ (Device Driver) เมื่อโอเอสต้องการที่จะติดต่อกับอุปกรณ์ใด โอเอสจะทำการติดต่อผ่านทางดีไวซ์ไดรเวอร์ชนิดนั้นแล้วดีไวซ์ไดรเวอร์จะทำการติดต่อกับอุปกรณ์จริงเอง

ความหมายของดีไวซ์ไดรเวอร์

ดีไวซ์ไดรเวอร์เป็นโปรแกรมพิเศษที่ถูกโหลดเมื่อระบบทำงาน มีหน้าที่ในการเชื่อมต่อการทำงานระหว่างแอพพลิเคชัน (Application) ของผู้ใช้งานกับอุปกรณ์ฮาร์ดแวร์ (Hardware) ซึ่งควรออกแบบให้มีความยืดหยุ่นในการใช้งาน ไม่จำกัดการใช้งานฮาร์ดแวร์ โดยให้ตัวแอพพลิเคชันเป็นตัวเลือกการทำงานเอง ส่วนดีไวซ์ไดรเวอร์ทำหน้าที่เพียงจัดการทรัพยากร (Resource) ภายในของคอมพิวเตอร์ให้เหมาะสมแก่การใช้งาน

ลีนุกซ์จะแยกประเภทของดีไวซ์ (Device) ออกเป็นหลัก ๆ 3 ชนิด แต่ละมอดูลปกติจะสร้างหนึ่งในสามแบบนี้คือ char module, block module และ network module

- **คาแร็กเตอร์ดีไวซ์ (Character Devices)**

คาแร็กเตอร์ดีไวซ์เป็นแบบหนึ่งที่สามารถเข้าถึงได้โดยกระแสข้อมูลแบบไบต์ (stream of bytes)เหมือนกับไฟล์ ดังนั้นการสร้างฟังก์ชันการทำงานจึงเหมือนกับฟังก์ชันของไฟล์ เช่นอย่างน้อยต้องสร้างมีฟังก์ชัน open(), close(), write() และ read() ที่เป็นซิสเต็มคอลล์ (system calls) ยกตัวอย่างคาแร็กเตอร์ดีไวซ์เช่น ซีเรียลพอร์ต (serial ports)

- **บล็อกดีไวซ์ (Block Devices)**

บล็อกดีไวซ์คล้ายกับคาแร็กเตอร์ดีไวซ์ แต่บล็อกดีไวซ์จะถูกเข้าถึงได้ด้วยไฟล์ซิสเต็ม (file system nodes) ในไดเร็กทอรี /dev บล็อกดีไวซ์สามารถจัดการได้ด้วยไอโอโอเปอร์เรชั่น (I/O Operations) ที่เคลื่อนย้ายข้อมูล (transfer) เป็นบล็อก ๆ เท่านั้น ความแตกต่างกันระหว่างคาแร็กเตอร์ดีไวซ์และบล็อกดีไวซ์คือการจัดการกับการเคลื่อนย้ายข้อมูล

- **เน็ตเวิร์คดีไวซ์ (Network Devices)**

อุปกรณ์ที่ทำหน้าที่รับหรือส่งแพ็กเกตข้อมูล ซึ่งทำงานร่วมกับบีเอสดีซ็อกเก็ต (BSD Socket) ผ่านข้อมูลโดยใช้ซ็อกเก็ตบัฟเฟอร์ (Socket Buffer) เพื่อประมวลผลตามมาตรฐานของโพรโตคอล (Protocol) ชั้นต่าง ๆ จนถึงการติดต่อกับอุปกรณ์ฮาร์ดแวร์จริง

การสร้างและรันงานมอดูล (Building and Running Modules)

การเขียนดีไวซ์ไดรเวอร์ คือ การเขียนโปรแกรมเพื่อรันในส่วนของลีนุกซ์เคอร์เนล ซึ่งไดรเวอร์จะรันบนเคอร์เนลโหมด (Kernel mode) โดยไดรเวอร์บนลีนุกซ์จะถูกแบ่งออกเป็น 2 แบบ คือแบบบิวลท์อิน (Build-in) และแบบมอดูล (Module) ซึ่งจะต่างกันที่ช่วงเวลาที่ใช้เรียกการทำงาน

ลีนุกซ์มอดูลเป็นเซตของฟังก์ชันและชนิดของข้อมูล ซึ่งสามารถจะคอมไพล์ (Comply) แยกจากตัวเคอร์เนลได้ และจะถูกเรียกใช้งานหลังจากที่บูตเคอร์เนลแล้ว มอดูลจะใช้ในการอิมพลีเมนต์ (Implement) ดีไวซ์ไดรเวอร์ ซึ่งจะใช้ติดต่อกับอุปกรณ์เฉพาะของไดรเวอร์นั้นเท่านั้น

ขั้นตอนในการเขียนโปรแกรมดีไวซ์ไดรเวอร์

1. เรียนรู้การทำงานภายในของโอเอสที่ใช้งาน เช่น วินโดวส์ หรือ ลีนุกซ์ เป็นต้น
2. เรียนรู้ว่าจะเขียนดีไวซ์ไดรเวอร์อย่างไรบนโอเอสที่ใช้งานอยู่เช่น DDK, DDI หรือ DKI เป็นต้น
3. หาทูล (Tool) มาใช้สำหรับการพัฒนาและการดีบักในเคอร์เนลโหมด
4. ทำการเขียนดีไวซ์ไดรเวอร์บนเคอร์เนลโหมด ซึ่งเป็นไดรเวอร์พื้นฐานของฮาร์ดแวร์ที่เป็นส่วนติดต่อกับอินพุตและเอาต์พุต
5. ทำการเขียนแอพพลิเคชันในยูสเซอร์โหมด (User mode) ซึ่งสามารถจะใช้ฮาร์ดแวร์ผ่าน ทางดีไวซ์ไดรเวอร์ที่เขียนอยู่ในเคอร์เนลโหมดได้
6. ทำขั้นตอนที่ 1-4 ใหม่ สำหรับเมื่อมีการเปลี่ยนเป็นโอเอสตัวใหม่ ซึ่งควรที่จะสามารถรันโค้ดในข้อ 5 ได้

4.2 โปรแกรมมอดูล

แบ่งเป็น 2 ส่วนหลัก ๆ คือ

1. module_init(*function*) : จะถูกเรียกใช้งานเมื่อมีการติดตั้งมอดูลเข้าไปในเคอร์เนล ใช้ในการติดตั้งค่าหรือเตรียมพร้อมการทำงานของมอดูลนั้น โดยฟังก์ชันนี้จะไปเรียก *function* มาทำงาน โดยเราจะต้องไปเขียนฟังก์ชันที่ใช้ในตอนเริ่มต้นโปรแกรม อย่างเช่น

```
static int hello_init(void)
{
        printk(KERN_ALERT “Hello, world\n”);
        return 0;
}
```

แล้วก็มีการประกาศ module_init(hello_init); ไว้ด้านล่าง

การเรียกใช้งาน : root# insmod *./hello.ko* (สมมุติว่าหลังจากคอมไพล์แล้วได้ hello.ko)

2. module_exit(*function*) : จะถูกเรียกใช้งานเมื่อต้องการกำจัดมอดูลออกจากเคอร์เนลของระบบ ใช้ในการจัดการกำจัดสิ่งที่เราได้สร้างขึ้นมาจากโปรแกรมของเราให้ระบบมีสภาพเหมือนก่อนติดตั้งมอดูล โดยฟังก์ชันนี้จะไปเรียก *function* มาทำงาน โดยเราจะต้องไปเขียนฟังก์ชันที่ใช้ในตอนกำจัด อย่างเช่น

```
static void hello_exit(void)
{
        printk(KERN_ALERT “Goodbye, cruel world\n”);
}
```

แล้วก็มีการประกาศ module_exit(hello_exit); ไว้ด้านล่าง

การเรียกใช้งาน : root# rmmod *hello* (สมมุติว่าเรารีจีสเตอร์มอดูลชื่อว่า hello)

ที่กล่าวมาในการเขียนโปรแกรมดีไวซ์ไดรเวอร์เราต้องมีการประกาศอินคลูด (include) เฮดเดอร์ไฟล์ที่จะเป็นด้วยซึ่งไฟล์ที่จำเป็นต้องอินคลูดมีดังนี้

```
#include <linux/init.h>
#include <linux/module.h>
```

4.3 การคอมไพล์

จะใช้เมกไฟล์ (Makefile) โดยการเขียนเมกไฟล์เบื้องต้นมีส่วนที่สำคัญดังนี้

obj-m := *hello.o*

เป็นการบอกว่ามีหนึ่งมอดูลที่จะสร้าง โดยสร้างจากออปเจ็กไฟล์ (object file) ชื่อ hello.o ผลลัพธ์ที่ได้จะเป็น hello.ko หลังจากที่บิลต์ (built) จากออปเจ็กไฟล์

แต่ถ้าเราต้องการมอดูลจากหลายไฟล์ เราสามารถทำได้ดังตัวอย่างด้านล่างนี้

obj-m := module.o

```
module-objs := file1.o file2.o
```

เป็นการบอกว่าสร้างมอดูลจากไฟล์ module.ko นั้นสร้างจากออปเจ็กไฟล์ file1.o และ file2.o

เมกไฟล์ด้านบนที่กล่าวมายังไม่สามารถใช้งานได้ เพราะยังขาดในส่วนของตำแหน่งของซอร์สโค้ดของเคอร์เนล ซึ่งจำเป็นต้องใช้ในการคอมไพล์ด้วยเช่นถ้าเคอร์เนลซอร์สของคุณอยู่ที่ตำแหน่ง ~/kernel-2.6 เช่นตัวอย่างนี้

```
make –C ~/kernel-2.6 M=`pwd` modules
```

คำสั่งจะเริ่มจากการเปลี่ยนไดเรกทอรีไปเป็นไดเรกทอรีที่ระบุไว้ด้วยออปชัน -C (ซึ่งระบุตำแหน่งของเคอร์เนลซอร์ส) ออปชัน M= ทำให้เมกไฟล์ย้ายกลับมาที่ซอร์สไดเรกทอรีของมอดูลก่อนที่จะบิลด์ modules ซึ่ง modules นั้นถูกระบุไว้ในตัวแปร obj-m หลังจากนั้นเราก็ใช้คำสั่ง make ที่คอมมานไลน์ โปรแกรม make จะไปอ่านเมกไฟล์แล้วทำงานเอง

4.4 คาแร็กเตอร์ดีไวซ์ไดรเวอร์

การเขียนโปรแกรมไดรเวอร์ของคาแร็กเตอร์ดีไวซ์ เป็นแบบที่ง่ายที่สุด เนื่องจากไม่มีการเกี่ยวข้องกับบัฟเฟอร์ โดยไดรเวอร์จะทำงานในลักษณะของมอดูล คือ สามารถโหลดได้เมื่อเคอร์เนลต้องการใช้งาน ซึ่งทำงานเหมือนเป็นอุปกรณ์จริง

เคอร์เนลจะใช้เลขเมเจอร์เพื่อกำหนดไดรเวอร์ที่จะใช้กับอุปกรณ์ และดีไวซ์ไดรเวอร์จะใช้เลขไมเนอร์ในการแยกแยะอุปกรณ์แต่ละชิ้น ซึ่งการจะเข้าถึงเลขเมเจอร์และเลขไมเนอร์ได้นั้น จะใช้ตัวแปรที่สร้างจาก dev_t (ประกาศไว้ใน linux/types.h) วิธีการเซตค่าเมเจอร์และไมเนอร์นั้นกระทำได้ผ่านมาโครที่ประกาศไว้ในไฟล์ <linux/kdev_t.h> โดยสามารถแบ่งได้เป็น

```
MAJOR(dev_t dev);
MINOR(dev_t dev);
ถ้าคุณมีเลขเมเจอร์และไมเนอร์แล้วต้องการใส่ลงใน dev_t ให้ใช้
MKDEV(int major, int minor);
```

4.5 การจองและคืนเลขดีไวซ์

สิ่งแรกที่ไดรเวอร์จำเป็นต้องทำในการติดตั้งคาแร็กเตอร์ดีไวซ์คือให้ได้รับจำนวนดีไวซ์หนึ่งหรือมากกว่าที่จะทำงาน มีฟังก์ชันที่สำคัญคือ register_chrdev_region ซึ่งได้ประกาศไว้ใน <linux/fs.h>

```
int register_chrdev_region(dev_t first, unsigned int count, char *name);
```

โดยที่ first เลขเริ่มต้นที่อยู่ในช่วงที่คุณต้องการจอง โดยมากจะเป็นเลข 0 count คือจำนวนของเลขของดีไวซ์ที่ต่อเนื่องกันทั้งหมด สุดท้าย name เป็นชื่อของดีไวซ์ที่จะไปปรากฏใน /proc/devices และ sysfs

บ่อยครั้งที่ไม่รู้ว่าเลขเมเจอร์ที่ดีไวซ์จะใช้ ก็สามารถใช้การวิธีการจอง ณ เวลาที่ติดตั้งนั้นเลย (dynamicly-allocated) ได้ โดยเคอร์เนลจะจองเลขเมเจอร์ให้เลย ณ ตอนที่ติดตั้ง แต่คุณต้องใช้ฟังก์ชัน

```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
                        unsigned int count, char *name);
```

ฟังก์ชันนี้ dev จะเป็นเอาท์พุทเดียวที่ได้รับถ้า ฟังก์ชันนั้นทำสำเร็จ firstminor จะเป็นเลขไมเนอร์ตัวแรกที่จะใช้ มักจะเป็น 0count และ nameนั้นเหมือนกับฟังก์ชัน register_chrdev_region

ส่วนตอนการคืนเลขดีไวซ์ใช้ฟังก์ชัน

```
void unregister_chrdev_region(dev_t first, unsigned int count);
```

ซึ่งจะถูกเรียกใช้งานในฟังก์ชันที่กำจัดมอดูล

4.6 ฟังก์ชันที่ใช้ทำงานกับอุปกรณ์ (ไฟล์)

เคอร์เนลจะทำงานกับอุปกรณ์ผ่านรูทีนของไดรเวอร์ที่เก็บไว้ในโครงสร้างข้อมูล file_operations โดยในกรณีที่ฟังก์ชันใดไม่ได้ทำงานกับอุปกรณ์ จะไม่มีการใส่ค่าในพอยเตอร์นั่น โดยโครงสร้างของ file_operations มีดังนี้

```
struct file_operations fops = {
.owner  =   THIS_MODULE,
.llseek =   llseek,
.read   =   read,
.write  =   write,
.ioctl  =   ioctl,
.open   =   open,
.release =  release,
};
```

ในโครงสร้างของ file_operations นั้นจะประกอบด้วยพอยเตอร์ไปยังฟังก์ชันต่าง ๆ เช่น .read ก็จะชี้ไปยังฟังก์ชันที่อยู่หลังเครื่องหมายเท่ากับในที่นี่ก็คือฟังก์ชัน read

สรุปการทำงานของฟังก์ชัน ดังต่อไปนี้

struct module *owner

เป็นฟิลด์แรกใน file_operations ซึ่ไม่ได้เป็นโอเปอร์เรชันทั้งหมด แต่เป็นพอยเตอร์ที่บอกว่าถึงเจ้าของมอดูลนี้ ฟิลด์นี้จะช่วยป้องกันมอดูลจากการถูกเอาออก (unload) ขณะที่โอเปอร์เรชันยังคงใช้งานอยู่ โดยปกติแล้วจะถูกเซตไว้เป็น THIS_MODULE มาโครนี้จะถูกกำหนดไว้ใน <linux/module.h>

loff_t (*llseek) (struct file *, loff_t, int);

ใช้เปลี่ยนตำแหน่งในการอ่านหรือเขียนข้อมูลลงไฟล์ โดยค่าที่ส่งกลับจะเป็นจำนวนเต็มบวก หรือจุดออฟเซต (Offset) จากจุดเริ่มต้นไฟล์ ถ้าไม่มีการระบุฟังก์ชันนี้ จะทำให้ไดรเวอร์ไม่สามารถกำหนดตำแหน่งบนข้อมูลของอุปกรณ์ได้

ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);

ใช้แอ็กเซสข้อมูลจากอุปกรณ์ ถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL จะทำให้ฟังก์ชันของระบบ read ไม่สามารถทำงานได้ จะเกิด ERROR –EINVAL โดยค่าที่ส่งกลับจะเป็นจำนวนไบต์ที่อ่านได้ เมื่อทำงานสำเร็จ

ssize_t (*write) (struct file *,const char __user *, size_t, loff_t *);

ฟังก์ชันจะส่งข้อมูลไปยังอุปกรณ์ โดยค่าที่จะส่งกลับจะเป็นจำนวนไบต์ที่เขียนได้

int (*readdir) (struct file *, void *, filldir_t);

ฟังก์ชันนี้ ควรจะกำหนดให้เป็น NULL สำหรับไฟล์ node ของอุปกรณ์ เนื่องจากปกติจะใช้อ่านข้อมูลในไดเรกทรอรี (Directory)

int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);

ใช้สำหรับการออกคำสั่งเฉพาะของอุปกรณ์ เช่น การฟอร์แมต (Format) ดิสก์ โดยไม่มีการอ้างอิงไปยังตาราง fops โดยค่าที่ส่งกลับจะเป็นจำนวนเต็มบวก

int (*mmap) (struct file *, struct vm_area_struct *);

ใช้แม็พ (Map) พื้นที่หน่วยความจำของอุปกรณ์และหน่วยความจำของโปรเซส

int (*open) (struct inode *, struct file *);

ใช้เมื่อต้องการจะเริ่มการทำงานกับอุปกรณ์ โดยถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL จะทำให้การเรียกใช้งานฟังก์ชันสำเร็จเสมอ แต่ไม่มีค่าส่งกลับ

int (*release) (struct inode *, struct file *);

ใช้งานเมื่อไฟล์ดีไวซ์ไดรเวอร์ (/dev/…) ของอุปกรณ์ถูกปิด

int (*fsync) (struct file *, struct dentry *, int);

ใช้รีเซต (Reset) อุปกรณ์ให้อยู่ในสถานะพร้อมทำงาน

บทที่ 5
การเขียนพีซีไอไดรเวอร์สำหรับลีนุกซ์

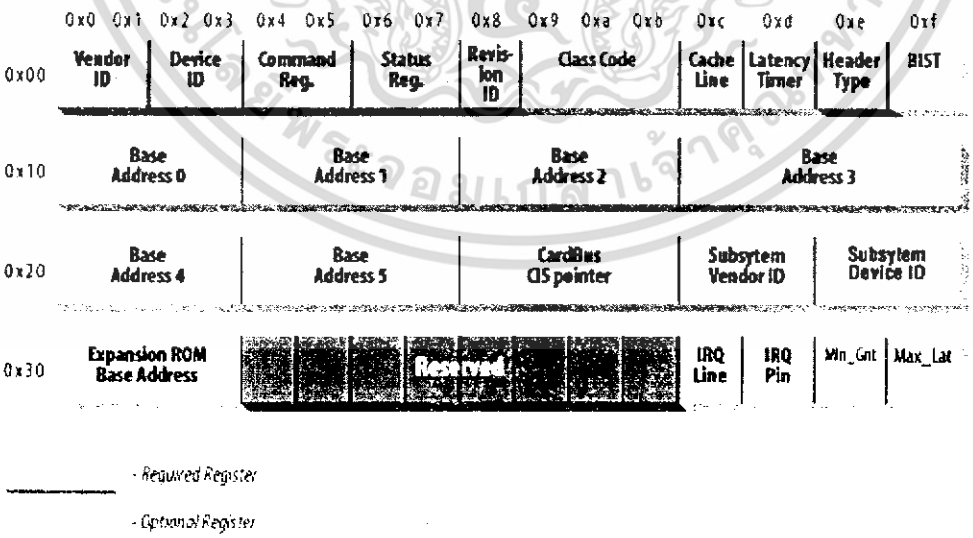
5.1 พีซีไออินเตอร์เฟส

สถาปัตยกรรมพีซีไอนั้นได้สร้างมาเพื่อแทนที่สถาปัตยกรรมแบบไอเอสเอ มีจุดหมายหลักอยู่สามประการคือ 1. เพื่อเพิ่มประสิทธิภาพในการส่งข้อมูลระหว่างคอมพิวเตอร์กับอุปกรณ์ต่อพ่วง (peripheral) 2. เพื่อให้ไม่ขึ้นกับแพลตฟอร์ม (platform) ให้มากที่สุดที่จะเป็นไปได้ และ3. เพื่อให้ง่ายในการใส่และถอดอุปกรณ์ต่าง ๆ ออกจากระบบ

พีซีไอบัสนั้นมีประสิทธิภาพสูงกว่าไอเอสเอเพราะใช้สัญญาณนาฬิกาที่มีความถี่สูงขึ้นกว่าสัญญาณนาฬิกานั้นจะทำงานอยู่ที่ความถี่ 25 หรือ 30 MHz (ขึ้นอยู่กับสัญญาณนาฬิกาของระบบด้วย) มีการส่งข้อมูลที่เป็นแบบ 32 บิต เรื่องของการไม่ขึ้นกับแพลตฟอร์มนั้นเป็นจุดมุ่งหมายสำคัญในการออกแบบคอมพิวเตอร์บัสและยังเป็นส่วนสำคัญที่เป็นคุณสมบัติของพีซีไอ เพราะในโลกของพีซีไอสามารถนำพีซีไอไปใช้ในเครื่องที่เป็น IA-32, Alpha, PowerPC, SPARC64 และบางระบบอื่นได้เป็นอย่างดี

5.2 คอนฟิกกูเรชั่นรีจิสเตอร์ (Configuration registers)

รูปด้านล่างนี้เป็นรูปที่แสดงตำแหน่งของรีจิสเตอร์ต่าง ๆ ของพีซีไอดีไวซ์ ทุกพีซีไอดีไวซ์ต้องมีพื้นที่อย่างน้อย 256 ไบต์ โดย 64 ไบต์แรกจะเป็นมาตรฐานที่เหลือก็แล้วแต่ดีไวซ์แต่ละตัว



รูปที่ 5.1 มาตรฐานของ PCI Configuration registers

จากรูปจะเห็นว่ารีจิสเตอร์บางตัวนั้นจำเป็นต้องมี (required register) และบางตัวก็เป็นเพียงส่วนที่เสริมเข้ามา (optional register) ทุกพีซีไอดีไวซ์จะต้องเก็บค่าที่มีความหมายชัดเจนในส่วนของรีจิสเตอร์ที่ต้องมี ส่วนเนื้อหาของรีจิสเตอร์ที่เป็นส่วนเพิ่มเติมนั้นขึ้นอยู่กับดีไวซ์นั้น ๆ

สิ่งที่ต้องระลึกถึงไว้เสมอคือ พีซีไอรีจิสเตอร์นั้นจะเป็นลิตเติ้ลเอ็นเดียน (little-endian) เสมอ ดังนั้นเรื่องของการเรียงลำดับของข้อมูลนั้นเป็นปัญหาที่ต้องระวัง

พีซีไอรีจิสเตอร์ที่ใช้ในการระบุดีไวซ์แต่ละตัวมีอยู่สามตัวจากห้าตัว คือ เว็นเดอร์ไอดี (vendorID) ดีไวซ์ไอดี (deviceID) และคลาส (class) ผู้ผลิตพีซีไอดีไวซ์จะกำหนดค่าที่เหมาะสมให้กับรีจิสเตอร์เหล่านี้โดยทำให้เป็นแบบอ่านได้อย่างเดียว ซึ่งผู้ที่เขียนไดรเวอร์สามารถใช้ข้อมูลตรงนี้ในการค้นหาดีไวซ์ได้ และในส่วนของ ซับซิสเต็มเว็นเดอร์ไอดี (subsystem vendorID) และซับซิสเต็มดีไวซ์ไอดี (subsystem deviceID) อาจจะถูกเซตเอาไว้ในบางครั้งโดยผู้ผลิตเพื่อใช้ในการแบ่งแยกความแตกต่างระหว่างดีไวซ์ที่เหมือนกัน

vendorID

เป็นรีจิสเตอร์ขนาด 16 บิต เป็นตัวบ่งบอกผู้ผลิตดีไวซ์นั้น ตัวอย่างเช่น ดีไวซ์ของ intel จะมีเลขเว็นเดอร์ไอดีเป็น 0x8086 เป็นต้น

deviceID

เป็นรีจิสเตอร์ขนาด 16 บิต จะถูกเลือกโดยโรงงานที่ผลิต ไอดีตัวนี้จะใช้คู่กันกับเว็นเดอร์ไอดี กลายเป็นเลขที่ไม่ซ้ำขนาด 32 บิต ใช้ในการแยกแยะฮาร์ดแวร์แต่ละตัว มักจะใช้คำว่าลายมือชื่อ (signature) แทนคู่ของเว็นเดอร์ไอดีและดีไวซ์ไอดี

class

ทุก ๆ ดีไวซ์จะมีคลาสอยู่ คลาสรีจิสเตอร์มีขนาด 16 บิต แบ่งออกเป็น 8 บิตบนเป็นเบสคลาส (base class) หรือกลุ่ม ตัวอย่างเช่น อีเทอร์เนต (ethernet) และ โทเค็นริง (token ring) ทั้งสองอย่างจะอยู่ในกลุ่มของเน็ตเวิร์ค (network) ขณะที่อนุกรม (serial) และขนาน (parallel) จะอยู่ในกลุ่มของการติดต่อสื่อสาร (communication) บางดีไวซ์ยังสามารถสนับสนุนหลาย ๆ อย่างบนดีไวซ์เดียว

subsystem vendorID, subsystem deviceID

ส่วนนี้สามารถใช้ในการแยกแยะดีไวซ์ที่เป็นฮาร์ดแวร์ตัวเดียวกัน คือถ้าชิป (chip) ที่ใช้นั้นทำได้หลายอย่างบนโลคอลบัส แต่ทำงานต่างกันอย่างสิ้นเชิง และไดรเวอร์จะต้องแยกแยะแต่ละดีไวซ์ที่ทำงานด้วยได้

5.3 โครงสร้างของพีซีไอไดรเวอร์

พีซีไอไดรเวอร์นั้นมีอยู่สองแบบคือ แบบเก่าและแบบใหม่ โดยแบบใหม่จะยกเว้นเรื่องของการตรวจสอบดีไวซ์ (probing) และสนับสนุนการทำงานแบบฮอตปลั๊ก (hot plug) คือการที่สามารถ

ถอดและใส่อุปกรณ์ในขณะที่เครื่องยังเปิดอยู่ ส่วนแบบเก่านั้นการตรวจสอบดีไวซ์ผู้เขียนจะต้องทำเอง ดังนั้นถ้าเราไม่มีเหตุผลในการใช้งานการตรวจสอบแบบเก่าก็ไม่ควรใช้ หลังจากที่ไดรเวอร์นั้นพบดีไวซ์แล้วจำเป็นต้องทำตามขั้นตอนดังต่อไปนี้

เปิดการใช้งานดีไวซ์ (enable the device)

เข้าใช้งานพื้นที่ที่กำหนดไว้ในดีไวซ์ (access device configuration space)

ค้นหาทรัพยากรต่าง ๆ ที่ดีไวซ์จัดเตรียมไว้ให้เช่น หมายเลขไออาร์คิว (IRQ numbers) และแอดเดรส เป็นต้น

จัดเตรียมทรัพยากรเหล่านั้น (allocate these resource)

ติดต่อกับดีไวซ์ (communicate with the device)

5.4 ไดรเวอร์แบบใหม่ (New-style)

ไดรเวอร์แบบใหม่นั้นมักจะเรียกฟังก์ชัน pci_register_driver ในตอนที่เริ่มต้นด้วยตัวชี้ (pointer) ไปยังสตรักเจอร์ (structure) ที่อธิบายไดรเวอร์ (struct pci_driver) โดยจะมีข้อมูลดังนี้

name ชื่อของไดรเวอร์

id_table ตัวชี้ไปยังตารางของดีไวซ์ไอดี

probe ตัวชี้ไปยังฟังก์ชันที่ใช้ในการตรวจสอบดีไวซ์ (probing function) จะถูกเรียกสำหรับทุกดีไวซ์ที่ตรงกับตารางไอดียังไม่ถูกดูแลโดยไดรเวอร์ใด ๆ ฟังก์ชันนี้จะส่งผ่านตัวชี้ที่ชี้ไปยัง pci_dev ที่เป็นสตรักเจอร์ที่ใช้แทนดีไวซ์และข้อมูลของตัวที่ตรงกันกับในตารางไอดี ถ้าไม่มีข้อผิดพลาดใด ๆ จะส่งค่ากลับเป็น 0 แต่ถ้าเป็นอย่างอื่นจะส่งค่ากลับเป็นหมายเลขที่ตรงกับข้อผิดพลาด (error code) ฟังก์ชันนี้จะถูกเรียกจากโพสเซสคอนเท็กซ์ (process context) เสมอ ดังนั้นจึงสามารถหลับ (sleep) ได้

remove ตัวชี้ไปยังฟังก์ชันที่ใช้ในการกำจัดค่าต่าง ๆ ซึ่งจะถูกเรียกในตอนที่ไดรเวอร์นั้นถูกถอดถอน (remove) ฟังก์ชันนี้จะถูกเรียกจากโพสเซสคอนเท็กซ์เสมอ ดังนั้นจึงสามารถหลับได้

save_state บันทึกสถานะของดีไวซ์ก่อนที่จะถูกพัก (suspend)

suspend ทำให้ดีไวซ์อยู่สถานะพลังงานน้อย (low power state)

resume ทำให้ดีไวซ์กลับมาจากสถานะพลังงานน้อย

enable_wake เปิดให้ดีไวซ์สร้างสถานการณ์ปลุก (wake event) จาก สถานะพลังงานน้อย

ตารางไอดี นั้นเป็นอาร์เรย์ของ struct pci_device_id ที่จบด้วยศูนย์ทั้งหมด แต่ละส่วนประกอบด้วย

vendor. เว็นเดอร์และดีไวซ์ไอดีที่ตรงกัน (หรือใช้ PCI_ANY_ID)

device

subvendor, ซับซิสเต็มเว็นเดอร์และดีไวซ์ไอดีที่ตรงกัน (หรือใช้ PCI_ANY_ID)

subdevice

class, คลาสของดีไวซ์ที่ตรงกัน

class_mask คลาสมาส์ค เป็นตัวบอกว่าใจใช้บิตไหนในการเปรียบเทียบ

driver_data ข้อมูลที่เป็นของไดรเวอร์นั้นเท่านั้น

ไดรเวอร์ส่วนมากจะไม่ต้องการใช้ส่วนของ driver_data

กรุณาใส่เครื่องหมายเหล่านี้ลงไปในส่วนของฟังก์ชันที่ใช้ตอนเริ่มต้นและส่วนของฟังก์ชันที่ใช้ตอนลบที่ที่ควรจะว่างนั้นมีหลักเกณฑ์ในการพิจารณาดังนี้

__init ไว้ในโค้ดที่จะถูกเรียกตอนเริ่มต้น และจะไม่ถูกใช้งานอีกหลังจากไดรเวอร์ผ่านส่วนของการเริ่มต้นไปแล้ว

__exit ไว้ในโค้ดที่จะถูกเรียกตอนออก ใช้ไม่ได้กับไดรเวอร์ที่ไม่ได้เป็นแบบโมดูล

__devinit ไว้ในโค้ดที่จะถูกเรียกตอนเริ่มต้นดีไวซ์ เหมือนกับ __init ถ้าเคอร์เนลนั้นไม่ได้คอมไพล์ด้วย CONFIG_HOTPLUG

__devexit เหมือนกับ __exit

5.5 การค้นหาพีซีไอดีไวซ์ด้วยตนเอง (แบบเก่า)

พีซีไอไดรเวอร์ที่ไม่ได้ใช้ฟังก์ชัน pci_register_driver() ในการค้นหาพีซีไอดีไวซ์ จะค้นหาโดยการใช้ขั้นตอนดังต่อไปนี้

การค้นหาโดยใช้เว็นเดอร์และดีไวซ์ไอดี

```
struct pci_dev *dev = NULL;
while (dev = pci_find_device(VENDOR_ID, DEVICE_ID, dev))
        configure_device(dev);
```

การค้นหาโดยใช้คลาสไอดี

```
pci_find_class(CLASS_ID, dev)
```

การค้นหาโดยใช้ทั้งเว็นเดอร์/ดีไวซ์ไอดี และซับซิสเต็ม เว็นเดอร์/ดีไวซ์ ไอดี

```
pci_find_subsys(VENDOR_ID, DEVICE_ID, SUBSYS_VENDOR_ID, SUBSYS_DEVICE_ID, dev)
```

ผู้เขียนดีไวซ์ไดรเวอร์สามารถใช้ PCI_ANY_ID แทนในส่วนของ VENDOR_ID และ DEVICE_ID ได้ ซึ่งจะหมายถึงเป็นไอดีทุกตัว ใช้สำหรับการค้นหาเฉพาะ VENDOR_ID หรือ DEVICE_ID นั้น ๆ

5.6 เปิดการใช้งานดีไวซ์ (Enable the device)

ก่อนที่จะเริ่มดำเนินการใด ๆ กับดีไวซ์ที่ค้นพบ จะมีการเรียกใช้งานฟังก์ชัน pci_enable_device() เสียก่อน เพื่อเป็นการเปิดการใช้งานพื้นที่ไอโอ (I/O region) และพื้นที่หน่วยความจำ (memory region) ของดีไวซ์ จองไออาร์คิวถ้าจำเป็น ปลุกดีไวซ์ถ้าดีไวซ์นั้นอยู่ในสถานะพัก

5.7 การเข้าถึงพื้นที่พีซีไอคอนฟิกสเปซ (PCI config space)

ผู้เขียนไดรเวอร์สามารถใช้ฟังก์ชัน pci_(read|write)_config_(byte|word|dword) ในการเข้าถึงคอนฟิกสเปซที่ดีไวซ์นั้นจัดเตรียมเอาไว้ผ่านทาง struct pci_dev * ทุกฟังก์ชันจะส่งค่ากลับเป็น 0 เมื่อทำงานสำเร็จหรือเป็นหมายเลขข้อผิดพลาด ที่สามารถแปลงเป็นข้อความได้โดยใช้ pcibios_strerror ไดรเวอร์ส่วนมากจะคิดว่าการเข้าถึงนั้นจะไม่ผิดพลาดบนดีไวซ์ที่ถูกต้อง

ถ้าไม่ใช้ pci_dev ก็สามารถใช้ผ่านทางฟังก์ชัน pci_bus_(read|write)_config_(byte|word|dword)

5.8 แอดเดรสและอินเตอร์รัพต์

แอดเดรสหน่วยความจำและพอร์ต และหมายเลขอินเตอร์รัพต์ ไม่ควรอ่านจากคอนฟิกสเปซ แต่ควรใช้ค่าที่อยู่ในสตรักเจอร์ pci_dev ซึ่งถูกเคอร์เนลทำการรีแมป (remap) ให้

ผู้เขียนไดรเวอร์ยังคงต้องใช้ request_region() สำหรับพื้นที่ไอโอ (I/O region) และ request_mem_region() สำหรับพื้นที่หน่วยความจำ เพื่อให้แน่ใจว่าไม่มีใครกำลังใช้งานดีไวซ์ตัวเดียวกัน

ทุกส่วนที่ดูแลอินเตอร์รัพต์ควรจะเป็นแบบ SA_SHIRQ และใช้ devid ในการแมปไออาร์คิวไปยังดีไวซ์

บทที่ 6
การออกแบบ

6.1 แนวความคิดในการออกแบบ

การออกแบบชุดต้นแบบการเข้ารหัสด้วยฮาร์ดแวร์ นั้นจะเริ่มด้วยการมองภาพรวมของการทำงานทั้งหมดของระบบ หลังจากนั้นจะทำการศึกษาทฤษฎีต่าง ๆ ที่เกี่ยวข้องจนเข้าใจความต้องการทั้งหมดของระบบแล้ว เราจึงเริ่มทำการออกแบบระบบรวมทั้งหมด โดยในการออกแบบนั้น จะทำการแบ่งระบบออกเป็นส่วนย่อย ๆ โดยในแต่ละส่วนจะถูกออกแบบให้มีส่วนติดต่อที่เหมือนกับส่วนติดต่อมาตรฐานให้มากที่สุด เพื่อความสามารถในการใช้งานร่วมกันได้กับระบบที่มีอยู่แล้วในปัจจุบัน และความสามารถในการนำเอาส่วนต่าง ๆ กลับมาใช้ได้อีก

6.2 ภาพรวมในการออกแบบ

ในการออกแบบ เราจะยึดเอาการติดต่อระหว่างส่วนต่าง ๆ เป็นหลัก ซึ่งทำให้เราแบ่งส่วนต่าง ๆ ออกมาเป็น 3 ส่วนใหญ่ ๆ คือ ส่วนของอุปกรณ์ฝั่งฮาร์ดแวร์ ซอฟต์แวร์ดีไวซ์ไดรเวอร์ (Software Device Driver) และซอฟต์แวร์ โดยฮาร์ดแวร์และซอฟต์แวร์ทั้ง 2 ส่วน จะทำงานร่วมกันโดยผ่านบริการของระบบปฏิบัติการ (OS Services) ส่วนของฮาร์ดแวร์จะทำการติดต่อกับพีซีไอ (PCI Bus) โดยมีการ์ดพีซีไอ (PCI Card) เป็นส่วนหน้าในการเชื่อมต่อของฝั่งฮาร์ดแวร์ โดยการทำงานส่วนใหญ่ในฝั่งนี้จะขึ้นอยู่ที่บอร์ดเอฟพีจีเอ (FPGA Board) เป็นหลัก ซึ่งการเข้ารหัสของข้อมูลจะถูกกระทำในส่วนของฝั่งฮาร์ดแวร์นี้ รวมถึงการควบคุมการติดต่อกับพีซีไอบัสด้วย ในรุ่นแรกนี้ เราจะยังไม่ได้ใช้ความสามารถทั้งหมดของบัส เนื่องด้วยข้อจำกัดของเวลา และขนาดพื้นที่บอร์ดเอฟพีจีเอของเนื่องจากการพัฒนาในส่วนของฮาร์ดแวร์จะต้องใช้เวลานานกว่าการพัฒนาในส่วนของซอฟต์แวร์

สำหรับส่วนของการเข้ารหัสที่อยู่ในฝั่งฮาร์ดแวร์นั้น ประกอบด้วยสามฟังก์ชันคือในส่วนของการทำการสุ่มเลขเทียมโดยใช้อัลกอริธึม LFSR, การทำแฮชอัลกอริธึมโดยใช้อัลกอริธึม SHA1, และการเข้าและถอดรหัสลับซึ่งเป็นแบบ DES และ 3DES

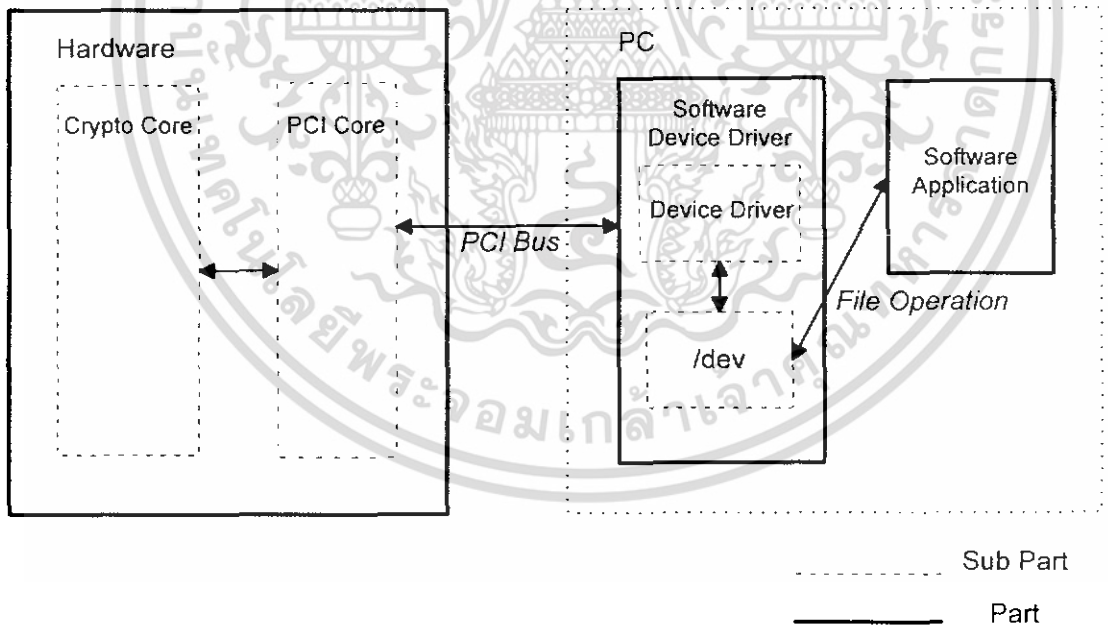
ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์ จะทำงานโดยฝังตัวเข้ากับบริการของระบบปฏิบัติการ โดยจะอาศัยบริการหลัก (กลาง) ของระบบปฏิบัติการทำงาน และสร้างบริการของระบบปฏิบัติการขึ้นมาใหม่ชุดหนึ่ง เพื่อใช้ในการเข้าถึงอุปกรณ์ฝั่งฮาร์ดแวร์ รวมถึงเพื่อให้บริการแก่โปรแกรมใช้งานทั่วไปให้สามารถเรียกใช้งานได้ด้วย สำหรับระบบปฏิบัติการที่เลือกมาใช้จะเป็น

ระบบปฏิบัติการ Linux ที่ Kernel รุ่น 2.6.0 มาตรฐาน สาเหตุที่เลือกนำระบบปฏิบัติการตัวนี้มาใช้เพราะเล็งเห็นถึงความง่ายในการพัฒนาและรวมถึงชุดพัฒนาที่สามารถหามาใช้ได้ง่าย

อีกส่วนหนึ่งของซอฟต์แวร์ คือ เราจะทำการพัฒนาโปรแกรมที่มาทดสอบการทำงานทั้งสามส่วนให้ถูกต้องตามมาตรฐาน สำหรับการทำงานระหว่างซอฟต์แวร์ทดสอบกับส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์นั้น ยังต้องติดต่อถึงกันและกันโดยผ่านส่วนติดต่อมาตรฐานของระบบปฏิบัติการอีกที รวมถึงในด้านสภาวะแวดล้อมของการใช้งาน โปรแกรมดีไวซ์ไดรเวอร์จะรันในเคอร์เนลโหมด ส่วนโปรแกรมจะรันในยูสเซอร์โหมด

6.3 การทำงานร่วมกันในแต่ละส่วน

ส่วนต่างๆของชุดต้นแบบนี้จะติดต่อซึ่งกันและกันตามรูปด้านล่าง ในระบบของชุดต้นแบบจะไม่ มีการติดต่อถึงกันโดยตรงระหว่างส่วนต่างๆ นอกเหนือจากที่แสดงไว้ในรูปข้างล่าง สำหรับส่วนฝั่งฮาร์ดแวร์จะทำการติดต่อกับระบบปฏิบัติการผ่านทางพีซีไอบัส(PCI Bus) และโปรแกรมจะเรียกใช้งานดีไวซ์ไดรเวอร์ผ่านทางไฟล์โอเปเรชัน(File Operation) โดยการรับส่งไฟล์ผ่านฟังชันอ่าน(Write) และเขียน(Read) ควบคุมการทำงานต่างผ่านฟังชั่นไอโอคอนโทร(I/O Control)



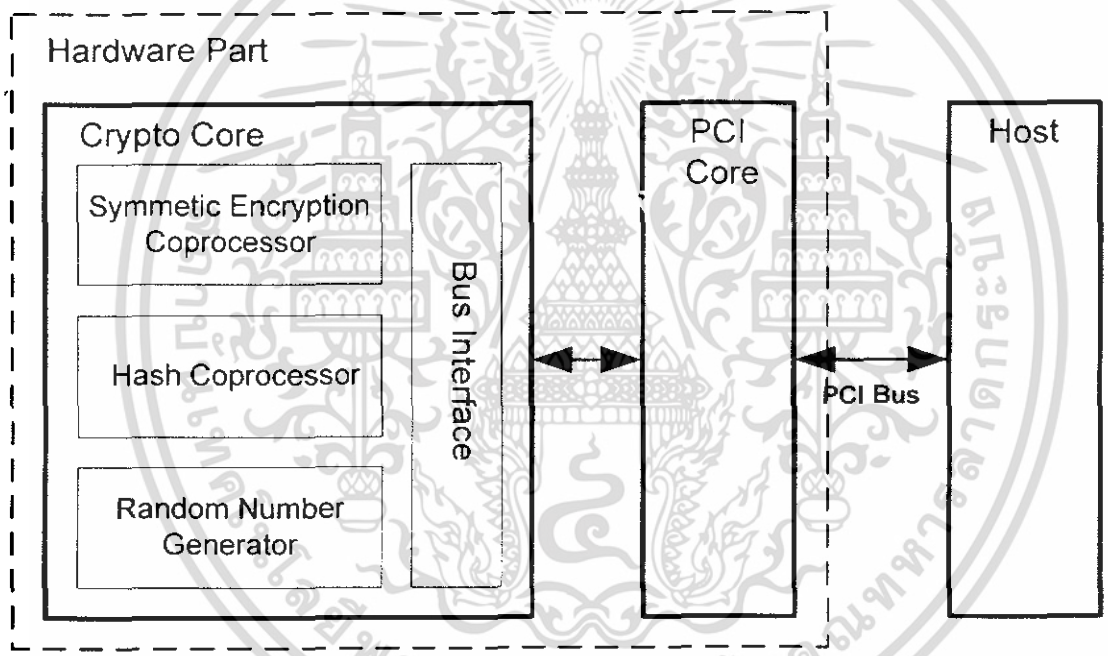
รูปที่ 6.1 แสดงการทำงานร่วมกันระหว่างแต่ละส่วนของระบบ

6.4 การทำงานในแต่ละส่วนย่อย

6.4.1 การออกแบบการทำงานส่วนของฮาร์ดแวร์

เนื่องจากการทำงานโดยใช้ซอฟต์แวร์นั้น ยังมีขีดจำกัดในเรื่องความเร็วในการส่งข้อมูล และความยืดหยุ่นในการนำไปประยุกต์ใช้งานในสภาพการทำงานปกติ ดังนั้นโครงงานนี้จึงได้ออกแบบการทำงานในส่วนที่ต้องมีการใช้การประมวลผลมากใช้ฮาร์ดแวร์เข้ามาช่วยแก้ปัญหาในส่วนนี้ และต้นแบบในโครงงานนี้ยังสามารถนำไปพัฒนาต่อให้ใช้งานได้เป็นอิสระโดยไม่ต้องพึ่งพาเครื่องคอมพิวเตอร์ส่วนบุคคลในการทำงานได้ด้วย

ในส่วนออกแบบอุปกรณ์ส่วนฮาร์ดแวร์เพื่อให้เป็นมาตรฐานและง่ายในการไปพัฒนาต่อจึงได้แบ่งออกเป็นสองส่วนคือ ส่วนการติดต่อกับพีซีไอบัส(PCI Core) และ ส่วนที่รวมการทำงานในด้านการเข้ารหัสถอดรหัสไว้ (Crypto Core) ตามรูปข้างล่าง



รูปที่ 6.2 ระบบย่อยส่วนต่างๆของฮาร์ดแวร์

และสำหรับส่วนชุดเข้ารหัส(Crypto Core) ภายในประกอบด้วยส่วนย่อยๆอีกคือ

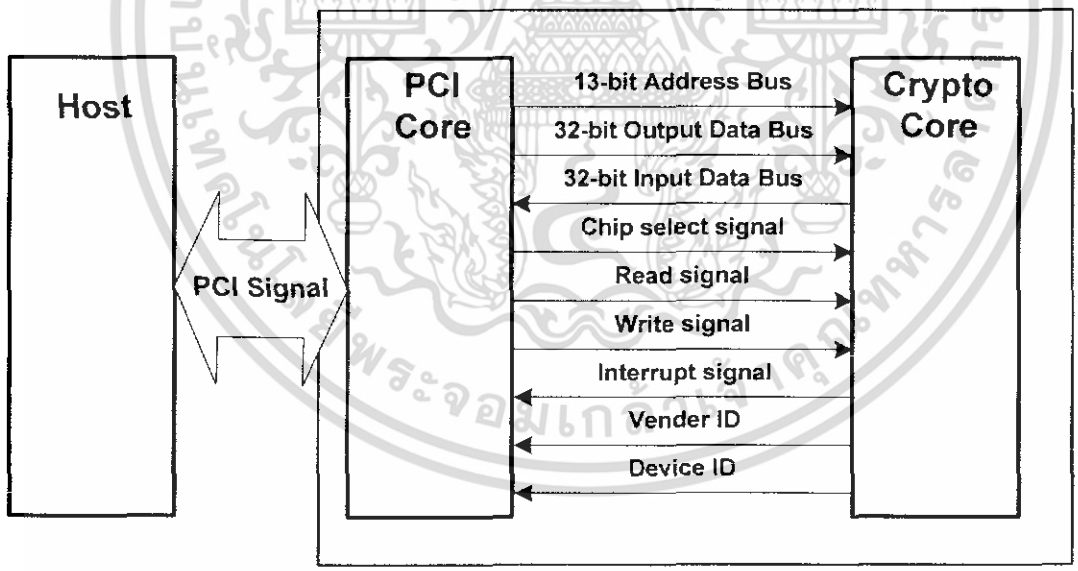
- การเข้าและถอดรหัสลับโดยอัลกอริธึม DES
- การแฮชอัลกอริธึมโดยใช้อัลกอริธึม SHA1
- การสุ่มเลขเทียมโดยใช้อัลกอริธึม LFSR128
- ส่วนติดต่อ Bus Interface
- ส่วนจัดการ Device ID/Vender ID

1. ชุดเชื่อมต่อพีซีไอ (PCI Core)

สำหรับชุดเชื่อมต่อพีซีไอ (PCI Core) ทำหน้าที่เชื่อมต่อการทำงานระหว่างพีซีไอ (PCI Bus) กับชุดเข้ารหัส (Crypto Core) โดยที่สัญญาณการติดต่อระหว่างชุดเชื่อมต่อพีซีไอ (PCI Core) และ ชุดเข้ารหัส (Crypto Core) เป็นการเชื่อมต่อแบบ Memory Interface ซึ่งใช้สัญญาณ Address Bus, Data Bus, Read, Write และ Chip Select เป็นหลักในการเชื่อมต่อโดยมีสัญญาณต่างๆที่เชื่อมต่อกับชุดเข้ารหัส (Crypto Core) ดังนี้

- Address Bus 13 บิต (lbaddr[14:2]): เป็นสัญญาณ address bus ของ PCI.
- Input Data Bus 32 บิต (lbdatain[31:0])
- Output Data Bus 32 บิต (lbdataout[31:0])
- สัญญาณ Read, ทำงานแบบ active low (lbrdb)
- สัญญาณ Write, ทำงานแบบ active low (lbwrb)
- สัญญาณ Chip Select, ทำงานแบบ active low (lbcsb)
- สัญญาณ Interrupt, ทำงานแบบ active high (lbint)
- สัญญาณ Vender ID
- สัญญาณ Device ID

ลักษณะการเชื่อมต่อดังรูปข้างล่างนี้



รูปที่ 6.3 สัญญาณการติดต่อระหว่างชุดเชื่อมต่อพีซีไอ (PCI Core) และ ชุดเข้ารหัส (Crypto Core)

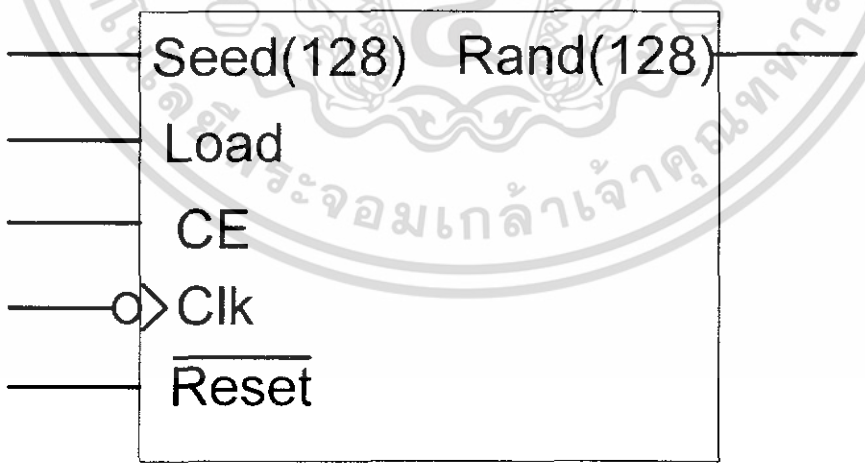
2. ชุดเข้ารหัส (Crypto Core)

สำหรับชุดเข้ารหัส (Crypto Core) ทำหน้าที่เชื่อมต่อการทำงานสัญญาณ Address Bus, Data Bus, Read, Write และ Chip Select ที่ได้จากชุดเชื่อมต่อพีซีไอ (PCI Core) แล้วส่งสัญญาณให้กับชุดเข้ารหัส (Crypto Core) ภายใน โดยที่ชุดเข้ารหัส (Crypto Core) จะทำหน้าที่จัดการสัญญาณต่างๆที่ส่งมาในรูปแบบ Memory Interface ให้เป็นสัญญาณสำหรับชุดเข้ารหัสต่างๆภายใน โดยจะทำการแปลงตำแหน่ง Address Bus ที่ได้รับแล้วตรวจสอบดูว่าสัญญาณ Read, Write, Chip Select เป็นอย่างไรถ้าตรงกับที่กำหนดไว้ก็ให้ส่งสัญญาณนั้นๆไปตามที่กำหนดไว้ ถ้าสัญญาณไหนต้องมีการคงค่าไว้ก็จะทำการ คงค่า (Latch) ไว้ด้วย แต่ถ้าตำแหน่ง Address ไม่ตรงตามกำหนดก็จะไม่สนใจสัญญารณนั้นๆ และสำหรับชุดเข้ารหัส (Crypto Core) แล้วภายในประกอบด้วยส่วนต่างๆ ดังนี้

- การเข้าและถอดรหัสลับโดยอัลกอริธึม DES
- การแฮชอัลกอริธึมโดยใช้อัลกอริธึม SHA1
- การสุ่มเลขเทียมโดยใช้อัลกอริธึม LFSR128
- ส่วนติดต่อ Bus Interface
- ส่วนจัดการ Device ID/Vender ID

3. Linear Feedback Shift Register (LFSR)

สำหรับ LFSR หรือ Linear Feedback Shift Register เป็นการสร้างเลขสุ่มเทียม (Pseudo Random Number Generator) ขนาด 128 bits ที่ใช้หลักการเลื่อนข้อมูลมาสร้างเลขสุ่ม สำหรับตัวสร้างเลขสุ่มเทียมตัวนี้ใช้เวลาเพียง 1 สัญญาณนาฬิกา (Clock) ในการสร้างเลขสุ่มเทียม



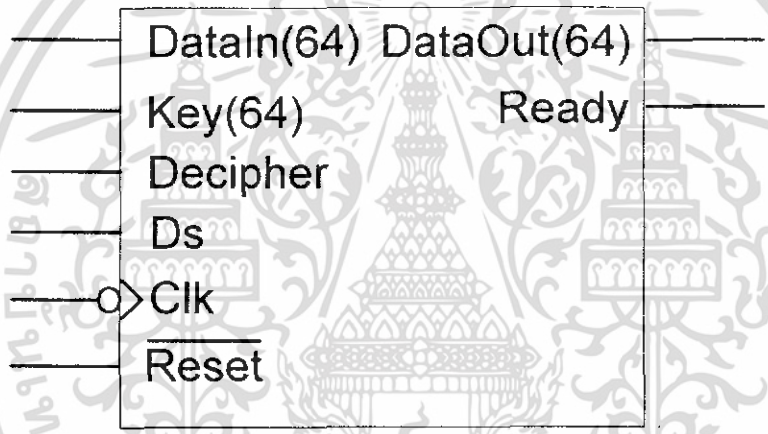
รูปที่ 6.4 สัญญาณต่างๆของ Linear Feedback Shift Register (LFSR)

ตารางที่ 6.1 ตารางแสดงสัญญาณต่างๆของ LFSR

pin	direction	description
Seed (32)	input	ขนาด 32 bits ใช้สำหรับป้อนค่าเริ่มต้นให้กับ LFSR
Load	input	'1' เพื่อทำการป้อนค่าเริ่มต้นให้กับ LFSR
CE	input	'1' เพื่อกำหนดให้ LFSR ทำการสร้างเลขสุ่มเลขใหม่ขึ้นมา
Reset	input	'0' เป็นการเริ่มต้นเพื่อกำหนดค่า หรือ เริ่มต้นการทำงานใหม่
Clk	input	สัญญาณนาฬิกาเพื่อให้มีการทำงานแบบ synchronous
Rand (32)	Output	สร้างเลขสุ่มขนาด 32 bits เมื่อมีการทำงานที่เรียบร้อย

4. Data Encryption Standard (DES)

สำหรับ DES หรือ Data Encryption Standard เป็นการเข้ารหัสและถอดรหัสลับแบบกุญแจที่เหมือนกัน โดยใช้เวลา 16 Clock ในการเข้าหรือถอดรหัสลับโดยกุญแจและข้อความขนาด 64 bits



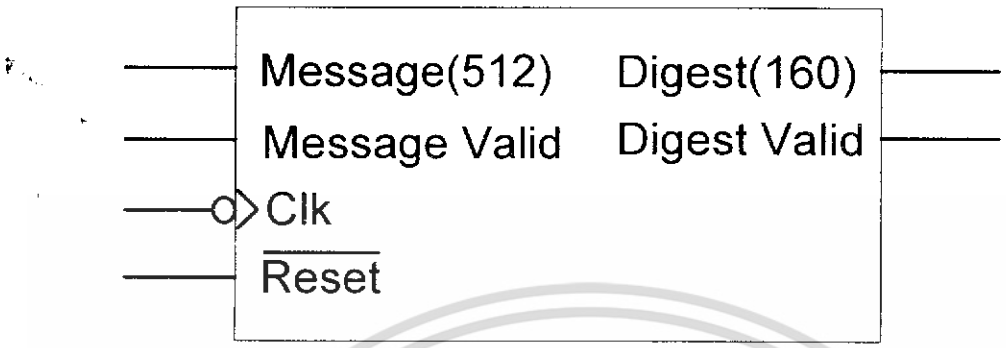
รูปที่ 6.5 แสดงสัญญาณ Data Encryption Standard (DES)

ตารางที่ 6.2 ตารางแสดงสัญญาณต่างๆของ DES

pin	direction	description
DataIn (64)	input	สัญญาณขนาด 64 bits ส่งเข้า เพื่อเข้ารหัสหรือถอดรหัส
Key (64)	input	ส่ง Key ขนาด 64 bits
Decipher	input	'0' เพื่อต้องการเข้ารหัส '1'เพื่อถอดรหัส
Ds	input	'1' เพื่อเริ่มการเข้าหรือถอดรหัส
Reset	input	'0' เป็นการเริ่มต้นเพื่อกำหนดค่า หรือ เริ่มต้นการทำงานใหม่
Clk	input	สัญญาณนาฬิกาเพื่อให้มีการทำงานแบบ synchronous
DataOut (64)	Output	เมื่อมีการเข้าหรือถอดรหัสแล้วจะได้ผลลัพธ์ขนาด 64
Ready	Output	เมื่อมีการเข้ารหัสหรือถอดรหัสเรียบร้อยจะมีสัญญาณ '1' นาน 1 Clock

5. Secure Hash Algorithm (SHA1)

SHA1 หรือ Secure Hash Algorithm เป็นการแปลงข้อความแบบทางเดียว โดยที่ข้อความในการแปลงแต่ละครั้งมีความยาวขนาด 512 bits และ ข้อความที่ออกมามีขนาด 160 bits สำหรับการแปลงข้อความ (Digest) ใช้เวลาทั้งหมด 80 Clock



รูปที่ 6.6 สัญญาณต่างๆของ Secure Hash Algorithm (SHA1)

ตารางที่ 6.3 ตารางแสดงสัญญาณต่างๆของ SHA1

pin	direction	description
Message (512)	input	ข้อความขนาด 512 bits ในการหา Digest
Message Valid	input	'1' เป็นการเริ่มการหา Digest
Reset	input	'0' เป็นการเริ่มต้นเพื่อกำหนดค่า หรือ เริ่มต้นการทำงานใหม่
Clk	input	สัญญาณนาฬิกาเพื่อให้มีการทำงานแบบ synchronous
Digest (160)	Output	Digest ขนาด 160 bits เมื่อกระบวนการหา Digest เรียบร้อย
Digest Valid	Output	เมื่อกระบวนการหา Digest เรียบร้อยแล้วสัญญาณนี้จะเป็น '1' นาน 1 clock

6. Device ID

สำหรับการติดต่อในระบบพีซีไอระบบปฏิบัติการจะทำการแยกแยะอุปกรณ์แต่ละตัวด้วย Device ID และ Vender ID ดังนั้นจึงมีส่วนนี้ทำหน้าที่ส่งค่า Device ID และ Vender ID ให้กับ Crypto Core เท่านั้น ตอนนี้ให้ค่าคงที่ไว้ดังนี้

- Vender ID -> F0F0
- Device ID -> F0F0

7. การนำไปใช้

ชุดเข้ารหัส(Crypto core) ของเราประกอบด้วย Linear Feedback Shift Register, Data Encryption Standard, Secure Hash Algorithm ทั้งสามส่วนมีขนาดมากเกินกว่าที่จะบรรจุลงบนการ์ดใบเดียว เราจึงใช้การ์ดพีซีไอเนกประสงค์สองใบสำหรับบรรจุชุดเข้ารหัส จึงได้แบ่งออกเป็นสองกลุ่มดังนี้

- กลุ่มที่ 1 - Secure Hash Algorithm
- กลุ่มที่ 2 - Linear Feedback Shift Register, Data Encryption Standard

ในการส่งสัญญาณต่างๆให้ชุดเข้ารหัสแล้วเราได้ออกแบบการทำงานเป็นแบบ Memory Interface ผ่านทางสัญญาณ Read, Write, Chip Select, Address, Data In, Data Out ในการส่งข้อมูลต่างๆต้องส่งตามตำแหน่งที่กำหนด

การ์ดใบที่ 1 (Secure Hash Algorithm)

ภายในบรรจุด้วย Secure Hash Algorithm มีตำแหน่ง Address และการเข้าถึงค่าต่างๆดังนี้

ตารางที่ 6.4 ตารางแสดงตำแหน่ง Address ของการ์ดใบที่ 1

Pin	Address	Direction
0004	Message 0	Input
0008	Message 1	Input
000c	Message 2	Input
0010	Message 3	Input
0014	Message 5	Input
0018	Message 6	Input
0020	Message 7	Input
0024	Message 8	Input
0028	Message 9	Input
002c	Message 10	Input
0030	Message 11	Input
0034	Message 12	Input
0038	Message 13	Input
003c	Message 14	Input
0040	Message 15	Input

ตารางที่ 6.4 (ต่อ)

0044	Message Valid	Input
0048	Reset	Input
0050	Digest E	Output
0054	Digest D	Output
0058	Digest C	Output
005c	Digest B	Output
0060	Digest A	Output
0064	Digest Valid	Output

การ์ดใบที่ 2 (Linear Feedback Shift Register, Data Encryption Standard)

ภายในบรรจุด้วย Linear Feedback Shift Register, Data Encryption Standard มีตำแหน่ง Address และการเข้าถึงค่าต่างๆดังนี้

Data Encryption Standard

ตารางที่ 6.5 ตารางแสดงตำแหน่ง Address ของการ์ดใบที่ 2 ในส่วนของ DES

Pin	Address	Direction
0004	Data In (Low)	Input
0008	Data In (High)	Input
000c	Key (Low)	Input
0010	Key (High)	Input
0014	Decipher	Input
0018	DS	Input
0020	Data Out (Low)	Output
0024	Data Out (High)	Output
0028	Ready	Output

Linear Feedback Shift Register

ตารางที่ 6.6 ตารางแสดงตำแหน่ง Address ของการ์ดใบที่ 2 ในส่วนของ LFSR

Pin	Address	Direction
0104	Seed (31 downto 0)	Input
0108	Seed (63 downto 32)	Input
010c	Seed (95 downto 64)	Input
0110	Seed (127 downto 96)	Input
0114	Load	Input
0118	CE	Input
0120	Rand (31 downto 0)	Output
0124	Rand (63 downto 32)	Output
0128	Rand (95 downto 64)	Output
012c	Rand (127 downto 96)	Output

6.4.2 การออกแบบการทำงานส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์

ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์นั้นถูกโปรแกรมออกมาเพื่อทำหน้าที่ 2 อย่าง คือ การควบคุมข้อ กำหนดในการใช้งานอุปกรณ์ และสร้างบริการสำหรับการใช้งานอุปกรณ์จากโปรแกรมผู้ใช้งานทั่วไป

แต่ซอฟต์แวร์ดีไวซ์ไดรเวอร์ยังแบ่งส่วนติดต่อได้เป็น 2 ฝั่ง คือ ส่วนติดต่อกับอุปกรณ์ และส่วนติดต่อ ผู้ใช้งาน ในส่วนติดต่อกับอุปกรณ์นั้น เราจะใช้บริการในระดับล่างของระบบปฏิบัติการเพื่อจองทรัพยากร และเพื่อรับส่งข้อมูลกับอุปกรณ์ฮาร์ดแวร์ เช่น การจองพอร์ตแอดเดรสของอุปกรณ์ และการส่งข้อมูลออกทางพอร์ต เป็นต้น สำหรับส่วนติดต่อผู้ใช้งานนั้นจะใช้บริการในระดับบนเพื่อเพิ่มบริการของตัวเองเข้ากับบริการที่มีอยู่ของระบบปฏิบัติการ การทำงานหลัก ๆ ของไดรเวอร์ คือ การควบคุมการรับส่งข้อมูลระหว่าง ส่วนติดต่อทั้ง 2 ฝั่ง

สำหรับส่วนติดต่อกับผู้ใช้งานนั้น เราจะเลือกใช้การให้บริการโดยมองอุปกรณ์เสมือนเป็นแฟ้มข้อมูลธรรมดา (Device File) ซึ่งในการใช้งานนั้น โปรแกรมฝั่งผู้ใช้งานสามารถเปิด ดีไวซ์ไฟล์ได้โดยผ่านทางการเรียกใช้ฟังก์ชัน (Function) open() และสามารถรับส่งข้อมูลเพื่อทำการเข้ารหัสได้โดยผ่าน ทางการเรียกใช้ฟังก์ชัน read() และ write() รวมถึงสามารถควบคุมการใช้งานอื่น ๆ ได้โดยผ่านทางการเรียกใช้ฟังก์ชัน ioctl() ซึ่งในตัวติดต่อกับผู้ใช้งานนี้นั้น จะให้บริการเฉพาะฟังก์ชันก์ที่จำเป็นสำหรับการใช้งานในเบื้องต้นนี้เท่านั้น โดยในซอฟต์แวร์ดีไวซ์ไดรเวอร์ของเรานั้นจะรองรับฟังก์ชันตามการประกาศโครงสร้างต่อไปนี้

```
struct file_operations cryptochip_fops = {
        .owner   =      THIS_MODULE,
        .read    =      cryptochip_read,
        .write   =      cryptochip_write,
        .ioctl   =      cryptochip_ioctl,
        .open    =      cryptochip_open,
        .release =      cryptochip_release,
};
```

การควบคุมการติดต่อสื่อสารนั้น เราจะต้องสามารถรับประกันความถูกต้องของข้อมูลได้ด้วย ซึ่งจะต้องรับประกันได้ว่า ถ้าโปรแกรมสามารถ write() หรือ read() ได้ โดยไม่ได้รับการแจ้งเตือน (return) ความผิดพลาด ดังนั้นข้อมูลที่ได้ทำการรับส่งหรือทำการประมวลผลกับดีไวซ์ไดรเวอร์นั้น ก็จะต้องเป็นข้อมูลที่มีความถูกต้องด้วย ซึ่งสิ่งสำคัญสำหรับการนี้คือ การป้องกันการเรียกโปรแกรมซ้อนทับ โดยเฉพาะอย่างยิ่ง การเขียนทับที่ตำแหน่งเดียวกัน ซึ่งในประเด็นนี้ เราจึงแก้ปัญหาโดยการใช้ซีมาฟอร์ (Semaphore) มาช่วยในการป้องกันการรันฟังก์ชันซ้อนทับกัน

ดีไวซ์ไดรเวอร์นั้นได้ถูกออกแบบให้ประกอบด้วยส่วนของเฮดเดอร์ไฟล์และโปรแกรมไฟล์ ซึ่งส่วนของเฮดเดอร์ไฟล์จะเก็บค่าต่าง ๆ ที่เกี่ยวกับการ์ดไว้ ทำให้ง่ายในการปรับแต่ง มีไฟล์ชื่อว่า cryptocard.c และ cryptocard.h

6.4.3 การออกแบบการทำงานส่วนของซอฟต์แวร์แอพพลิเคชัน

ทำการออกแบบโปรแกรมแอพพลิเคชันเพื่อมาทดสอบการทำงานทั้ง 3 ส่วน คือ ส่วนของการทำการสุ่มเลขเทียมว่าสามารถทำงานได้จริงถูกต้องและมีประสิทธิภาพ ส่วนของการทำแฮชอัลกอริธึม ส่วนของการทำการเข้ารหัสลับและถอดรหัสลับ เพื่อให้ง่ายต่อการใช้งานและเป็นมาตรฐานจึงทำการออกแบบการใช้งานทั้ง 3 ส่วนผ่านทางไฟล์โอเปอเรชันซึ่งเป็นบริการของระบบปฏิบัติการ ส่งข้อมูลผ่านทางฟังก์ชัน read, write, และ ควบคุมการทำงานโดยใช้ I/O control ดังนี้

1. LFSRขนาด 128 บิต ทำการส่งค่าตั้งต้น (seed) ผ่านฟังก์ชัน write และอ่านค่าเลขสุ่มเทียมผ่านฟังก์ชัน read

2. SHA1 ทำการส่งข้อความที่ต้องการผ่านทางฟังก์ชัน write และอ่านค่าเมสเซสไดเจสต์ที่ได้ออกมาผ่านฟังก์ชัน read

3. DES และ 3DES ทำการส่งข้อความที่ต้องการผ่านทางฟังก์ชัน write และอ่านค่าข้อความที่ทำการเข้ารหัสหรือถอดรหัสแล้วผ่านฟังก์ชัน read โดยเลือกโหมดและเลือกการเข้าหรือถอดรหัสผ่านทาง I/O Control

6.5 แนวคิดในการออกแบบชุดทดสอบ

สำหรับโครงงานนี้ เราได้ออกแบบให้มีลักษณะเป็นมอดูลาร์ (Modular) ให้มากที่สุด ดังนั้น แต่ละส่วนจะมีการทำงานเป็นอิสระ ซึ่งจะช่วยเป็นอย่างมากในเรื่องของการนำเอาส่วนต่าง ๆ กลับมาใช้ได้อีกในงานต่อไป

บทที่ 7

การทดลองและผลการทดลอง

7.1 DES/3DES

เราได้ทำไดรเวอร์ให้สามารถทำงานได้ทั้ง des และ 3des โดยใน des สามารถทำได้ทั้งหมด 4 mode คือ ECB, CBC, OFB, CFB การทดสอบโปรแกรม เราได้ให้

รูปที่ 7.1 แสดงภาพหน้าจอขณะรันโปรแกรมทดสอบ

<Encryption>
Jan 6 00:01:59 isag21 kernel: Open cryptochip
Jan 6 00:01:59 isag21 kernel: io_v_base: 32898
Jan 6 00:01:59 isag21 kernel: io_base: 4294377472
Jan 6 00:01:59 isag21 kernel: io_len: 65536
Jan 6 00:01:59 isag21 kernel: PASS_PCI: 1145525264
Jan 6 00:01:59 isag21 kernel: PCI Interrupts line 10
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl for 4004B50A command
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl before switch case
Jan 6 00:01:59 isag21 kernel: IOCTL : INSERT_KEY_L
Jan 6 00:01:59 isag21 kernel: INSERT_KEY_L : key_l = 12345678 → เป็นคีย์ที่เป็นฐาน 16 สั่งให้พิมพ์ออกมาเป็นเลข 16
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl for 4004B509 command
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl before switch case
Jan 6 00:01:59 isag21 kernel: IOCTL : INSERT_KEY_H
Jan 6 00:01:59 isag21 kernel: INSERT_KEY_H : key_h = 12345678
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl for 4004B508 command
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl before switch case
Jan 6 00:01:59 isag21 kernel: IOCTL : ASSIGN_DIRECTION → DIRECTION = 0 เป็น Encryption
Jan 6 00:01:59 isag21 kernel: DIRECTION = 0
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl for 4004B50B command
Jan 6 00:01:59 isag21 kernel: cryptochip_ioctl before switch case
Jan 6 00:01:59 isag21 kernel: IOCTL : ASSIGN_MODE
Jan 6 00:01:59 isag21 kernel: ASSIGN_MODE : data = 0
Jan 6 00:01:59 isag21 kernel: modeop = 0
Jan 6 00:01:59 isag21 kernel: Input Source
Jan 6 00:01:59 isag21 kernel: Hello World → Plain text
Jan 6 00:01:59 isag21 kernel: Hello World
Jan 6 00:01:59 isag21 kernel: 00000^F
Jan 6 00:01:59 isag21 kernel: cryptochip_write
Jan 6 00:01:59 isag21 kernel: count = 32
Jan 6 00:01:59 isag21 kernel: Mode of Operation is ECB → บอก Mode of Operation
Jan 6 00:01:59 isag21 kernel: direction = 0
Jan 6 00:01:59 isag21 kernel: src1 = H
Jan 6 00:01:59 isag21 kernel: tmp1 = 00000048 → ค่า ASCII ของตัว H
Jan 6 00:01:59 isag21 kernel: ret1 = 48000000
Jan 6 00:01:59 isag21 kernel: src2 = e
Jan 6 00:01:59 isag21 kernel: tmp2 = 00000065
Jan 6 00:01:59 isag21 kernel: ret2 = 48650000
Jan 6 00:01:59 isag21 kernel: src3 = l
Jan 6 00:01:59 isag21 kernel: tmp3 = 0000006C
Jan 6 00:01:59 isag21 kernel: ret3 = 48656C00
Jan 6 00:01:59 isag21 kernel: src4 = l
Jan 6 00:01:59 isag21 kernel: tmp4 = 0000006C
Jan 6 00:01:59 isag21 kernel: ret4 = 48656C6C
...
Jan 6 00:01:59 isag21 kernel: src4 = ^F
Jan 6 00:01:59 isag21 kernel: tmp4 = 00000006 → ค่าที่ Pad ลงไปเพื่อให้เต็ม block
Jan 6 00:01:59 isag21 kernel: ret4 = 30303006
Jan 6 00:01:59 isag21 kernel: = 0
Jan 6 00:01:59 isag21 kernel: i = 1
Jan 6 00:01:59 isag21 kernel: x = 48656C6C
Jan 6 00:01:59 isag21 kernel: y = 6F20576F → Plain text block แรก
Jan 6 00:01:59 isag21 kernel: Enable hardware to encrypt
Jan 6 00:01:59 isag21 kernel: Device has been finished its work
Jan 6 00:01:59 isag21 kernel: Cipher[0] = D1E3FDE9
Jan 6 00:01:59 isag21 kernel: Cipher[1] = 412237B3 → Cipher text block แรก
...
Jan 6 00:01:59 isag21 kernel: cryptochip_read function → Cipher text ทั้งหมด
Jan 6 00:01:59 isag21 kernel: direction = 0
Jan 6 00:01:59 isag21 kernel: output = D1E3FDE9412237B324A7000C325558F1385725CF6758C20D2F28AA53BE379A34
Jan 6 00:01:59 isag21 kernel: kree(conv);
Jan 6 00:01:59 isag21 kernel: kree(output);
Jan 6 00:01:59 isag21 kernel: This is cryptochip_release();

```
                        <Decryption>
Jan  6 00:02:25 isag21 kernel: Open cryptochip
...                     <เหมือนกับ Encryption>
Jan  6 00:02:25 isag21 kernel: IOCTL : ASSIGN_MODE
Jan  6 00:02:25 isag21 kernel: ASSIGN_MODE : data = 0
Jan  6 00:02:25 isag21 kernel: modeop = 0
Jan  6 00:02:25 isag21 kernel: Input Source
Jan  6 00:02:25 isag21 kernel: D1E3FDE9412237B324A7000C325558F1385725CF6758C20D2F28AA53BE379A34
Jan  6 00:02:25 isag21 kernel: cryptochip_write
Jan  6 00:02:25 isag21 kernel: count = 64
Jan  6 00:02:25 isag21 kernel: Mode of Operation is ECB
Jan  6 00:02:25 isag21 kernel: direction = 1
Jan  6 00:02:25 isag21 kernel:  = 0
Jan  6 00:02:25 isag21 kernel: i = 1
Jan  6 00:02:25 isag21 kernel: x = D1E3FDE9
Jan  6 00:02:25 isag21 kernel: y = 412237B3
Jan  6 00:02:25 isag21 kernel: Enable hardware to encrypt
Jan  6 00:02:25 isag21 kernel: Device has been finished its work
Jan  6 00:02:25 isag21 kernel: Cipher[0] = 48656C6C
Jan  6 00:02:25 isag21 kernel: Cipher[1] = 6F20576F
...
Jan  6 00:02:25 isag21 kernel: cryptochip_read function
Jan  6 00:02:25 isag21 kernel: direction = 1
Jan  6 00:02:25 isag21 kernel: output = Hello World
Jan  6 00:02:25 isag21 kernel: Hello World
Jan  6 00:02:25 isag21 kernel: 00000^F
Jan  6 00:02:25 isag21 kernel: kree(conv);
Jan  6 00:02:25 isag21 kernel: kree(output);
Jan  6 00:02:25 isag21 kernel: This is cryptochip_release();
```

Cipher text ทั้งหมด

DIRECTION = 1 คือ

Decryption

Cipher text block แรก

Plain text block แรก

Plain text ที่ถูกต้อง

padding

1 CBC Mode

Encryption

```
isag21:"/crypto_des# ./demo -e -f h -M 1 -k 0x12345678 0x12345678 -i 0x12345678 0x12345678 > h.hex
isag21:"/crypto_des# cat h.hex
5D384CA69596532FAC4A01C2D688FBF71AD59E1F34D8BEB5B72CA6200D32E179isag21:"/crypto_des#
isag21:"/crypto_des# ./demo -d -f h.hex -M 1 -k 0x12345678 0x12345678 -i 0x12345678 0x12345678 > h.txt
isag21:"/crypto_des# cat h.txt
Hello World
Hello World
isag21:"/crypto_des#
```

Decryption

รูปที่ 7.2 แสดงภาพผลการทดลองจากการรันโปรแกรมทดสอบ

```
                    <Encryption>
Jan  6 00:12:16 isag21 kernel: Open cryptochip
Jan  6 00:12:16 isag21 kernel: io_v_base: 32898
Jan  6 00:12:16 isag21 kernel: io_base: 4294377472
Jan  6 00:12:16 isag21 kernel: io_len: 65536
Jan  6 00:12:16 isag21 kernel: PASS_PCI: 1145525264
Jan  6 00:12:16 isag21 kernel: PCI Interrupts line 10
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B50A command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : INSERT_KEY_L
Jan  6 00:12:16 isag21 kernel: INSERT_KEY_L : key_l = 12345678
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B509 command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : INSERT_KEY_H
Jan  6 00:12:16 isag21 kernel: INSERT_KEY_H : key_h = 12345678
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B510 command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : INSERT_IV_L
Jan  6 00:12:16 isag21 kernel: INSERT_IV_L : crypto_devices->iv_l = 12345678
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B511 command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : INSERT_IV_H
Jan  6 00:12:16 isag21 kernel: INSERT_IV_H : crypto_devices->iv_h = 12345678
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B508 command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : ASSIGN_DIRECTION
Jan  6 00:12:16 isag21 kernel: DIRECTION = 0
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl for 4004B50B command
Jan  6 00:12:16 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 00:12:16 isag21 kernel: IOCTL : ASSIGN_MODE
Jan  6 00:12:16 isag21 kernel: ASSIGN_MODE : data = 1
Jan  6 00:12:16 isag21 kernel: modeop = 1
Jan  6 00:12:16 isag21 kernel: Input Source
Jan  6 00:12:16 isag21 kernel: Hello World
Jan  6 00:12:16 isag21 kernel: Hello World
Jan  6 00:12:16 isag21 kernel: 00000^F
Jan  6 00:12:16 isag21 kernel: cryptochip_write
Jan  6 00:12:16 isag21 kernel: count = 32
Jan  6 00:12:16 isag21 kernel: Mode of Operation is CBC
Jan  6 00:12:16 isag21 kernel: direction = 0
Jan  6 00:12:16 isag21 kernel: CBC Mode Encryption
Jan  6 00:12:16 isag21 kernel: src1 = H
Jan  6 00:12:16 isag21 kernel: tmp1 = 00000048
Jan  6 00:12:16 isag21 kernel: ret1 = 48000000
Jan  6 00:12:16 isag21 kernel: src2 = e
Jan  6 00:12:16 isag21 kernel: tmp2 = 00000065
Jan  6 00:12:16 isag21 kernel: ret2 = 48650000
```

เป็นคีย์ที่เป็นฐาน 16 สั่งให้พิมพ์ออกมาเป็นเลข 16

เป็น Initial Vector ที่เป็นฐาน 16 สั่งให้พิมพ์ออกมาเป็นเลข 16

บอก Mode of Operation

ค่า ASCII ของตัว H

```
Jan  6 00:12:16 isag21 kernel: src3 = l
Jan  6 00:12:16 isag21 kernel: tmp3 = 0000006C
Jan  6 00:12:16 isag21 kernel: ret3 = 48656C00
Jan  6 00:12:16 isag21 kernel: src4 = l
Jan  6 00:12:16 isag21 kernel: tmp4 = 0000006C
Jan  6 00:12:16 isag21 kernel: ret4 = 48656C6C
...
Jan  6 00:12:16 isag21 kernel: src4 = ^F
Jan  6 00:12:16 isag21 kernel: tmp4 = 00000006
Jan  6 00:12:16 isag21 kernel: ret4 = 30303006
Jan  6 00:12:16 isag21 kernel: XOR Expession
Jan  6 00:12:16 isag21 kernel: firstround = 1
Jan  6 00:12:16 isag21 kernel: conv XOR IV
Jan  6 00:12:16 isag21 kernel: conv[0] = 48656C6C
Jan  6 00:12:16 isag21 kernel: conv[1] = 6F20576F
Jan  6 00:12:16 isag21 kernel: Evaluted conv
Jan  6 00:12:16 isag21 kernel: conv[0] = 5A513A14
Jan  6 00:12:16 isag21 kernel: conv[1] = 7D140117
Jan  6 00:12:16 isag21 kernel: Enable hardware to encrypt
Jan  6 00:12:16 isag21 kernel: Device has been finished its work
Jan  6 00:12:16 isag21 kernel: Cipher[0] = 5D384CA6
Jan  6 00:12:16 isag21 kernel: Cipher[1] = 9596532F
Jan  6 00:12:16 isag21 kernel: XOR Expession
Jan  6 00:12:16 isag21 kernel: firstround = 0
Jan  6 00:12:16 isag21 kernel: conv XOR conv
Jan  6 00:12:16 isag21 kernel: conv[2] = 726C6420
Jan  6 00:12:16 isag21 kernel: conv[0] = 5D384CA6
Jan  6 00:12:16 isag21 kernel: conv[3] = 726C6420
Jan  6 00:12:16 isag21 kernel: conv[1] = 9596532F
Jan  6 00:12:16 isag21 kernel: Evaluted conv
Jan  6 00:12:16 isag21 kernel: conv[2] = 2F542886
Jan  6 00:12:16 isag21 kernel: conv[3] = 9FDE3643
Jan  6 00:12:16 isag21 kernel: Enable hardware to encrypt
Jan  6 00:12:16 isag21 kernel: Device has been finished its work
Jan  6 00:12:16 isag21 kernel: Cipher[2] = AC4A01C2
Jan  6 00:12:16 isag21 kernel: Cipher[3] = D688FBF7
...
Jan  6 00:12:16 isag21 kernel: cryptochip_read function
Jan  6 00:12:16 isag21 kernel: direction = 0
Jan  6 00:12:16 isag21 kernel: output = 5D384CA69596532FAC4A01C2D688FBF71AD59E1F34D8BEB5B72CA6200D32E179
Jan  6 00:12:16 isag21 kernel: kree(conv);
Jan  6 00:12:16 isag21 kernel: kree(output);
Jan  6 00:12:16 isag21 kernel: This is cryptochip_release();
```

ค่าที่ Pad ลงไปเพื่อให้เต็ม block

Plain text block แรก

Plain text block แรก ที่ผ่านการ XOR กับ IV

Cipher text block แรก

Plain text block ที่สอง ที่ XOR กับ cipher text block แรก

Plain text block ที่สอง ที่ผ่านการ XOR กับ cipher text block แรก

Cipher text block ที่สอง

Cipher text ทั้งหมด

<Decryption>
Jan 6 00:12:37 isag21 kernel: Open cryptochip
... <เหมือนกับตอน Encryption>
Jan 6 00:12:37 isag21 kernel: cryptochip_ioctl for 4004B50B command
Jan 6 00:12:37 isag21 kernel: cryptochip_ioctl before switch case
Jan 6 00:12:37 isag21 kernel: IOCTL : ASSIGN_MODE
Jan 6 00:12:37 isag21 kernel: ASSIGN_MODE : data = 1
Jan 6 00:12:37 isag21 kernel: modeop = 1
Jan 6 00:12:37 isag21 kernel: Input Source
Jan 6 00:12:37 isag21 kernel: 5D384CA69596532FAC4A01C2D688FBF71AD59E1F34D8BEB5B72CA6200D32E179
Jan 6 00:12:37 isag21 kernel: cryptochip_write
Jan 6 00:12:37 isag21 kernel: count = 64
Jan 6 00:12:37 isag21 kernel: Mode of Operation is CBC
Jan 6 00:12:37 isag21 kernel: direction = 1
Jan 6 00:12:37 isag21 kernel: CBC Mode Decryption
Jan 6 00:12:37 isag21 kernel: Text_H =5D384CA6
Jan 6 00:12:37 isag21 kernel: Text_L = 9596532F
Jan 6 00:12:37 isag21 kernel: Enable hardware to decrypt
Jan 6 00:12:37 isag21 kernel: Device has been finished its work
Jan 6 00:12:37 isag21 kernel: firstround = 1
Jan 6 00:12:37 isag21 kernel: x = 5A513A14
Jan 6 00:12:37 isag21 kernel: x ^= 12345678 := 48656C6C
Jan 6 00:12:37 isag21 kernel: y = 7D140117
Jan 6 00:12:37 isag21 kernel: y ^= 12345678 := 6F20576F
Jan 6 00:12:37 isag21 kernel: cipher_high = 5D384CA6
Jan 6 00:12:37 isag21 kernel: cipher_low= 9596532F
Jan 6 00:12:37 isag21 kernel: Cipher[0] =48656C6C
Jan 6 00:12:37 isag21 kernel: Cipher[1] =6F20576F
Jan 6 00:12:37 isag21 kernel: Text_H = AC4A01C2
Jan 6 00:12:37 isag21 kernel: Text_L = D688FBF7
Jan 6 00:12:37 isag21 kernel: Enable hardware to decrypt
Jan 6 00:12:37 isag21 kernel: Device has been finished its work
Jan 6 00:12:37 isag21 kernel: firstround = 0
Jan 6 00:12:37 isag21 kernel: x = 2F542886
Jan 6 00:12:37 isag21 kernel: x ^= 5D384CA6 := 726C6420
Jan 6 00:12:37 isag21 kernel: y = 9FDE3643
Jan 6 00:12:37 isag21 kernel: y ^= 9596532F := 0A48656C
Jan 6 00:12:37 isag21 kernel: cipher_high = AC4A01C2
Jan 6 00:12:37 isag21 kernel: cipher_low= D688FBF7
Jan 6 00:12:37 isag21 kernel: Cipher[2] =726C6420
Jan 6 00:12:37 isag21 kernel: Cipher[3] =0A48656C
...
Jan 6 00:12:37 isag21 kernel: cryptochip_read function
Jan 6 00:12:37 isag21 kernel: direction = 1
Jan 6 00:12:37 isag21 kernel: output = Hello World
Jan 6 00:12:37 isag21 kernel: Hello World
Jan 6 00:12:37 isag21 kernel: 00000^F
Jan 6 00:12:37 isag21 kernel: kree(conv);
Jan 6 00:12:37 isag21 kernel: kree(output);
Jan 6 00:12:37 isag21 kernel: This is cryptochip_release();

Cipher text ทั้งหมด

DIRECTION = 1 คือ
Decryption

Cipher text block แรก

Cipher text block แรก
นำมา XOR กับ IV

Plain text block แรก

Plain text block ที่สอง ที่
ผ่านการ decrypt นำมา
XOR กับ cipher text
block แรก

Plain text block แรก

Plain text ออกทั้ง
padding

2. 3DES

```
isag21:~/crypto_des# ./demo -e -f h -M 4 -k 0x12345678 0x12345678 0x12345678 0x12345678 \
> 0x12345678 0x12345678 0x12345678 -i 0x12345678 0x12345678 > h.hex
isag21:~/crypto_des# ./demo -e -f h -M 4 -k 0x12345678 0x12345678 \
> 0x12345678 0x12345678 0x12345678 0x12345678 0x12345678 -i 0x12345678 0x12345678 > h.hex
isag21:~/crypto_des# cat h.hex
5A513A147D140117283D5E34775C647B44527E63182E081F64584E53281E3819isag21:~/crypto_des#
isag21:~/crypto_des# ./demo -d -f h.hex -M 4 -k 0x12345678 0x12345678 \
> 0x12345678 0x12345678 0x12345678 0x12345678 -i 0x12345678 0x12345678 > h.txt
isag21:~/crypto_des# cat h.txt
Hello World
Hello World
isag21:~/crypto_des#
```

รูปที่ 7.3 แสดงภาพหน้าจอตอนรันโปรแกรมทดสอบ

<Encryption>

```
Jan  6 07:05:11 isag21 kernel: Open cryptochip
Jan  6 07:05:11 isag21 kernel: io_v_base: 32898
Jan  6 07:05:11 isag21 kernel: io_base: 4294377472
Jan  6 07:05:11 isag21 kernel: io_len: 65536
Jan  6 07:05:11 isag21 kernel: PASS_PCI: 1145525264
Jan  6 07:05:11 isag21 kernel: PCI Interrupts line 10
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50A command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_L
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_L : key_l = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B509 command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_H
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_H : key_h = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50D command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_L
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_L2 : key_L2 = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50C command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_H
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_H2 : key_H2 = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50F command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_L
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_L3 : key_L3 = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50E command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_KEY_H
Jan  6 07:05:11 isag21 kernel: INSERT_KEY_H3 : key_H3 = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B510 command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_IV_L
Jan  6 07:05:11 isag21 kernel: INSERT_IV_L : crypto_devices->iv_l = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B511 command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : INSERT_IV_H
Jan  6 07:05:11 isag21 kernel: INSERT_IV_H : crypto_devices->iv_h = 12345678
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B508 command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : ASSIGN_DIRECTION
Jan  6 07:05:11 isag21 kernel: DIRECTION = 0
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl for 4004B50B command
Jan  6 07:05:11 isag21 kernel: cryptochip_ioctl before switch case
Jan  6 07:05:11 isag21 kernel: IOCTL : ASSIGN_MODE
Jan  6 07:05:11 isag21 kernel: ASSIGN_MODE : data = 4
Jan  6 07:05:11 isag21 kernel: modeop = 4
```

Key แรก

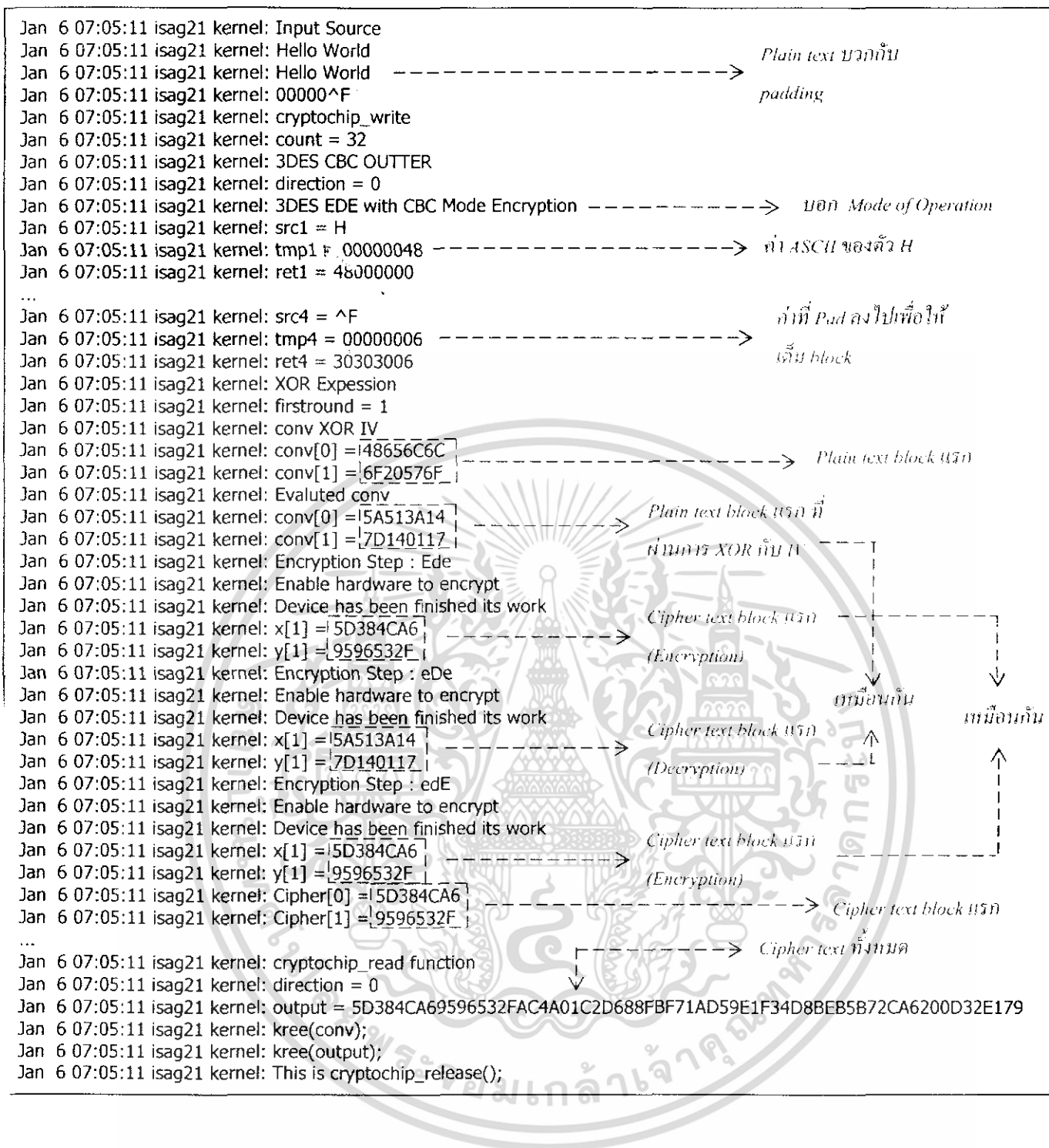
Key ที่สอง

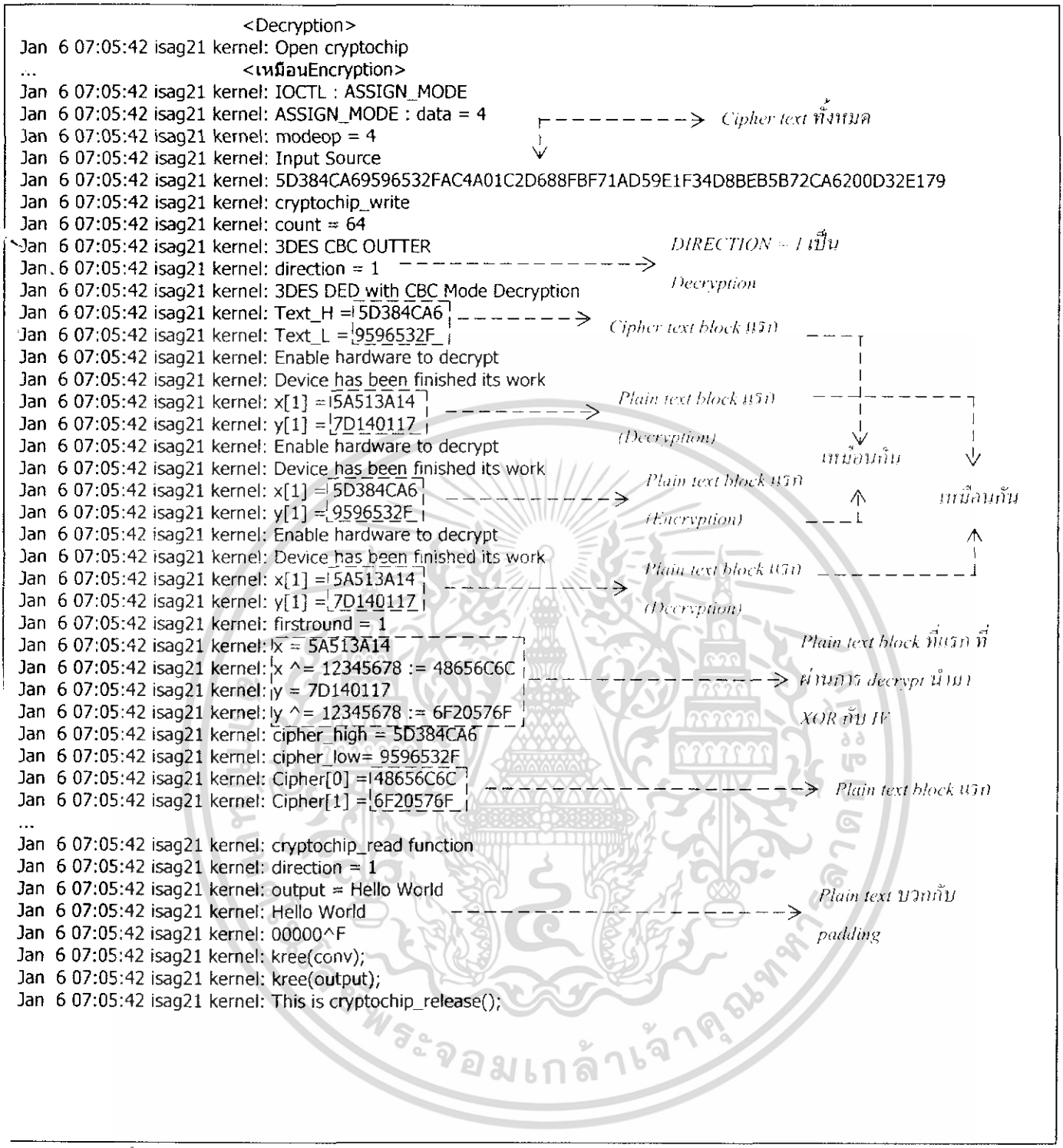
Key ที่สาม

Initial Vector

DIRECTION = 1 คือ Decryption

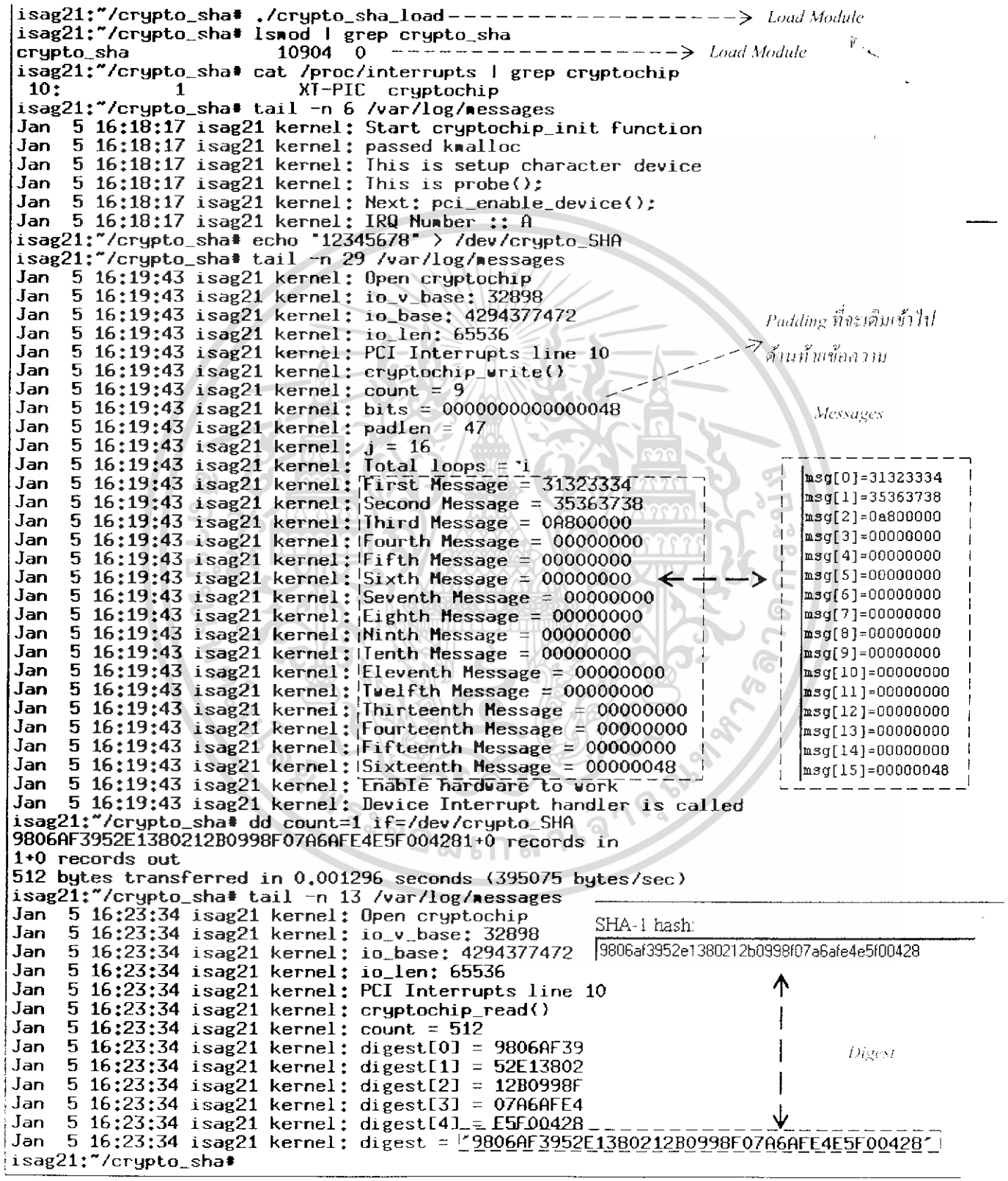
```
Jan  6 07:05:11 isag21 kernel: Input Source
Jan  6 07:05:11 isag21 kernel: Hello World
Jan  6 07:05:11 isag21 kernel: Hello World
Jan  6 07:05:11 isag21 kernel: 00000^F
Jan  6 07:05:11 isag21 kernel: cryptochip_write
Jan  6 07:05:11 isag21 kernel: count = 32
Jan  6 07:05:11 isag21 kernel: 3DES CBC OUTTER
Jan  6 07:05:11 isag21 kernel: direction = 0
Jan  6 07:05:11 isag21 kernel: 3DES EDE with CBC Mode Encryption
Jan  6 07:05:11 isag21 kernel: src1 = H
Jan  6 07:05:11 isag21 kernel: tmp1 = 00000048
Jan  6 07:05:11 isag21 kernel: ret1 = 48000000
...
Jan  6 07:05:11 isag21 kernel: src4 = ^F
Jan  6 07:05:11 isag21 kernel: tmp4 = 00000006
Jan  6 07:05:11 isag21 kernel: ret4 = 30303006
Jan  6 07:05:11 isag21 kernel: XOR Expession
Jan  6 07:05:11 isag21 kernel: firstround = 1
Jan  6 07:05:11 isag21 kernel: conv XOR IV
Jan  6 07:05:11 isag21 kernel: conv[0] =48656C6C
Jan  6 07:05:11 isag21 kernel: conv[1] =6F20576F
Jan  6 07:05:11 isag21 kernel: Evaluted conv
Jan  6 07:05:11 isag21 kernel: conv[0] =5A513A14
Jan  6 07:05:11 isag21 kernel: conv[1] =7D140117
Jan  6 07:05:11 isag21 kernel: Encryption Step : Ede
Jan  6 07:05:11 isag21 kernel: Enable hardware to encrypt
Jan  6 07:05:11 isag21 kernel: Device has been finished its work
Jan  6 07:05:11 isag21 kernel: x[1] =5D384CA6
Jan  6 07:05:11 isag21 kernel: y[1] =9596532F
Jan  6 07:05:11 isag21 kernel: Encryption Step : eDe
Jan  6 07:05:11 isag21 kernel: Enable hardware to encrypt
Jan  6 07:05:11 isag21 kernel: Device has been finished its work
Jan  6 07:05:11 isag21 kernel: x[1] =5A513A14
Jan  6 07:05:11 isag21 kernel: y[1] =7D140117
Jan  6 07:05:11 isag21 kernel: Encryption Step : edE
Jan  6 07:05:11 isag21 kernel: Enable hardware to encrypt
Jan  6 07:05:11 isag21 kernel: Device has been finished its work
Jan  6 07:05:11 isag21 kernel: x[1] =5D384CA6
Jan  6 07:05:11 isag21 kernel: y[1] =9596532F
Jan  6 07:05:11 isag21 kernel: Cipher[0] =5D384CA6
Jan  6 07:05:11 isag21 kernel: Cipher[1] =9596532F
...
Jan  6 07:05:11 isag21 kernel: cryptochip_read function
Jan  6 07:05:11 isag21 kernel: direction = 0
Jan  6 07:05:11 isag21 kernel: output = 5D384CA69596532FAC4A01C2D688FBF71AD59E1F34D8BEB5B72CA6200D32E179
Jan  6 07:05:11 isag21 kernel: kree(conv);
Jan  6 07:05:11 isag21 kernel: kree(output);
Jan  6 07:05:11 isag21 kernel: This is cryptochip_release();
```



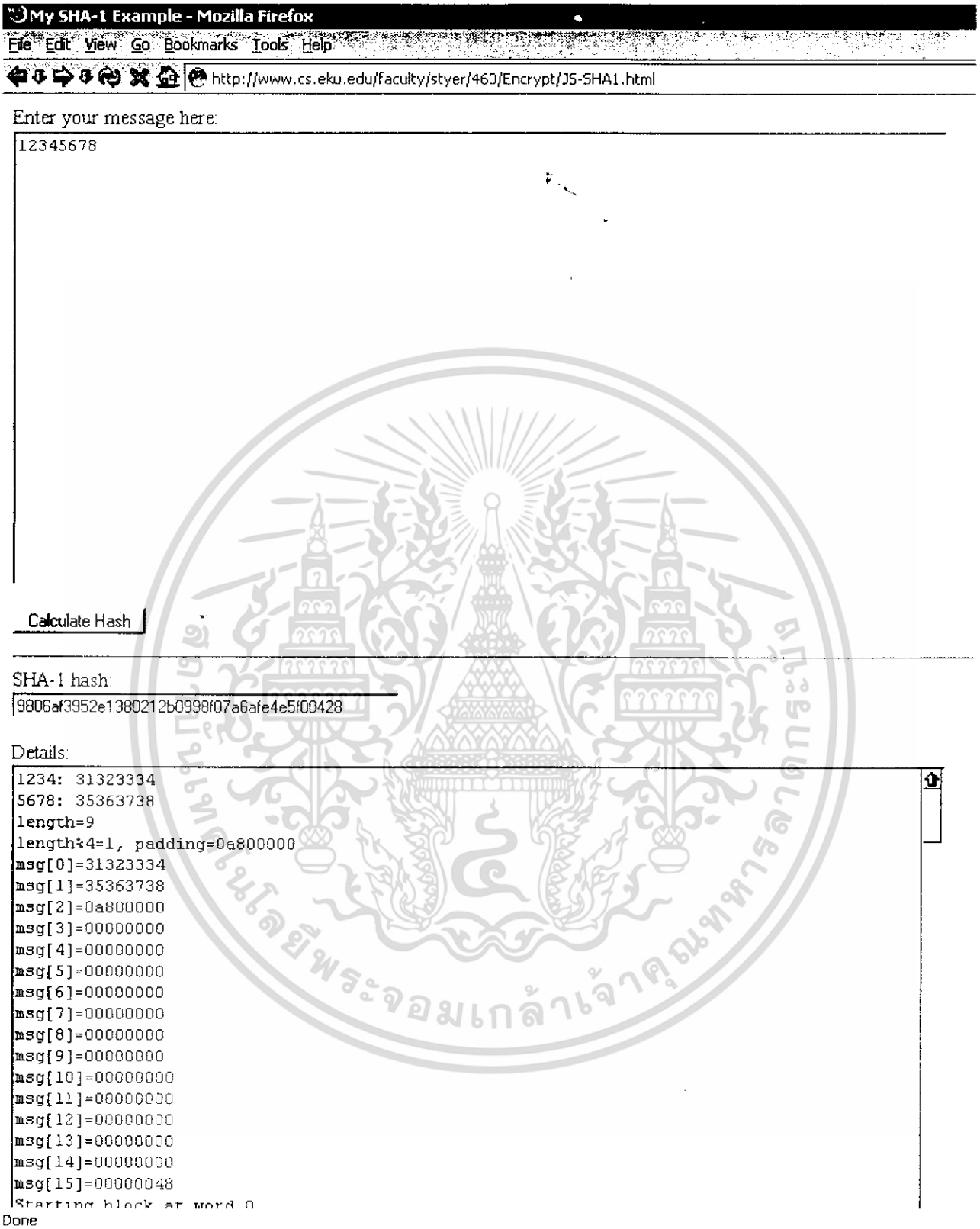


7.2 SHA1

เราสามารถตรวจสอบการทำงานของไดรเวอร์ได้โดยดูจาก log file โดยเราได้เขียนให้ไดรเวอร์นั้นพิมพ์ค่าต่าง ๆ ออกมา แล้วนำผลลัพธ์ไปตรวจสอบกับผลลัพธ์ในเวบ (http://www.cs.eku.edu/faculty/styer/460/Encrypt/JS-SHA1.html)



รูปที่ 7.4 แสดงผลลัพธ์ที่ได้จากการทำ SHA-1

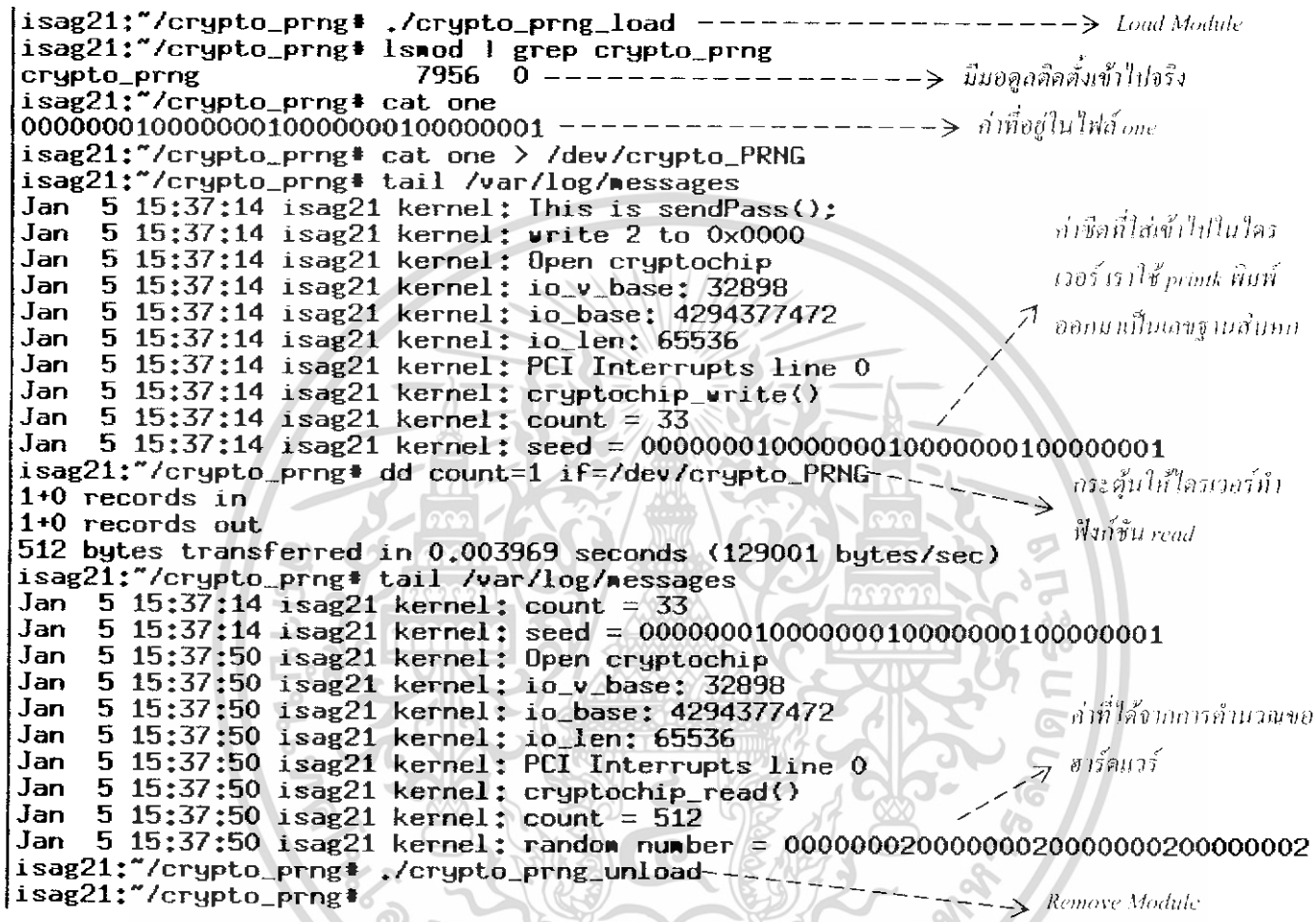


รูปที่ 7.5 ภาพผลลัพธ์ที่ได้จากเวบไซต์ของการทำ SHA-1

7.3 Pseudo Random Number Generator

เนื่องจากเราเลือกใช้ LFSR ดังนั้นเมื่อเรากรอกค่า 1 ลงไปในทุกบล็อกซึ่งมี 4 บล็อก ดังนั้นเราจึงสร้างไฟล์ขึ้นมาเก็บค่า 00000001000000010000000100000001 ลงในไฟล์ชื่อ one

จากนั้นเราใช้คำสั่ง cat เพื่อส่งค่าในไฟล์เข้าไดรเวอร์ และใช้คำสั่ง dd เพื่อให้เกิดการทำงานของฟังก์ชัน read ในไดรเวอร์ แล้วอ่านค่าใน log file ว่าค่าที่ได้ออกมานั้นตรงกันรึเปล่า



รูปที่ 7.6 แสดงผลลัพธ์ที่ได้จากการสร้างเลขสุ่มเทียมจากค่าตั้งต้นที่กำหนด

7.4 การทดสอบประสิทธิภาพการทำงาน

ในการทดสอบประสิทธิภาพการทำงานนั้น มีประเด็นที่ได้ทำการทดสอบคือความเร็วในการทำงาน โดยทำการทดสอบบนระบบที่มีส่วนประกอบดังนี้

- CPU : 166 MHz
- RAM : EDO 92 MB
- OS : Linux kernel 2.6.8-2-386
- PCI CARD : CryptoChip Application Prototype

การทดสอบการเข้าและถอดรหัสลับ

ในการทดสอบจะทำการเข้าและถอดรหัสลับกับไฟล์ที่มีขนาดต่าง ๆ และนำมาหาค่าเฉลี่ยเพื่อจะได้เป็นอัตราเร็วในการทำงาน การนับเวลารวมที่ใช้ในการทำงานนั้นจะใช้การบันทึกเวลาก่อนที่จะมีการงานส่วนของการเข้าและถอดรหัสลับ แล้วนำมาหักลบกับเวลาที่บันทึกหลังจากการทำงานเสร็จสิ้นลง ด้านล่างที่เห็นเป็นส่วนหนึ่งของโค้ดในโปรแกรมทดสอบ โค้ดนี้เป็นฟังก์ชันของการเข้ารหัสลับและฟังก์ชันถอดรหัสลับ จะว่ามีส่วนของการบันทึกเวลาอยู่ด้วย

โหมดของการทำงานนั้นเลือกเป็น CBC และวิธีที่ใช้เป็น 3DES

ฟังก์ชัน encryption()

```
void encryption(char *ctext, char *ptext, int ptext_s, int modeop)
{
    int err;
    int dirc = 0;
    err = ioctl(fd, ASSIGN_DIRECTION, &dirc);
    if (err != 0)
        printf("err = %d\n", err);

    err = ioctl(fd, ASSIGN_MODE, &modeop);
    if (err != 0)
        printf("err = %d\n", err);

    /* Time count */
    time_t startTime, endTime;
    printf("\nEncryption\n");
    /* Start Timer */
```

```
    time(&startTime);
    printf("\nStart Time : %ld\n", (long)startTime);

    /* encryption */
    write(fd, ptext, ptext_s);
    read(fd, ctext, ptext_s*2);

    /* Stop Timer */
    time(&endTime);
    printf("\nEnd Time : %ld\n", (long)endTime);
    printf("\nElapsed Time : %ld seconds\n", (long) (endTime - startTime));
}
```

ฟังก์ชัน decryption()

```
void decryption(char *ctext, char *dtext, int dtext_s, int modeop)
{
    int err;
    int dirc = 1;
    err = ioctl(fd, ASSIGN_DIRECTION, &dirc);
    if (err != 0)
        printf("err = %d\n", err);

    err = ioctl(fd, ASSIGN_MODE, &modeop);
    if (err != 0)
        printf("err = %d\n", err);

    /* Time count */
    time_t startTime, endTime;
    printf("\nDecryption\n");
    /* Start Timer */
    time(&startTime);
    printf("\nStart Time : %ld\n", (long)startTime);
```

```
        /* decryption */
        write(fd, dtext, dtext_s);
        read(fd, ctext, dtext_s/2);

        /* Stop Timer */
        time(&endTime);
        printf("\nEnd Time : %ld\n", (long)endTime);
        printf("\nElapsed Time : %ld seconds\n", (long) (endTime - startTime));
    }
```

ตัวอย่างการทดสอบ

```
isag21:~/crypto_des# ll
totai 256
drw-rw-r--  3 root root   4096 2006-03-12 23:16
-rw-rw-r--  1 root root  34655 2006-02-01 18:32 crypto_des.c
-rw-rw-r--  1 root root   3207 2006-02-08 15:41 crypto_des.h
-rw-r--r--  1 root root  19588 2006-03-12 17:59 crypto_des.ko
-rwxrwxr--  1 root root    980 2005-12-10 11:22 crypto_des_load
-rw-r--r--  1 root root   1524 2006-02-01 18:32 crypto_des.mod.c
-rw-r--r--  1 root root   3520 2006-02-01 18:32 crypto_des.mod.o
-rw-r--r--  1 root root  16668 2006-03-12 17:59 crypto_des.o
-rwxrwxr--  1 root root    184 2005-12-10 11:22 crypto_des_unload
-rwxr-xr-x  1 root root  13911 2006-03-12 23:27 demo
-rw-rw-r--  1 root root  11581 2006-03-12 23:27 demo.c
-rw-r--r--  1 root root 112248 2006-03-12 23:29 devices.txt
-rw-rw-r--  1 root root    455 2006-01-21 13:33 h
drwxr-xr-x  2 root root   4096 2006-02-08 12:08
-rw-rw-r--  1 root root    853 2006-02-01 14:49 Makefile
isag21:~/crypto_des# ./demo -e -f crypto_des.c -M 4 -k 0x12345678 \
> 0x12345678 0x12345678 0x12345678 0x12345678 \
> -i 0x12345678 0x12345678 -o h.hex

Encryption

Start Time : 1142181127

End Time : 1142181135

Elapsed Time : 8 seconds
isag21:~/crypto_des# ./demo -d -f h.hex -M 4 -k 0x12345678 \
> 0x12345678 0x12345678 0x12345678 0x12345678 \
> -i 0x12345678 0x12345678 -o h.txt

Decryption

Start Time : 1142181187

End Time : 1142181191

Elapsed Time : 4 seconds
isag21:~/crypto_des#
```

รูปที่ 7.7 ภาพตัวอย่างการทดสอบเข้าและถอดรหัสลับไฟล์

ผลการทดสอบกับไฟล์ขนาดต่าง ๆ ได้ผลลัพธ์ดังนี้

การทำงาน	ขนาดของไฟล์ (ไบต์)	ระยะเวลาที่ใช้ (วินาที)
เข้ารหัสลับ	445	0
ถอดรหัสลับ	912	0
เข้ารหัสลับ	3207	1
ถอดรหัสลับ	6416	1
เข้ารหัสลับ	34655	8
ถอดรหัสลับ	69312	4

ตารางที่ 7.1 ตารางแสดงผลการทดสอบอัตราเร็วในการเข้าและถอดรหัสลับ

จากผลการทดสอบที่ได้ทำให้เราสามารถนำมาหาค่าเฉลี่ยอัตราเร็วในการเข้าและถอดรหัสได้ โดยผลลัพธ์ที่ได้คือ

- อัตราเร็วในการเข้ารหัสลับคือประมาณ 3769 ไบต์ต่อวินาที
- อัตราเร็วในการถอดรหัสลับคือประมาณ 11872 ไบต์ต่อวินาที

ผลลัพธ์ที่ได้จากการทดสอบพบว่าอัตราเร็วในการเข้ารหัสลับนั้นช้ากว่าการถอดรหัสลับ เพราะว่าการเข้ารหัสลับต้องใช้เวลาส่วนหนึ่งในการแปลงข้อมูลที่เข้ามาให้เป็นอยู่ในรูปแบบที่กำหนด ส่วนการถอดรหัสลับนั้นทำน้อยกว่าเพราะข้อมูลที่เข้ามาเป็นเลขฐานสิบหกอยู่แล้ว

บทที่ 8
บทวิจารณ์และสรุป

8.1 บทสรุป

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัสลับที่พัฒนาขึ้นนี้สามารถทำงานได้ครบทั้งสามฟังก์ชันคือในส่วนของการทำการสุ่มเลขเทียมโดยใช้อัลกอริธึม LFSR การทำแฮชอัลกอริธึมโดยใช้อัลกอริธึม SHA1 และการเข้าและถอดรหัสลับซึ่งเป็นแบบ DES และพัฒนาแบบ 3DES ให้สามารถทำงานได้ดีและได้คำตอบที่ถูกต้อง

งานต้นแบบที่พัฒนาประยุกต์ชิปเข้าและถอดรหัสลับนี้หวังว่าจะเป็นต้นแบบที่สามารถพัฒนาเพื่อไปใช้ร่วมกับงานอื่นๆที่ต้องการความปลอดภัยและความรวดเร็วของข้อมูล เช่น อาจพัฒนาไปใช้ร่วมกับสมาร์ตการ์ด

8.2 วิจารณ์สิ่งที่ได้จากโครงงาน

การ์ดที่ได้นั้นยังมีข้อจำกัดเรื่องของการนำไปใช้งานอยู่ตรงยังไม่สามารถนำไปใช้ในเครื่องเดียวกันได้ ซึ่งข้อพกพร่องตรงนี้เป็นความรับผิดชอบของผู้ผลิตการ์ด ทำให้ประสิทธิภาพของการนำการ์ดนี้ไปใช้ลดลง แต่การพัฒนาการ์ดจากสถาปัตยกรรมที่เป็นไอเอสเอแบบเดิมมาเป็นแบบพีซีไอนั้นทำให้เรื่องของประสิทธิภาพดีขึ้น ไดรเวอร์ที่เราเขียนขึ้นมาเพื่อใช้กับการ์ดนี้ยังไม่ดีเท่าที่ควรเพราะเรื่องของโปรโตคอลที่ใช้ยังไม่เหมาะสม เนื่องจากถ้าโปรแกรมเมอร์ต้องการเข้าและถอดรหัสลับโปรแกรมเมอร์ที่เป็นระดับแอพพลิเคชันต้องเข้าใจการทำงานของดีไวซ์แล้วสั่งงานดีไวซ์ผ่านไอโอคอนโทลซึ่งยังไม่สะดวก ถ้ามีการสร้างมอดูลขึ้นมาครอบไดรเวอร์แล้วทำงานแบบระดับขั้น (Hierarchy) หรือทำเปลี่ยนสตรักเจอร์ที่ใช้จากไฟล์สตรักเจอร์เป็นอย่างอื่นที่เราสร้างขึ้นมาเองให้เหมาะสมจะทำให้ใช้งานได้ง่ายขึ้น

8.3 ปัญหาอุปสรรคและแนวทางแก้ไข

ปัญหาอุปสรรค

1. ต้นแบบนี้เป็นการพัฒนาขึ้นโดยใช้ภาษาซีบนระบบปฏิบัติการลีนุกซ์ จึงพบปัญหาในการเขียนโปรแกรม ในส่วนที่เกี่ยวข้องกับดีไวซ์ไดรเวอร์

2. การโปรแกรมส่วนของดีไวซ์ไดรเวอร์นั้นต้องทำงานร่วมกันกับเคอร์เนลดังนั้นจึงมีความยากลำบากในการทำความเข้าในการทำงานของเคอร์เนลในส่วนที่เกี่ยวข้อง

3. การเขียนโปรแกรมในส่วนดีไวซ์ไดรเวอร์ เมื่อเกิดข้อผิดพลาดเครื่องจะทำงานต่อไม่ได้แล้วต้องทำการรีบูต

4. จำนวนเกตบนบอร์ดนั้นมีจำนวนจำกัดทำให้มีข้อจำกัดในการออกแบบ

5. ในบางครั้ง การจำลองการทำงานสามารถใช้งานได้แต่เมื่อนำไปคอนฟิกกูเรชันแล้วทำงานได้ไม่สมบูรณ์ตามที่ต้องการ

แนวทางการแก้ไขปัญหา

1. ในส่วนของการเขียนโปรแกรมบนระบบปฏิบัติการลีนุกซ์ด้วยภาษาซีนั้น ทางผู้พัฒนาได้แก้ไขปัญหาโดยการค้นคว้าหาข้อมูลจากหนังสือต่าง ๆ ค้นคว้าจากอินเตอร์เน็ต และสอบถามจากอาจารย์ที่ปรึกษาโดยตรง

2. เรื่องของความซับซ้อนของการทำงานของเคอร์เนลนั้นผู้เขียนเลือกที่จะศึกษาเฉพาะในส่วนที่เกิดปัญหาและจำเป็นต้องใช้งาน แต่บางเรื่องก็มีข้อมูลไม่เพียงพอต้องใช้การลองผิดลองถูกบ้าง

3. ในการเขียนโปรแกรมต้องใช้ความรอบคอบมากเป็นพิเศษเพื่อที่จะได้ไม่เกิดข้อผิดพลาดที่ทำให้ต้องรีบูตเครื่องคอมพิวเตอร์ และทางผู้พัฒนาได้บรรเทาปัญหาด้วยการพัฒนาส่วนของดีไวซ์ไดรเวอร์บนเครื่องคอมพิวเตอร์อีกเครื่องหนึ่ง

4. เลือกอัลกอริธึมที่เหมาะสมกับพื้นที่บนบอร์ด ซึ่งในเบื้องต้นผู้วิจัยนั้นมีความสนใจในอัลกอริธึม AES แต่เมื่อได้ศึกษาแล้วพบว่าจำนวนเกตบนบอร์ดนี้ไม่เพียงพอกับบอร์ดที่เราเลือกใช้จึงเปลี่ยนมาศึกษาอัลกอริธึม DES และ 3DES แทนและแบ่งการทำงานบางส่วนให้ซอฟต์แวร์ดีไวซ์ไดรเวอร์

8.4 แนวทางการพัฒนาต่อ

ระบบต้นแบบการพัฒนาต้นแบบประยุกต์ชิบเข้าและถอดรหัสลับ ได้พัฒนาการทำการสุ่มเลขเทียมด้วยอัลกอริธึม LFSR การทำแฮชอัลกอริธึมแบบ SHA1 และการเข้าหรือถอดรหัสลับด้วยอัลกอริธึมแบบ DES และ 3 DES ด้วยฮาร์ดแวร์ ซึ่งทั้ง 3 ส่วนนี้สามารถนำไปประยุกต์ใช้ในสมาร์ทการ์ด, cryptochip, secure disk

ส่วนของลีนุกซ์ดีไวซ์ไดรเวอร์สามารถนำแนวทางไปประยุกต์ใช้กับอุปกรณ์อื่นๆที่ใช้งานผ่านทาง PCI Bus ได้ เมื่อรวมการทำงานของฮาร์ดแวร์ทั้ง 3 ส่วน และการทำงานของลีนุกซ์ดีไวซ์ไดรเวอร์แล้วก็สามารถนำไปประยุกต์ใช้งานในวีพีเอน (VPN) หรือ Crypto API และแอพพลิเคชันอื่นๆที่ต้องการการเข้าและถอดรหัสลับ การสุ่มเลขเทียม และการทำแฮชอัลกอริธึม

บรรณานุกรม

หนังสือและเอกสารอ้างอิง

[1] Linux device drivers :O'Reilly ผู้แต่ง Alessandro Rubini&Jonathan Corbet :

[2] Beginning Linux Programming : Wiley Publishing,inc : ผู้แต่ง Neil Matthew,Richard Stones

[3] ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล, ชำนาญ ปัญญาใส วัชรากร หนูทอง กรุงเทพ, ซีเอ็ดยูเคชัน, 2547

[4] Cryptography and Public Key Infrastructure on the Internet , Klaus Schmeh , WILEY .

เว็บไซต์อ้างอิง

[1] http://www.model.com

[2] http://www.xilinx.com

[3] http://www.faqs.org/rfcs/rfc3174.html

[4] http://www.answers.com/topic/advanced-encryption-standard

[5] http://aleph-proj-alphapp.web.cern.ch/aleph-proj-alphapp/mt19937-1_8cpp-source.html

[6] http://en.wikipedia.org/wiki/Pseudorandom_number_generator

[7] http://www.itl.nist.gov/fipspubs/fip180-1.htm

[8] http://www.google.co.th

[9] http://www.cs.cku.edu/faculty/styer/460/Encrypt/JS-SHA1.html

ภาคผนวก ก.

จำนวนและตำแหน่งของแท็ปของอัลกอริทึม LFSR

ตารางที่ ก.1 แสดงช่วงความยาวของรอบและตำแหน่งของแท็ปเมื่อจำนวนบิตเปลี่ยนไป

จำนวนบิต	ช่วงความยาวของรอบ	ตำแหน่งแท็ป(แบบ Galois)
2	3	0,1
3	7	0,2
4	15	0,3
5	31	1,4
6	63	0,5
7	127	0,6
8	255	1,2,3,7
9	511	3,8
10	1023	2,9
11	2047	1,10
12	4095	0,3,5,11
13	8191	0,2,3,12
14	16383	0,2,4,13
15	32767	0,14
16	65535	1,2,4,15
17	131071	2,16
18	262143	6,17
19	524287	0,1,4,18
20	1048575	2,19
21	2097151	1,20
22	4194303	0,21
23	8388607	4,22
24	16777215	0,2,3,23
25	33554431	2,24

ตารางที่ ก.1 (ต่อ)

26	67108863	0,1,5,25
27	134217727	0,1,4,26
28	268435455	2,27
29	536870911	1,28
30	1073741823	0,3,5,29
31	2147483647	2,30
32	4294967295	1,5,6,31
33	4294967295	12,32
34	17179869183	33,32,31,6
35	34359738367	34,1
36	68719476735	35,2
37	1.37439E+11	36,35,34,33,32,31
38	2.74878E+11	37,36,32,31
39	5.49756E+11	38,3
40	1.09951E+12	39,20,18,1
41	2.19902E+12	40,2
42	4.39805E+12	41,22,21,0
43	8.79609E+12	42,5,4,0
44	1.75922E+13	43,26,25,0
45	3.51844E+13	44,3,2,0
46	7.03687E+13	45,20,19,0
47	1.40737E+14	46,4
48	2.81475E+14	47,27,26,0
49	5.6295E+14	48,8
50	1.1259E+15	49,26,25,0
51	2.2518E+15	50,15,14,0
52	4.5036E+15	51,2
53	9.0072E+15	52,15,14,0
54	1.80144E+16	53,36,35,0
55	3.60288E+16	54,23

ตารางที่ ก.1 (ต่อ)

56	7.20576E+16	55,21,20,0
57	1.44115E+17	56,6
58	2.8823E+17	57,18
59	5.76461E+17	58,21,20,0
60	1.15292E+18	59,0
61	2.30584E+18	60,15,14,0
62	4.61169E+18	61,56,55,0
63	9.22337E+18	62,0
64	1.84467E+19	63,3,2,0
65	3.68935E+19	64,17
66	7.3787E+19	65,9,8,0
67	1.47574E+20	66,9,8,0
68	2.95148E+20	67,8
69	5.90296E+20	68,28,26,1
70	1.18059E+21	69,15,14,0
71	2.36118E+21	70,5
72	4.72237E+21	71,5246,5
73	9.44473E+21	72,24
74	1.88895E+22	73,15,14,0
75	3.77789E+22	74,10,9,0
76	7.55579E+22	75,35,34,0
77	1.51116E+23	76,3029,0
78	3.02231E+23	77,19,18,0
79	6.04463E+23	78,8
80	1.20893E+24	79,37,36,0
81	2.41785E+24	80,3
82	4.8357E+24	81,37,34,2
83	9.67141E+24	82,45,44,0
84	1.93428E+25	83,12
85	3.86856E+25	84,27,26,0

ตารางที่ ก.1 (ต่อ)

86	7.73713E+25	85,12,11,0
87	1.54743E+26	86,12
88	3.09485E+26	87,71,70,0
89	6.1897E+26	88,37
90	1.23794E+27	89,18,17,0
91	2.47588E+27	90,83,82,0
92	4.95176E+27	91,12,11,0
93	9.90352E+27	92,1
94	1.9807E+28	93,20
95	3.96141E+28	94,10
96	7.92282E+28	95,48,46,1
97	1.58456E+29	96,5
98	3.16913E+29	97,10
99	6.33825E+29	98,46,44,1
100	1.26765E+30	99,38
101	2.5353E+30	100,6,5,0
102	5.0706E+30	101,66,65,0
103	1.01412E+31	102,8
104	2.02824E+31	103,10,9,0
105	4.05648E+31	104,15
106	8.11296E+31	105,14
107	1.62259E+32	106,64,62,1
108	3.24519E+32	107,30
109	6.49037E+32	108,6,5,0
110	1.29807E+33	109,12,11,0
111	2.59615E+33	110,9
112	5.1923E+33	111,44,42,1
113	1.03846E+34	112,8
114	2.07692E+34	113,81,80,0
115	4.15384E+34	114,14,13,0

ตารางที่ ก.1 (ต่อ)

116	8.30767E+34	115,70,69,0
117	1.66153E+35	116,19,17,1
118	3.32307E+35	117,32
119	6.64614E+35	118,7
120	1.32923E+36	119,117,110,6
121	2.65846E+36	120,17
122	5.31691E+36	121,59,58,0
123	1.06338E+37	122,1
124	2.12676E+37	123,36
125	4.25353E+37	124,107,106,0
126	8.50706E+37	125,36,35,0
127	1.70141E+38	126,0
128	3.40282E+38	127,28,26,1
129	6.80565E+38	128,4
130	1.36113E+39	129,2
131	2.72226E+39	130,47,46,0
132	5.44452E+39	131,28
133	1.0889E+40	132,51,50,1
134	2.17781E+40	133,56
135	4.35561E+40	134,10
136	8.71123E+40	135,125,124,0
137	1.74225E+41	136,20
138	3.48449E+41	137,7,6,0
139	6.96898E+41	138,7,4,2
140	1.3938E+42	139,28
141	2.78759E+42	140,31,30,0
142	5.57519E+42	141,20
143	1.11504E+43	142,20,19,0
144	2.23007E+43	143,69,68,0
145	4.46015E+43	144,51

ตารางที่ ก.1 (ต่อ)

146	8.9203E+43	145,59,58,0
147	1.78406E+44	146,37,36,0
148	3.56812E+44	147,26
149	7.13624E+44	148,109,108,0
150	1.42725E+45	149,52
151	2.8545E+45	150,2
152	5.70899E+45	151,65,64,0
153	1.1418E+46	152,0
154	2.2836E+46	153,128,126,1
155	4.56719E+46	154,31,30,0
156	9.13439E+46	155,115,114,0
157	1.82688E+47	156,26,25,0
158	3.65375E+47	157,26,25,0
159	7.30751E+47	158,30
160	1.4615E+48	159,18,17,0
161	2.923E+48	160,17
162	5.84601E+48	161,87,86,0
163	1.1692E+49	162,59,58,0
164	2.3384E+49	163,13,12,0
165	4.67681E+49	164,30,29,0
166	9.35361E+49	165,38,37,0
167	1.87072E+50	166,5
168	3.74144E+50	167,16,14,1

ภาคผนวก ข.

คุณสมบัติของเอฟพีจีเอตระกูล

คุณสมบัติของเอฟพีจีเอตระกูล

- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
  - Densities as high as 74,880 logic cells
  - Three power rails: for core (1.2V), I/Os (1.2V to 3.3V), and auxiliary purposes (2.5V)
- SelectIO™ signaling
  - Up to 784 I/O pins
  - 622 Mb/s data transfer rate per I/O
  - 18 single-ended signal standards
  - 6 differential I/O standards including LVDS, RSDS
  - Termination by Digitally Controlled Impedance
  - Signal swing ranging from 1.14V to 3.45V
  - Double Data Rate (DDR) support
- Logic resources
  - Abundant logic cells with shift register capability
  - Wide multiplexers
  - Fast look-ahead carry logic
  - Dedicated 18 x 18 multipliers
  - JTAG logic compatible with IEEE 1149.1/1532
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Clock skew elimination
  - Frequency synthesis
  - High resolution phase shifting

- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE development system
  - Synthesis, mapping, placement and routing
- MicroBlaze™ processor, PCI, and other cores
- Pb-free packaging options
- Low-power Spartan-3L Family and Automotive Spartan-3 XA Family options

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)			Distributed RAM (bits[1])	Block RAM (bits[1])	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000[2, 3]	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500[3]	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000[3]	4M	62,208	96	72	6,912	432K	1,728K	96	4	712	312
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	784	344

รูปที่ ข.1 แสดงรายละเอียดอุปกรณ์ในเอฟพีจีเอตระกูล Spartan 3