โปรแกรมรับส่งข้อมูลแบบปลอดภัย

Secure File Transferring

นายกิตติศักดิ์ ศรีจันทร์กุล
นายก่อ กิตติกรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

โปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย

Secure File Transferring

โดย

นายกิตติศักดิ์ ศรีจันทร์กุล

นายก่อ กิตติกรณ์

อาจารย์ที่ปรึกษา

อ.ธนา หงษ์สุวรรณ

อ.อัครเดช วัชระภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย

Secure File Transferring

ผู้จัดทำ

1. นายกิตติศักดิ์ ศรีจันทร์กุล รหัสประจำตัว 40010058
2. นายก่อ กิตติกรณ์ รหัสประจำตัว 40010088

_____ อาจารย์ที่ปรึกษา

(อาจารย์ธนา หงษ์สุวรรณ)

_____ อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชระภูพงษ์)

โปรแกรมรับส่งข้อมูลแบบปลอดภัย

นายกิตติศักดิ์ ศรีจันทร์กุล
นายก่อ กิตติกรณ์
อ.ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อ.อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในสภาวะปัจจุบันกระแสเทคโนโลยีสารสนเทศและอินเทอร์เน็ต มีส่วนสำคัญและมีอิทธิพลอย่างยิ่งต่อชีวิตมนุษย์ในทุกๆ ด้าน ประเด็นเรื่องความปลอดภัยของข้อมูลจึงเป็นเรื่องที่ควรให้ความสำคัญมากขึ้น เนื่องจากโปรแกรมประยุกต์ที่ทำงานบนอินเทอร์เน็ตส่วนใหญ่นั้นไม่ได้คำนึงถึงความปลอดภัยเท่าใด

โครงงานนี้ได้พัฒนาโปรแกรมเอฟทีพี ซึ่งเป็นที่นิยมใช้กันอย่างมากในการโอนย้ายแฟ้มข้อมูลในระบบเครือข่ายให้มีความปลอดภัยมากขึ้น โดยได้ประยุกต์ใช้กับโพรโตคอลเซ็กเคียวเชลล์ซึ่งเป็นที่นิยมใช้กันอย่างแพร่หลายในเชิร์ฟเวอร์ที่เปิดให้บริการแบบระยะไกล อีกยังมีการเข้ารหัสและการพิสูจน์ที่ดีมีประสิทธิภาพ สามารถป้องกันการโจมตีได้หลายรูปแบบ ในการพัฒนาโปรแกรมนี้ได้ใช้ภาษาจาวาเป็นตัวพัฒนา เพื่อให้มีความสามารถในการทำงานได้ในทุกระบบปฏิบัติการ

โปรแกรมในโครงงานนี้พัฒนาขึ้นมาเพื่อให้เกิดการใช้งานจริง โดยได้ออกแบบให้มีลักษณะเป็นคอมโพเน็นต์ เพื่อที่จะสามารถปรับปรุง แก้ไข พัฒนาเพิ่มต่อไปได้ง่าย

Secure File Transferring

Kittisak Sriechankul
Kor Kittikorn
Thana Hongsuwan Advisor
Akkradach Watcharapupong Advisor

ABSTRACT

Nowadays, Globalization and Internet technology become highly influenced on human life. Consequently, the security of data is in issue. Due to most of Internet application doesn't regard about security

Under this project, FTP program mostly use for network file transferring is developed for more security by applying to secure shell protocol, that is widely used in many servers providing remote service. Beside the secure shell protocol provide efficiency encryption and strong authentication which is uses to defend against the attacks known today. This project is developed by java language in addition to support any operating system.

This program is developed to be more practical for all uses, which is component designed for advantage of maintain and more development in further

กิตติกรรมประกาศ

โครงงานและปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ได้ เนื่องจากคำแนะนำจากอาจารย์ที่ปรึกษาทั้ง 2 ท่านคือ อาจารย์ธนา หงษ์สุวรรณ และอาจารย์อัครเดช วัชระภูพงษ์ ที่คอยแนะนำและเอาใจใส่กับการทำโครงงานนี้เป็นอย่างดี ซึ่งทางคณะผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษาทั้งสองท่านเป็นอย่างสูง

นอกเหนือจากนี้ก็ต้องขอขอบพระคุณคณะอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอย่างยิ่งที่ได้ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ อีกทั้งภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้เอื้อเฟื้ออุปกรณ์ทรัพยากร สถานที่และอำนวยความสะดวกต่างๆในการทำโครงงานนี้ด้วย

ขอขอบคุณเพื่อนๆ ชาว ISAG ได้แก่ พี่ตั้ม พี่ต่อ พี่เส็ง โอ่ง นก แต บู้ เบียร์ แนน ซี เป้ หนึ่ง กระบี่ ที่ช่วยเหลือในการทำงานและแก้ไขอุปสรรคต่างๆ ในการทำงานให้ผ่านพ้นไปด้วยดี และขอบคุณกำลังใจจากตุ๊กตาและมัสลิน ที่ยังทำให้มีแรงต่อสู้กับความยากลำบากต่อไป

สุดท้ายนี้ต้องขอขอบพระคุณบุคคลที่สำคัญที่สุดคือ บิดา มารดา ที่เคารพและเป็นที่รักยิ่ง ผู้ที่ให้กำเนิด คอยสั่งสอน ให้การศึกษาอย่างสูงสุด พร้อมทั้งการให้การสนับสนุนปัจจัยในด้านต่างๆ นับเป็นพระคุณอย่างสูงสุดหาที่เปรียบมิได้ คณะผู้จัดทำขอระลึกพระคุณอันยิ่งใหญ่สุดประมาณนี้ไว้กว่าชีวิตจะหาไม่ และกราบขอบพระคุณทุกท่านไว้ ณ ที่นี้ด้วย

กิตติศักดิ์ ศรีจันทร์กุล

ก่อ กิตติกรณ์

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ที่มาของปัญหา

อย่างที่ทราบกันดีแล้วว่า โพรโตคอลทีซีพี ไอพีพัฒนาขึ้นมาโดยไม่ได้คำนึงถึงหลักความปลอดภัยของข้อมูล ข้อมูลที่รับส่งกันในโพรโตคอลทีซีพี ไอพีนั้นเป็นข้อมูลที่สามารถลักลอบบันทึก (tapping หรือ eavedropping) โดยเครื่องที่อยู่บนเครือข่าย โดยเฉพาะอย่างยิ่งเครือข่ายระดับข่ายท้องถิ่น เช่น อีเทอร์เน็ต (Ethernet) หรือโทกเคนริง (Token Ring) เนื่องจากข้อมูลดังกล่าวนั้นมีลักษณะเป็นข้อความธรรมดาที่ไม่ได้มีการเข้ารหัสแต่อย่างใดสามารถตีความได้เลย และเนื่องจากที่เครือข่ายอินเทอร์เน็ตนั้นมีขนาดใหญ่มาก การส่งข้อมูลไปในอินเทอร์เน็ตจึงต้องอาศัยเราเตอร์ ในการหาเส้นทางเพื่อให้ข้อมูลสามารถส่งต่อไปเรื่อยๆ จนไปยังจุดหมายที่ต้องการได้ จะเห็นได้ว่าข้อมูลที่ส่งไปยังอาจถูกดักจับ ปลอมแปลงได้โดยง่าย

ในปัจจุบันนี้มีการขยายตัวของการใช้อินเตอร์เน็ตใช้ชีวิตประจำวันประกอบกับความเป็นจริงที่ว่าระบบคอมพิวเตอร์มีความจำเป็นที่ต้องเกี่ยวข้องกับระบบเน็ตเวิร์กมากขึ้นเรื่อยๆ มีผลทำให้มีการเพิ่มของการละเมิดความเป็นส่วนตัวของผู้ใช้คอมพิวเตอร์เพิ่มขึ้นเป็นเงาตามตัว การแก้ไขปัญหาในจุดนี้ไม่สามารถแก้ได้ในตัวของระบบหรือจุดใหญ่ๆ หากแต่ขึ้นอยู่กับจุดเล็กๆ ที่ประกอบกันเป็นระบบ

หากมองจากภาพรวมแล้วการเข้าใช้งานกับเครื่องเซิร์ฟเวอร์จากระยะไกลผ่านทางเครือข่ายหรือที่เรียกว่ารีโมตล็อกอิน (remote login) นั้นมีการใช้งานกันอย่างแพร่หลาย มีโปรแกรมต่างๆ ที่ใช้งานกันมากเช่นโปรแกรมเอฟทีพี (FTP) , เทลเน็ต (Telnet) จากการส่งข้อมูลในโปรแกรมดังกล่าวเป็นข้อมูลชนิดแพลนเท็กซ์ (Plain text) ที่ไม่มีการเข้ารหัสข้อมูลทำให้ไม่มีความปลอดภัยในการเข้าใช้งาน เนื่องจากอาจโดนผู้ที่ไม่ประสงค์ดีลักลอบดักหรือปลอมแปลงข้อมูลของเราได้ ดังนั้นเพื่อเป็นการป้องกันพฤติกรรมดังกล่าว จึงต้องมีการเข้ารหัสข้อมูลที่ส่ง ในที่นี้ใช้โพรโตคอลเซ็กเคียวเซลล์ที่ได้มีการกำหนดเป็นมาตรฐานใช้งานกันอย่างแพร่หลาย เพราะมีวิธีการพิสูจน์สิทธิ์และมีวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพก่อนติดต่อและส่งข้อมูล แม้ว่าผู้ที่ไม่ประสงค์ดีได้รับข้อมูลเหล่านั้นไปก็ไม่อาจทราบได้ว่าเป็นข้อมูลอะไร

โครงงานนี้จะประยุกต์ใช้โพรโตคอลเซ็กเคียวเซลล์กับโปรแกรมเอฟทีพี เพื่อความปลอดภัยของข้อมูลจากการลักลอบบันทึกของผู้ประสงค์ไม่ดี โดยเฉพาะอย่างยิ่งรหัสผ่านของผู้ใช้

1.2 วัตถุประสงค์ของโครงงาน

- เพื่อศึกษาแนวทางในการรับส่งแฟ้มข้อมูลแบบปลอดภัย
- เพื่อสร้างโปรแกรมต้นแบบที่สามารถรับส่งแฟ้มข้อมูลที่มีความปลอดภัย

1.3 ขอบเขตโครงงาน

- สร้างโปรแกรมรับส่งแฟ้มข้อมูล ที่ใช้โพรโตคอลเซ็กเคียวเซลล์เพื่อเข้ารหัสข้อมูลที่ติดต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์
- พัฒนาโปรแกรมจาวาเพื่อให้โปรแกรมที่พัฒนาสามารถใช้งานได้ในทุกระบบปฏิบัติการ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- เพิ่มความปลอดภัยให้กับการใช้งานเอฟทีพี โดยป้องกันการตีความหมายของการดักบันทึกข้อมูล ที่รับส่งอยู่บนโปรแกรมเอฟทีพี
- เพื่อกระตุ้นให้เห็นถึงความสำคัญของการรักษาความปลอดภัย
- สร้างโปรแกรมที่จะจัดการการใช้งานเอฟทีพีให้มีความปลอดภัย

บทที่ 2
ทีซีพี/ไอพี

2.1 ประวัติความเป็นมา (Carl –Michel,1993; Hunt,1992)

ในปี ค.ศ. 1969 หน่วยงาน ARPA (Advance Research Project Agency) ซึ่งเป็นหน่วยงานหนึ่งในกระทรวงกลาโหมของสหรัฐอเมริกาได้ทดลองเครือข่ายแบบสลับแพ็กเก็ต (packet switching network) โดยเรียกว่า เครือข่ายอาร์พาเน็ต (Arpanet) ซึ่งเป็นเครือข่ายแบบสลับแพ็กเก็ตแรกในโลก เพื่อใช้สำหรับการศึกษาวิจัยรวมถึงมีบทบาทสำคัญทางทหาร โดยเฉพาะอย่างยิ่งสงครามโลกครั้งที่ 1

ต่อมาในปี ค.ศ. 1975 เครือข่ายอาร์พาเน็ตได้ถูกใช้งานเป็นเครือข่ายปฏิบัติงานจริงโดยอยู่ในความรับผิดชอบของหน่วยงานสื่อสารทางทหาร (Defense Communication Agency) ซึ่งในช่วงนี้เองโพรโตคอลทีซีพี/ไอพี (TCP/IP protocol) ได้เริ่มถูกพัฒนาขึ้น

โพรโตคอลทีซีพี/ไอพี ถูกกำหนดให้เป็นมาตรฐานทางทหาร (Military Standards) ในปี ค.ศ.1993 โดยกำหนดให้ทุกเครื่องที่ต้องการเชื่อมโยงกับเครือข่ายนี้ จะต้องเปลี่ยนแปลงโพรโตคอลให้เป็นไปตามโพรโตคอลทีซีพี/ไอพี อาร์พาเน็ตเดิมถูกแบ่งเป็น 2 ส่วนคือ มิลเน็ต (Milnet) สำหรับการใช้งานทางทหาร และอาร์พาเน็ตที่มีขนาดเล็กลง โดยคำว่าอินเทอร์เน็ตยังได้รวมเอาเครือข่ายต่างๆ เข้ามาเป็นส่วนหนึ่งของเครือข่าย เช่น เอ็นเอฟเอสเน็ต (NSFnet) เป็นต้น

หลังจากที่อินเทอร์เน็ตถูกใช้งานเป็นเครือข่ายที่ใหญ่ที่สุดในโลก ทำให้การใช้งานโพรโตคอลทีซีพี/ไอพีแพร่หลายและเป็นโพรโตคอลที่สำคัญและมีการใช้งานมากที่สุดโพรโตคอลหนึ่ง

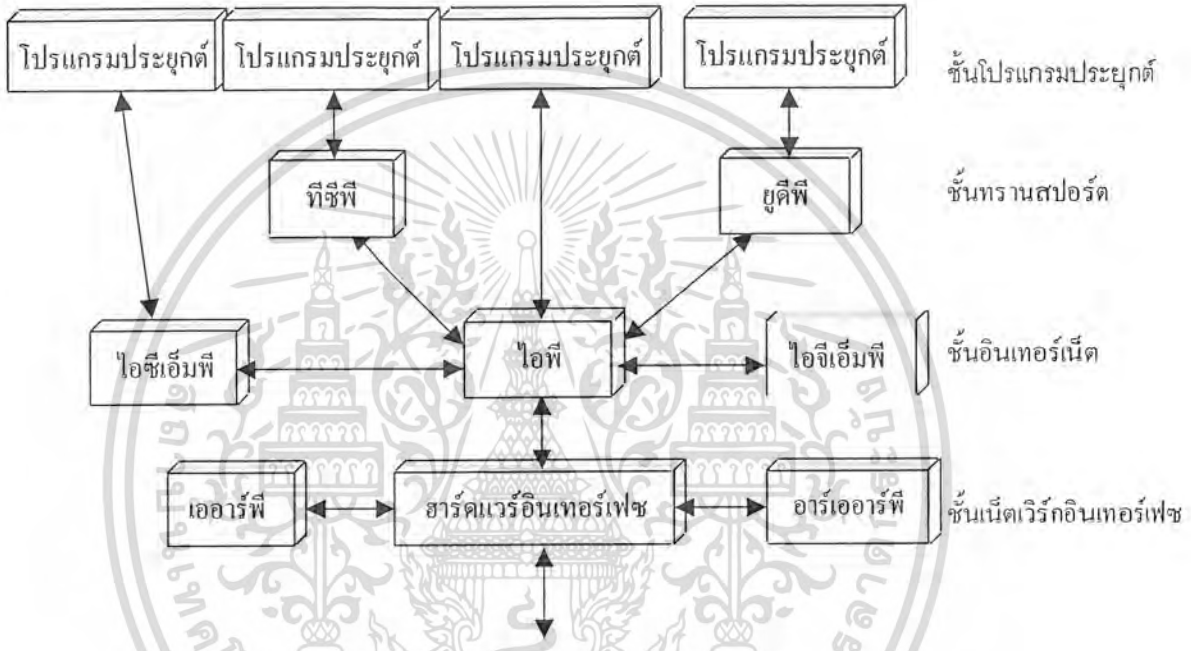
2.2 สถาปัตยกรรมของโพรโตคอล (protocol architecture) (Hunt,1992)

โพรโตคอลทีซีพี/ไอพี เป็นโพรโตคอลที่ได้รับความนิยมไม่ใช่เพราะว่าอินเทอร์เน็ต หรือทางทหารสหรัฐใช้ แต่เนื่องจากคุณสมบัติที่ดีของโพรโตคอล ได้แก่

- เป็นโพรโตคอลมาตรฐานของระบบเปิด โพรโตคอลทีซีพี/ไอพีสามารถพัฒนาได้โดยไม่ขึ้นอยู่กับฮาร์ดแวร์ หรือระบบปฏิบัติการใดๆ
- เป็นโพรโตคอลที่เป็นอิสระจาก เครือข่ายทางกายภาพ (physical network) เนื่องจากโพรโตคอลทีซีพี ไอพี สามารถทำงานได้บนอีเทอร์เน็ต (Ethernet), โทกเคนริง (Token ring), เอฟดีดีไอ (FDDI) พอร์ตอนุกรมอาร์เอส 232 (RS /232) หรือเครือข่ายสลับแพ็กเก็ตเอ็กซ์ 25 (X.25) เป็นต้น
- การกำหนดที่อยู่ (addressing) เป็นรูปแบบที่จำแนกเครื่องต่างๆ และเครือข่ายได้เป็นอย่างดี
- ความเป็นมาตรฐานของโพรโตคอลระดับบน ซึ่งมีการใช้งานอย่างกว้างขวาง เช่น เทลเน็ต (Telnet) , เอฟทีพี (FTP) เป็นต้น

ชั้นเลเยอร์ของโพรโตคอลทีซีพี/ไอพี มี 4 ชั้น แบ่งจากชั้นบนลงล่างดังนี้

- ชั้นโปรแกรมประยุกต์ (application layer)
- ชั้นทรานสปอร์ต (transport layer)
- ชั้นอินเทอร์เน็ต (Internet layer)
- ชั้นเน็ตเวิร์กอินเทอร์เฟซ (network interface layer)



รูปที่ 2-1 แสดงความสัมพันธ์ของชั้นต่างๆของโพรโตคอลทีซีพี/ไอพี

2.2.1 ชั้นโปรแกรมประยุกต์ (appplication layer)

ชั้นโปรแกรมประยุกต์เป็นชั้นของโพรโตคอลสำหรับโปรแกรมประยุกต์บนเครือข่ายทีซีพี/ไอพี รูปแบบของโปรแกรมประยุกต์ในชั้นนี้จะเป็นลักษณะของผู้ขอรับบริการและผู้ให้บริการ (client / server model) นั่นคือการใช้งานโปรแกรมประยุกต์หนึ่งๆจะมีโพรเซสหนึ่งทำหน้าที่เป็น โพรเซสขอรับบริการ (Client process) และโพรเซสสำหรับให้บริการ (server process) ผ่านเครือข่ายโดยมีโพรโตคอลสำหรับโปรแกรมประยุกต์ร่วมกัน โปรแกรมประยุกต์ต่างๆบนโพรโตคอลทีซีพี/ไอพี เช่น เทลเน็ต (Telnet) , เอฟทีพี (FTP) ,เอสเอ็มทีพี (STMP)เป็นต้น

2.2.2 ชั้นทรานสปอร์ต (Transport layer)

ชั้นการรับส่งหน้าที่ควบคุมการรับส่งระหว่างกัน ในชั้นนี้มีโพรโตคอลหลักๆ 2 โพรโตคอลคือ ทีซีพี (TCP) และยูดีพี (UDP)

ทีซีพี มีชื่อเต็มว่า ทรานส์มิชชันคอนโทรลโพรโตคอล (Transmission Control Protocol) เป็นโพรโตคอลแบบต้องมีการติดต่อกันก่อนของผู้รับส่งข้อมูล (Connection-oriented) ซึ่งทำให้เป็นโพรโตคอลที่มีความเชื่อถือได้ (Reliable) ของการรับส่ง ยูดีพี มีชื่อเต็มว่า ยูสเซอร์ดาต้าแกรมโพรโตคอล (User Datagram Protocol)เป็นโพรโตคอลที่มีการทำงานง่ายกว่าทีซีพี การรับส่งข้อมูลมีลักษณะ ดาต้าแกรม (Datagram) ข้อมูลที่รับส่งไม่จำเป็นจะต้องมีการติดต่อกันก่อน (connectionless) ทำให้การรับส่งข้อมูลขาดความเชื่อถือได้

2.2.3 ชั้นอินเทอร์เน็ต (internet layer)

ชั้นอินเทอร์เน็ตมีชื่อเรียกอีกอย่างหนึ่งว่า ชั้นเครือข่าย (network layer) ในชั้นนี้โพรโตคอลที่สำคัญคือ ไอพี (IP) อันเป็นหัวใจของชุดโพรโตคอลทีซีพี/ไอพี หน้าที่สำคัญคือการส่งข้อมูล (Route) ไปตามจุดต่างๆ ในเครือข่าย นอกจากไอพีแล้วในชั้นนี้ยังมีโพรโตคอลอีก 2 โพรโตคอลคือ ไอซีเอ็มพี (ICMP: Internet Control Message Protocol) และไอจีเอ็มพี (IGMP : Internet Group Management Protocol)

2.2.4 ชั้นเน็ตเวิร์กอินเทอร์เฟซ (network Interface Layer)

ชั้นเน็ตเวิร์กอินเทอร์เฟซมีชื่อเรียกอีกอย่างหนึ่งว่า ชั้นดาต้าลิงก์ (Data link layer) หรือชั้นลิงก์ (link layer)ทำหน้าที่เป็นดีไวซ์ไดร์เวอร์ (Device driver) ของระบบปฏิบัติการ เพื่อทำให้สามารถรับส่งข้อมูลกับฮาร์ดแวร์ สำหรับการสื่อสารข้อมูลในเครือข่าย

2.3 การอ้างอิงที่อยู่ของไอพีและพอร์ต (IP addressing and port)

ในระบบโพรโตคอลสำหรับเครือข่ายต่างๆ นั้นจำเป็นจะต้องมีวิธีการในการอ้างอิงที่อยู่ของจุด (node) ต่างๆ ซึ่งโพรโตคอลทีซีพี/ไอพีจะใช้ที่อยู่ของไอพี (IP address) ในการกำหนดที่อยู่ของจุดต่างๆ ในระบบเครือข่ายที่อยู่ของไอพีจะเป็นตัวเลขขนาด 32 บิต โดยแสดงในรูปเลขฐาน 10 จำนวน 4 ตัว คั่นกลางด้วยจุด ตัวอย่างเช่น 192.1.1.1 , 148.1.2.35 เป็นต้น

ที่อยู่ของไอพี แบ่งเป็น 2 ส่วนได้แก่

- ที่อยู่ของเครือข่าย (network address)
- ที่อยู่ของเครื่อง (host address)

ที่อยู่ของเครือข่ายจะเป็นการบ่งบอกที่อยู่ของจุดนั้นๆ อยู่ที่เครือข่ายใด และที่อยู่ของเครื่องเป็นที่อยู่ที่บอกว่าในเครือข่ายนั้นเป็นเครื่องใด

ในโพรโตคอลทีซีพี/ไอพีแบ่งระดับ (Class) ของที่อยู่ไอพี เป็น 5 ระดับ ดังนี้

ระดับ	จำนวนหลักของที่อยู่ของเครือข่าย	ช่วงของที่อยู่ของเครือข่าย	จำนวนหลักที่อยู่ของที่อยู่ของเครื่อง	ช่วงของที่อยู่ของเครื่อง
A	1	1-127	3	0.0.0-255.255.255
B	2	128.0-191.255	2	0.0-255.255
C	3	192.0.0-222.255.255	1	0.255
D	4	224.0.0.0-239.255.255.255	-	-
E	5	240.0.0.0-255.255.255.255	-	-

ตารางที่ 2-1 แสดงการแบ่งระดับ (Class) ของที่อยู่ไอพี

ที่อยู่ของไอพีที่ถูกสำรองไว้ โดยมีความหมายอย่างอื่นได้แก่

- ที่อยู่ไอพีที่ส่วนของที่อยู่ของเครื่องทั้งหมดเป็น 0 หมายถึง การอ้างอิงที่อยู่ของเครือข่าย
- ที่อยู่ไอพีส่วนของที่อยู่ของเครื่องทั้งหมดเป็น 1 หมายถึงการอ้างอิงที่อยู่สำหรับการประกาศ (Broadcast Address)
- ที่อยู่ไอพี 127.0.0.1 หมายถึง ที่อยู่ภายในเครื่องนั้น
- ที่อยู่ไอพี 255.255.255.255 หมายถึง ทุกเครื่อง

ในการทำงานของโปรแกรมประยุกต์ของโพรโตคอลทีซีพี/ไอพี ทั้งที่ใช้โพรโตคอลทีซีพีและยูดีพี ต้องมีกระบวนการในการระบุถึงโพรเซส ที่ทำการติดต่อด้วยซึ่งทั้งทีซีพีและยูดีพีใช้เลข 16 บิต เรียกว่า พอร์ต (port) โพรเซสให้บริการจะต้องมีพอร์ตที่เป็นที่รู้จักกันทั่วไป (wellknown port) เพื่อให้ผู้ขอรับบริการสามารถที่จะติดต่อมาได้ เช่น โพรเซสให้บริการเทลเน็ต (Telnet serverser) จะใช้พอร์ต 23 หรือโพรเซสให้บริการเอฟทีพี (FTP server) จะใช้พอร์ต 21

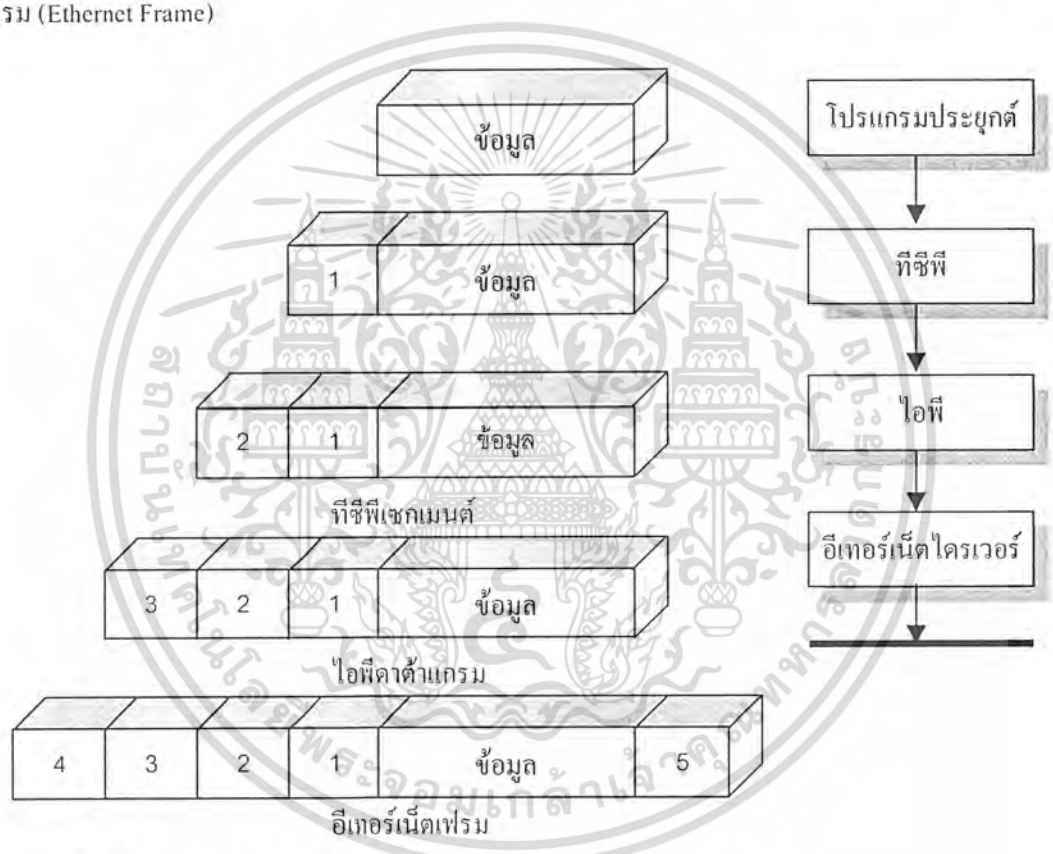
สำหรับโพรเซสขอรับบริการจะกำหนดพอร์ตขึ้นมาพอร์ตหนึ่งเพื่อใช้ติดต่อกับพอร์ตของโพรเซสให้บริการ ดังนั้นในแต่ละการติดต่อระหว่างโพรเซสขอรับบริการและโพรเซสให้บริการจะใช้ค่าที่อ้างอิง 5 ค่า ดังต่อไปนี้

- โพรโตคอลทีซีพีหรือยูดีพี
- ที่อยู่ไอพีของตนเอง
- พอร์ตของตนเอง
- ที่อยู่ของไอพีอีกด้านหนึ่ง
- พอร์ตของอีกด้านหนึ่ง

2.4 การห่อหุ้มแพ็กเก็ต (packet encapsulation)

ในการสื่อสารข้อมูลตามโพรโตคอลทีซีพี/ไอพีระหว่างการผ่านชั้นต่างๆ จากบนลงมาล่าง ข้อมูลจะถูกส่งเป็นรูปของแพ็กเก็ต (Packet) โดยในแต่ละระดับชั้นจะมีการเพิ่มส่วนหัว (header) ของแพ็กเก็ตเพื่อใช้สื่อสารกับโพรโตคอลระดับเดียวกันของอีกฝ่ายหนึ่ง

หน่วยข้อมูลที่ถูกเพิ่มเฮดเดอร์ของทีซีพีเรียกว่า ทีซีพีเซกเมนต์ (TCP segment) เมื่อส่งลงไปในชั้นไอพี มีการเพิ่มเฮดเดอร์ของไอพี เรียกว่า ไอพีดาต้าแกรม (IP Datagram) และเมื่อไอพีดาต้าแกรมถูกส่งลงไปในชั้นเน็ตเวิร์กอินเตอร์เฟซ เช่น อีเทอร์เน็ต (Ethernet) จะมีการเพิ่มเฮดเดอร์และเทรลเลอร์ เรียกว่า อีเทอร์เน็ตเฟรม (Ethernet Frame)



1) เฮดเดอร์ของโปรแกรมประยุกต์ (application header)

2) เฮดเดอร์ของทีซีพี (TCP header)

3) เฮดเดอร์ของไอพี (IP header)

4) เฮดเดอร์ของอีเทอร์เน็ต (Ethernet header)

5) เทรลเลอร์ของอีเทอร์เน็ต

รูปที่ 2-2 การห่อหุ้มแพ็กเก็ต

2.5 อีเทอร์เน็ตเฟรม (Ethernet frame)

โพรโตคอลทีซีพี/ไอพี เป็นโพรโตคอลที่สามารถใช้งานได้กับชนิดของการรับส่งข้อมูลหลายรูปแบบ เช่น อีเทอร์เน็ต (Ethernet) โทกเกนริง (Token ring) หรือเอฟดีดีไอ (FDDI) เป็นต้น

ประวัติความเป็นมา (Stevens ,1994) อีเทอร์เน็ตถูกคิดค้นขึ้นในปี ค.ศ.1982 โดย Digital Equipment Corp, Intel Corp. และ Xerox Corp. เพื่อใช้วางกับเครือข่ายท้องถิ่น (Local area network) โดยการใช้เทคโนโลยีที่เรียกว่า ซีเอสเอ็มเอ/ซีดี (CSMA/CD : Carrier Sense Multiple Access/Collision Detection) ทำงานที่ 10 เมกะบิตต่อวินาที หลังจากนั้น IEEE ได้กำหนดมาตรฐาน 802.3 ซึ่งมีลักษณะคล้ายคลึงกับอีเทอร์เน็ตแต่มีรายละเอียดที่แตกต่างกันออกไป

ลักษณะของเฟรมของอีเทอร์เน็ตเป็นดังนี้

รูปที่ 2-3 เฟรมอีเทอร์เน็ต

บทที่ 3
เอฟทีพี

3.1 ลักษณะทั่วไป

เอฟทีพีเป็นโพรโตคอลสำหรับถ่ายโอนแฟ้มระหว่างคอมพิวเตอร์สองระบบ เอฟทีพีนั้นอาศัยโพรโตคอลทีซีพีในการลำเลียงข้อมูล คุณสมบัติของเอฟทีพีคือช่วยให้การโอนย้ายแฟ้มข้อมูลข้ามเครือข่ายได้โดยง่ายถึงแม้ว่าระบบแฟ้มของคอมพิวเตอร์ทั้งสองด้านนั้นจะแตกต่างกัน

การโอนย้ายแฟ้มข้อมูลข้ามระหว่างโฮสต์เป็นความต้องการหลักอย่างหนึ่ง ในการใช้งานระบบเครือข่ายทั่วไป การสำเนาแฟ้มขึ้นใหม่หรือการสำเนาแฟ้มข้ามไดเรกทอรีเป็นงานปกติที่ทำได้โดยตรงและเรียบง่าย แต่การสำเนาแฟ้มข้ามเครื่องผ่านระบบเครือข่ายมีข้อระมัดระวังและต้องคำนึงหลายประการ ทั้งนี้เพราะโฮสต์ที่ทำงานภายใต้ระบบปฏิบัติการที่ต่างกันมักมีระบบจัดเก็บแฟ้มที่แตกต่างกัน นับตั้งแต่วิธีตั้งชื่อแฟ้มและไดเรกทอรี วิธีเข้าสู่ไดเรกทอรีย่อย หรือข้อกำหนดเพื่อป้องกันการใช้แฟ้มโดยไม่ได้รับอนุญาตเป็นต้น

ภายใต้สภาวะที่ปัจจุบันมีระบบปฏิบัติการที่แตกต่างกันเป็นจำนวนมาก การสร้างระบบปฏิบัติการหนึ่งๆ ให้รู้จักกับวิธีการจัดเก็บแฟ้มข้อมูลของระบบปฏิบัติการอื่นทั้งหมดเป็นสิ่งที่ทำได้ยาก ผู้ออกแบบทีซีพี/ไอพีแก้ปัญหานี้โดยสร้างโพรโตคอลประยุกต์เอฟทีพี (FTP : File Transfer Protocol) เพื่อทำหน้าที่เป็นตัวกลางแลกเปลี่ยนคำสั่ง เอฟทีพีแต่ละด้านจะเชื่อมโยงเข้ากับระบบแฟ้มข้อมูลของตนเองและทำงานตามชุดคำสั่งมาตรฐานทำให้โฮสต์หนึ่งไม่ต้องทราบว่าอีกโฮสต์หนึ่งนั้นมีการจัดเก็บข้อมูลอย่างไร

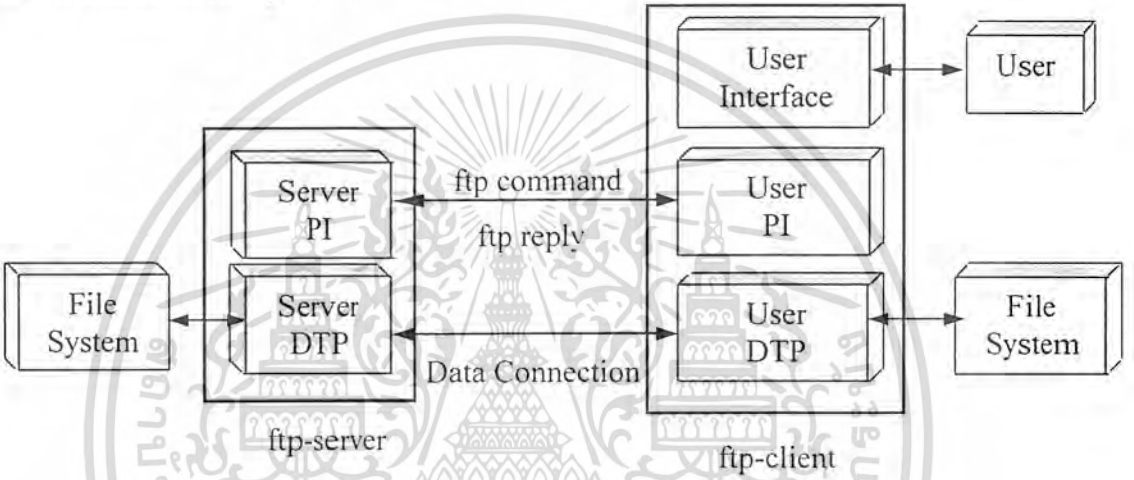
รูปที่ 3-1 การแปลงคำสั่งเอฟทีพี

จากรูปที่ 3-1 แสดงตัวอย่างวินโดวส์ไคลเอ็นขอบริการโอนย้ายด้วยเอฟทีพีจากยูนิกซ์เซิร์ฟเวอร์ เมื่อผู้ใช้ป้อนคำสั่ง dir คำสั่งจะถูกส่งไปยังเซิร์ฟเวอร์และแปลงไปสู่คำสั่ง ls การใช้เอฟทีพีจึงช่วยการจัดการไฟล์ข้ามเครือข่ายได้โดยง่าย และกำจัดความไม่เข้ากันในการจัดเก็บแฟ้มของระบบปฏิบัติการออกไป

3.2 แบบจำลองของเอฟทีพี

อาร์เอฟซี 959 นิยามแบบจำลองการทำงานของเอฟทีพีซึ่งทำงานตามแบบไคลเอ็นต์-เซิร์ฟเวอร์ เอฟทีพีเป็นโพรโตคอลที่ค่อยข้างซับซ้อน เนื่องจากมีการสถาปนาทีซีพีขึ้นสองส่วนคือ ส่วนรับส่งคำสั่ง และส่วนโอนย้ายข้อมูล

เอฟทีพีแต่ละด้านจะมีตัวแปลโพรโตคอล (Protocol Interpreter : PI) ต่อเชื่อมผ่านการเชื่อมต่อส่วนควบคุม (Control Connection : CC) ซึ่งเป็นช่องทางส่งคำสั่ง ตัวแปลโพรโตคอลมีหน้าที่แปลคำสั่งและดำเนินการตามคำสั่งนั้น ส่วนเชื่อมต่อที่สองคือ การเชื่อมต่อส่วนข้อมูล (Data Connection :DC) เป็นช่องทางส่งข้อมูลซึ่งอยู่ภายใต้การดูแลของโมดูลโอนย้ายข้อมูล (Data Tranfer Process : DTP) แบบจำลองของเอฟทีพีนั้นแสดงได้ดังรูปที่ 3-2



รูปที่ 3-2 แบบจำลองการทำงานของเอฟทีพี

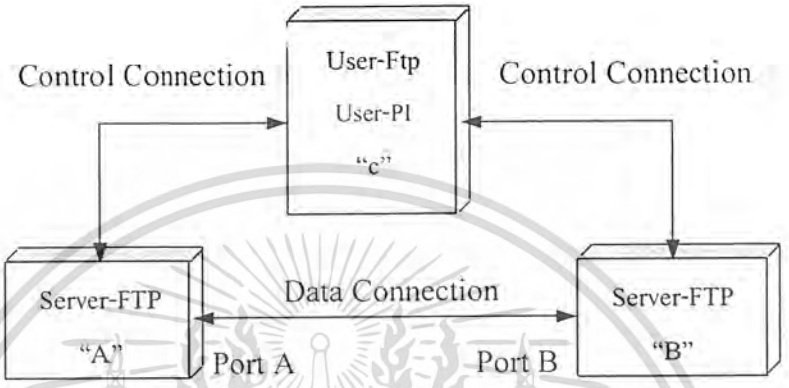
ข้อสังเกต 1. การเชื่อมต่อส่วนข้อมูล (data connection) นั้นอาจจะไม่ถูกใช้แบบในรูปที่แสดงนี้ได้

2. การเชื่อมต่อส่วนข้อมูล (data connection) ไม่จำเป็นจะต้องมีอยู่ตลอดเวลา

จากแบบจำลองที่แสดงดังรูปที่ 3-2 ตัวแปลโพรโตคอลจะเป็นส่วนที่เริ่มสร้างการเชื่อมต่อส่วนควบคุม (control connection) ซึ่งในส่วนนี้จะมีการทำงานเหมือนกับโพรโตคอลเทลเน็ต ตอนเริ่มต้นนั้นคำสั่งเอฟทีพีมาตรฐานจะถูกสร้างโดยยูสเซอร์-พีไอ และส่งไปยังเซิร์ฟเวอร์ผ่านการเชื่อมต่อส่วนควบคุม การตอบกลับมาตรฐานนั้นถูกส่งมาจากเซิร์ฟเวอร์-พีไอมาหายูสเซอร์-พีไอ เป็นการตอบรับของคำสั่งที่ส่งไป

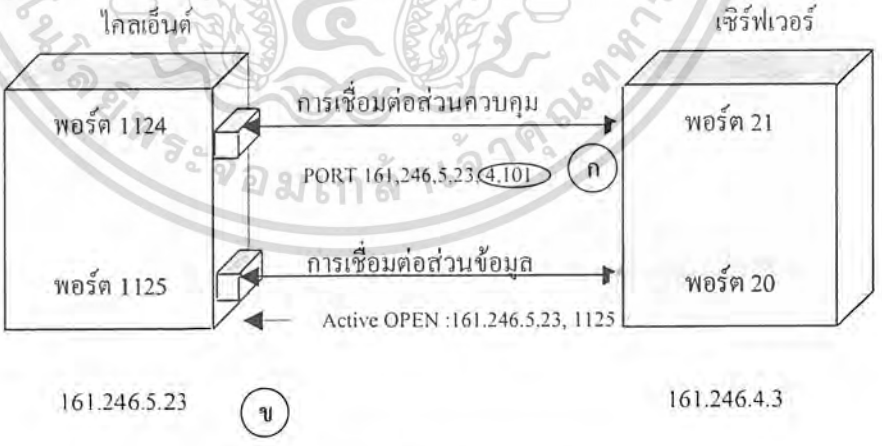
คำสั่งของเอฟทีพีจะระบุพารามิเตอร์สำหรับการเชื่อมต่อส่วนข้อมูลไปด้วย (พอร์ตข้อมูล, โหมดของการโอนย้ายข้อมูล, ประเภทของข้อมูล, โครงสร้าง) และลักษณะของการจัดการระบบไฟล์ ยูสเซอร์-ดีทีพี ควรจะรอรับข้อมูลบนพอร์ตข้อมูลที่ได้ระบุมา และเซิร์ฟเวอร์ก็จะเริ่มสร้างการเชื่อมต่อส่วนข้อมูลและการโอนย้ายข้อมูลด้วยพารามิเตอร์ที่ตรงกันกับฝั่งผู้ใช้ เป็นข้อสังเกตว่าพอร์ตข้อมูลนั้นไม่จำเป็นต้องอยู่ในโฮสต์เดียวกับที่เริ่มสร้างคำสั่งเอฟทีพี แต่เพียงผู้ใช้หรือยูสเซอร์-เอฟทีพีต้องแน่ใจว่าได้ 'รอรับข้อมูล' บนพอร์ตข้อมูลที่ได้ระบุไว้ และต้องเป็นที่สังเกตด้วยเหมือนกันว่าการเชื่อมต่อส่วนข้อมูลนั้นจะถูกใช้สำหรับการส่งและการรับข้อมูล

ในอีกสถานการณ์หนึ่งผู้ใช้คนหนึ่งต้องการที่จะส่งไฟล์ระหว่างโฮสต์ 2 แห่ง ซึ่งทั้งสองนั้นไม่ใช่โลคอลโฮสต์ ผู้ใช้คนนั้นต้องทำการสร้างการเชื่อมต่อส่วนควบคุม (Control Connection) ไปยังเซิร์ฟเวอร์ทั้งสอง ต่อจากนั้นก็จัดการส่วนการเชื่อมต่อข้อมูลให้กับโฮสต์ทั้ง 2 ในการจัดการส่วนนี้ ข้อมูลในส่วนการเชื่อมต่อส่วนควบคุมจะถูกส่งไปยังยูสเซอร์-พีไอ แต่ข้อมูลจะถูกส่งระหว่างกระบวนการโอนย้ายข้อมูล จากโมเดลนี้จะเป็นการติดต่อกันระหว่างเซิร์ฟเวอร์



รูปที่ 3-3 แสดงการโอนย้ายข้อมูลระหว่างเซิร์ฟเวอร์

โพรโตคอลต้องการให้ การเชื่อมต่อส่วนควบคุมถูกเปิดอยู่ในระหว่างการส่งข้อมูลกำลังดำเนินอยู่ มันเป็นหน้าที่ของผู้ใช้ที่จะต้องร้องขอเพื่อที่จะปิดการเชื่อมต่อในส่วนนี้เมื่อเสร็จสิ้นการใช้บริการของโปรแกรมเอฟทีพี ในขณะที่เซิร์ฟเวอร์ที่กำลังทำงานอยู่นั้น เซิร์ฟเวอร์ควรจะต้องยกเลิกการส่งข้อมูลถ้าการเชื่อมต่อถูกปิด โดยไม่ต้องมีคำสั่งใดๆ



รูปที่ 3-4 การสร้างส่วนเชื่อมต่อข้อมูล

จากรูปแสดงการสร้างส่วนเชื่อมต่อข้อมูลระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ ในที่นี้สมมติให้พอร์ตประจำการเชื่อมต่อส่วนควบคุมของไคลเอ็นคือ 1124 และเตรียมพอร์ต 1125 รอไว้สำหรับการเชื่อมต่อส่วนข้อมูล

ไคลเอ็นต์จะขอเปิดการเชื่อมต่อส่วนข้อมูลตามตำแหน่ง ก. โดยส่งรหัสคำสั่ง PORT ตามด้วยอาร์กิวเมนต์หกตัวแยกด้วยจุลภาคคือไอพีแอดเดรส (161,246,5,23) และหมายเลขพอร์ต 4 , 101 ซึ่งแสดงถึงพอร์ต 1125 (เลขพอร์ตเป็นรหัส 16 บิต สองชุดติดกัน ดังนั้นตัวเลข 4 , 101 คือ 4*256 +101 = 1125)

ต่อจากนั้นเซิร์ฟเวอร์จะสร้างการเชื่อมต่อจากพอร์ต 20 ไปยังไคลเอ็นต์ที่พอร์ต 1125 ตามตำแหน่ง ข.

3.3 การแทนข้อมูล (DATA REPRESENTATION)

เอฟทีพีมีรูปแบบการแทนข้อมูลเพื่อกำหนดวิธีการโอนย้ายและการจัดเก็บข้อมูล 4 ประเภทคือ ชนิดแฟ้มข้อมูล , ฟอร์แมตควบคุม, โครงสร้างข้อมูล , วิธีการส่งข้อมูล

3.3.1 ชนิดข้อมูล (Data Type)

คุณสมบัติที่สำคัญของเอฟทีพี คือความสามารถในการแปลงประเภทของข้อมูลให้ถูกต้องตามความเหมาะสม มีประโยชน์กรณีที่เครื่องเซิร์ฟเวอร์และเครื่องไคลเอนต์เป็นคอมพิวเตอร์ที่ต่างระบบกัน ซึ่งเอฟทีพีก็จะแปลงข้อมูลให้ถูกต้องตามความเหมาะสมได้โดยจะมีคำสั่ง TYPE เพื่อกำหนดประเภทของข้อมูลซึ่งซึ่งประกอบด้วย

เอ็นวีที-แอสกี (NVT–ASCII (Network Virtual Terminal ASCII)) เป็นข้อมูลทั่ว ๆ ไปที่ใช้รหัสแอสกี

เอ็บซีดิค (EBCDIC) เป็นข้อมูลที่ใช้ในเครื่องคอมพิวเตอร์เมนเฟรมของไอบีเอ็ม

อิเมจ (IMAGE) เป็นข้อมูลที่เป็นไบนารี อาจเป็นโปรแกรมหรือรูปภาพโดยมีข้อมูล 8 บิต

แบบเฉพาะโฮสต์ (LOCAL) เป็นข้อมูลที่เป็นไบนารีเช่นเดียวกับอิเมจ แต่จำนวนบิตต่อหนึ่งไบต์อาจจะแตกต่างกันไปตามประเภทของเครื่องคอมพิวเตอร์

3.3.2 การควบคุมรูปแบบ (Format Control)

ไม่มีฟอร์แมต (NonPrint) ไม่บรรจุรหัสควบคุม

ฟอร์แมตเทลเน็ต (Telnet format control) มีรหัสควบคุมตามแบบเทลเน็ตสำหรับกำหนดการทำงานเครื่องพิมพ์

ฟอร์แมตฟอร์แทรน (Fortran carriage control) แบบภาษาฟอร์แทรน

3.3.3 โครงสร้างข้อมูล (Data Structure)

เอฟทีพีสามารถกำหนดโครงสร้างของไฟล์ได้หลายแบบ เพื่อรองรับการใช้งานร่วมกับเครื่องคอมพิวเตอร์ชนิดที่ต่างกัน โดยแยกเป็น 3 ประเภทได้ดังนี้

แฟ้ม (file) ไม่มีโครงสร้างพิเศษใด ข้อมูลถือว่าเป็นไบต์เรียงต่อกัน

เรคอร์ด (record) ข้อมูลจัดแบ่งออกเป็นเรคอร์ด

หน้า (page) ข้อมูลมีหมายเลขหน้ากำกับหน้า และสามารถถ่ายโอนโดยไม่ต้องเรียงลำดับหน้า

3.3.4 วิธีการรับส่ง (Transmission Mode)

สายข้อมูล (stream) เป็นวิธีที่จะรับส่งข้อมูลเรียงลำดับไบต์ส่งต่อกันไปเรื่อยๆ ดังนั้นการรับส่งวิธีนี้จึงสามารถใช้ได้กับไฟล์ทุกประเภท ส่วนไฟล์ที่มีโครงสร้างต้องมีรหัสตัวอักษรพิเศษที่กำหนดการสิ้นสุดเรคอร์ด (End of Record หรือ EOR) และจบไฟล์ (End of File หรือ EOF) ด้วย ซึ่งการรับส่งข้อมูลจะสิ้นสุดโดยการตรวจสอบจากค่าของ EOF นี้

บล็อก (block) เป็นโหมดการรับส่งข้อมูลที่เป็นบล็อก ในแต่ละบล็อกจะมีส่วนหัวที่ระบุขนาดและรายละเอียดต่างๆ ของบล็อก เช่น สถานะแสดงบล็อกสุดท้ายในไฟล์หรือในเรคอร์ด ขนาดบิตที่ใช้ตรวจสอบว่าข้อมูลที่รับส่งในบล็อกนั้นถูกต้องหรือไม่ รวมทั้งข้อมูลที่เรียกว่า Restart Marker ซึ่งเป็นข้อมูลที่ใช้อ้างอิงเพื่อโอนข้อมูลต่อจากข้อมูลของเดิมที่โอนไม่สำเร็จ

บีบอัด (compress) เป็นวิธีเพิ่มประสิทธิภาพการรับส่งข้อมูล โดยใช้เทคนิคการบีบอัดข้อมูลให้เล็กลง โดยมีวิธีการง่ายๆ คือ ข้อมูลซ้ำๆ ที่เรียงต่อกันจะถูกลดมาเหลือ 3 ไบต์ซึ่ง 2 ไบต์แรกจะบอกให้รู้ว่าข้อมูลชุดดังกล่าวมีค่าซ้ำจำนวนกี่ตัว

3.4 Error Recovery

ในการรับส่งไฟล์แบบสายข้อมูล (Stream Mode) การแก้ไขกรณีข้อมูลสูญหายระหว่างการรับส่ง จะใช้ความสามารถของโพรโตคอลทีซีพี แต่ไม่มีการส่งข้อมูลซ้ำ แต่ถ้าหากเป็นแบบบล็อกโหมดหรือโหมดบีบอัด ข้อมูลสามารถส่งใหม่เฉพาะส่วนที่ตรวจพบข้อผิดพลาดได้

3.5 คำสั่งและการตอบกลับของเอฟทีพี

3.5.1 คำสั่งของเอฟทีพี (FTP Command)

คำสั่งของเอฟทีพีใช้รหัสแอสกีเอ็นวีทีเช่นเดียวกับเทลเน็ต ซึ่งจะปิดท้ายด้วย CR LF ชื่อคำสั่งในเอฟทีพีอยู่ในรูปรหัสอักขระตัวใหญ่ บางคำสั่งอาจมีอาร์กิวเมนต์เป็นค่าเผื่อเลือกประกอบอยู่ด้วย จำนวนคำสั่งในเอฟทีพีมีมากกว่า 30 คำสั่ง ซึ่งแบบออกเป็นประเภทได้ดังนี้

3.5.1.1 คำสั่งควบคุมการเข้าถึง (ACCESS CONTROL COMMAND)

คำสั่งดังต่อไปนี้จะเป็นรายละเอียดของคำสั่งที่ใช้ในการควบคุมการเข้าถึง

USER NAME (USER) ตามด้วยพารามิเตอร์ที่บ่งบอกถึงชื่อผู้ใช้ในระบบเพื่อเป็นการตรวจสอบสิทธิ์ผู้ใช้ว่ามีผู้ใช้ในระบบหรือไม่

PASSWORD (PASS) ตามด้วยพารามิเตอร์ที่บ่งบอกถึงรหัสลับของผู้ใช้ในระบบ เป็นการพิสูจน์สิทธิ์ของผู้ใช้ในระบบ

ACCOUNT (ACCT) เป็นวิธีการเข้าใช้งานอีกรูปแบบหนึ่งนอกเหนือจากการใช้คำสั่ง USER

CHANGE WORKING DIRECTORY (CWD) เป็นคำสั่งที่ใช้ในการเปลี่ยนไดเรกทอรีปัจจุบัน

CHANGE TO PARENT DIRECTORY (CDUP) เป็นคำสั่งที่ใช้เพื่อเข้าถึงไดเรกทอรีที่อยู่ก่อนไดเรกทอรีปัจจุบัน

STRUCTURE MOUNT (SMNT) จะทำให้ผู้ใช้สามารถใช้ระบบอื่นที่ผูกติดอยู่ได้โดยไม่ต้องใส่ชื่อผู้ใช้ กับรหัสผ่านอีก

REINITIALIZE (REIN) เป็นการรีเซตค่าใหม่ทั้งหมดแต่ไม่หยุดการโอนย้ายข้อมูลของการเชื่อมต่อส่วนข้อมูล

LOGOUT (QUIT) เป็นการยกเลิกการติดต่อ

3.5.1.2 คำสั่งพารามิเตอร์การโอนย้าย (Transfer Parameter Commands)

ค่าพารามิเตอร์ในการโอนย้ายข้อมูลทั้งหมดนั้นจะต้องมีค่าดิฟอลท์ และคำสั่งจะระบุค่าพารามิเตอร์การโอนย้ายข้อมูลก็ต่อเมื่อค่าพารามิเตอร์ดิฟอลท์มีการเปลี่ยนแปลง ค่าดิฟอลท์อาจจะเป็นค่าสุดท้ายที่มีการระบุไว้หรือถ้าไม่ได้มีการระบุไว้ ค่าดิฟอลท์มาตรฐานยังคงเป็นค่าเดิม

DATA PORT (PORT) จะส่งพารามิเตอร์ (แอดเดรสกับพอร์ตที่เปิดรอการติดต่อ)ไปยังเซิร์ฟเวอร์

PASSIVE (PASV) จะสั่งให้เซิร์ฟเวอร์เปลี่ยนไปอยู่ในโหมดพาสซีฟ

REPRESENTATION TYPE (TYPE) ใช้สำหรับตั้งค่าการรับส่งแฟ้มข้อมูลมี 2 โหมด คือโหมดแอสกีกับโหมดไบนารี

FILE STRUCTURE (STRU) เป็นการส่งไฟล์ตามลักษณะของไฟล์ที่จะส่งมีอยู่ 3 ชนิด คือ แฟ้ม, เรคอร์ด, เพจ โดยปกติจะถูกตั้งค่าเป็นแฟ้ม

TRANSFER MODE (MODE) เป็นการส่งไฟล์ที่มีหลายรูปแบบในการส่ง มีรูปแบบในการส่งอยู่ 3 ชนิด คือ สายข้อมูล, บล็อก, บีบอัด โดยปกติจะถูกตั้งค่าเป็นสายข้อมูล

3.5.1.3 คำสั่งใช้บริการของเอฟทีพี (FTP Service Commands)

คำสั่งที่ใช้บริการของเอฟทีพีจะอธิบายการโอนย้ายไฟล์ หรือการร้องขอจากผู้ใช้ ส่วนใหญ่แล้วการระบุค่าอาร์กิวเมนต์ของคำสั่งจะเป็นชื่อพาธ

RETRIEVE (RETR) ใช้เมื่อต้องการไฟล์ที่อยู่บนเซิร์ฟเวอร์

STORE (STOR) ใช้เมื่อต้องการเก็บไฟล์ไว้ในบนเซิร์ฟเวอร์

STORE UNIQUE (STOU) เป็นคำสั่งใช้เหมือนกัน STOR แต่จะสร้างเก็บไว้ในไดเรกทอรีที่มีชื่อเดียวกับไฟล์นั้น

APPEND (with create) (APPE) ใช้เมื่อต้องการส่งไฟล์ไปต่อกับไฟล์ที่มีอยู่แล้ว

ALLOCATE (ALLO) เป็นคำสั่งไว้สำหรับจองพื้นที่บนเซิร์ฟเวอร์ เพื่อเก็บไฟล์

RESTART (REST) คำสั่งนี้จะส่งค่าตำแหน่งของไฟล์ที่ต้องการจะดาวน์โหลดมา

RENAME FROM (RNFR) จะเปลี่ยนชื่อไฟล์โดยระบุชื่อไฟล์ที่จะเปลี่ยน โดยจะทำหลังคำสั่ง RNTO

RENAME TO (RNTO) จะเปลี่ยนเป็นชื่อไฟล์ใหม่ โดยจะทำก่อนคำสั่ง RNFR

ABORT (ABOR) เป็นคำสั่งยกเลิกการติดต่อ

DELETE (DELE) เป็นคำสั่งลบไฟล์ที่อยู่บนเซิร์ฟเวอร์

REMOVE DIRECTORY (RMD) เป็นคำสั่งลบไดเรกทอรีที่อยู่บน เซิร์ฟเวอร์

MAKE DIRECTORY (MKD) เป็นคำสั่งสร้างไดเรกทอรีบนเซิร์ฟเวอร์

PRINT WORKING DIRECTORY (PWD) แสดงไดเรกทอรีปัจจุบันบน เซิร์ฟเวอร์

LIST (LIST) แสดงรายชื่อไฟล์ที่อยู่บนเซิร์ฟเวอร์ในไดเรกทอรีปัจจุบัน

NAME LIST (NLST) แสดงรายชื่อไฟล์แบบมีรายละเอียดที่อยู่บนเซิร์ฟเวอร์ ในไดเรกทอรีปัจจุบัน

SITE PARAMETERS (SITE) เป็นการส่งพารามิเตอร์ที่เป็นคำสั่งที่สร้างขึ้นเฉพาะ

SYSTEM (SYST) เป็นคำสั่งที่จะแสดงชนิดของระบบปฏิบัติการที่เป็นเซิร์ฟเวอร์เอฟทีพี

STATUS (STAT) จะแสดงสถานะทั้งหมดที่เกี่ยวข้องกับการรับส่งแฟ้มข้อมูล

HELP (HELP) แสดงวิธีการใช้ของคำสั่งที่ต้องการจะใช้

NOOP (NOOP) ไม่มีผลอะไรกับเซิร์ฟเวอร์แต่ส่งไปเพื่อไม่ให้เซิร์ฟเวอร์ตัดการติดต่อ

3.5.2 การตอบกลับของเอฟทีพี (FTP REPLIES)

3.5.2.1 รหัสการตอบกลับ

การตอบกลับของเอฟทีพีประกอบด้วยรหัสตัวเลขแอสกี 3 หลัก พร้อมด้วยคำอธิบายซึ่งอาจมีหรือไม่ก็ได้ ซอฟต์แวร์ด้านไคลเอ็นต์เพียงแต่นำรหัสตัวเลขไปตรวจสอบการทำงาน ส่วนข้อความนั้นมีเพื่อให้ผู้ใช้เข้าใจถึงผลการทำงานด้วย แต่ละหลักของตัวเลข 3 หลัก จะมีความหมายเฉพาะตัว

ในตัวเลขตัวแรก จะแสดงการตอบกลับ ว่า ดี, ไม่ดี หรือว่าไม่สมบูรณ์ จะประกอบด้วยตัวเลข 0 ถึง 5 ซึ่งมีความหมายดังนี้

1 การร้องขอการกระทำกำลังเกิดขึ้น (initiated)

2 ตอบสนองการร้องขอเสร็จเรียบร้อย , เซิร์ฟเวอร์พร้อมที่จะรับการร้องขอครั้งใหม่

3 ได้รับคำสั่งแล้ว , แต่ในการร้องขอการปฏิบัติการต้องการข้อมูลมากกว่านี้

4 ไม่ยอมรับคำสั่ง , การร้องขอการปฏิบัติล้มเหลว , แต่เงื่อนไขที่ล้มเหลวสามารถที่จะร้องขอได้อีกครั้ง

5 ไม่ยอมรับคำสั่ง , การร้องขอการปฏิบัติล้มเหลว ,แต่เงื่อนไขที่ล้มเหลวมีท่าทางว่าจะกลับมาเกิดอีกครั้ง

ตัวเลขในหลักที่สองแสดงข้อมูลในส่วนการทำงาน ได้แก่

0 Syntax แสดงถึง ไวยากรณ์ผิดพลาด (syntax error)

1 Information แสดงถึงการร้องตอบการร้องขอข้อมูล

2 Connection หมายความว่าข้อความนั้นได้มีการตอบกลับสำหรับการร้องขอในการควบคุมและการรับส่งข้อมูล

3 Authentication and accounting แสดงถึงการล็อกอินหรือมีการปฏิบัติการ

4 ยังไม่ระบุการใช้งาน

5 File System แสดงข้อมูลเกี่ยวกับสถานะของระบบไฟล์ของเซิร์ฟเวอร์

สำหรับเลขหลักที่สาม เป็นตัวกำหนดความหมายเพิ่มเติมโดยเฉพาะกับรหัสการตอบกลับนั้น ๆ

ตัวอย่างรหัสและการตอบกลับ

110 restart marker reply. โดย yyyy = mmmm เมื่อ yyy เป็น user process data stream marker และmmmm เป็น ftpd equivalent marker

120 พร้อมที่จะให้บริการใน nnn นาที

200 คำสั่งถูกต้อง

211 สถานะของระบบ หรือ reply help ของระบบ

212 สถานะของไดเรกทอรี

230 ยูสเซอร์ล็อคอิน

250 การร้องขอที่จะกระทำกับไฟล์ ถูกต้อง

331 ชื่อผู้ใช้ถูกต้อง ต้องการรหัสผ่าน

425 ไม่สามารถเปิดการเชื่อมต่อส่วนข้อมูล (Data Connection)

451 การร้องขอบริการถูกยกเลิก เกิดข้อผิดพลาดในการทำงาน

500 ไวยากรณ์ผิดพลาด (Syntax error) ไม่สามารถแปลความหมายได้หรือคำสั่งนั้นมีความยาวมากเกินไป

530 ไม่ได้ล็อคอิน

550 การร้องขอให้เกิดขึ้น, ไม่มีไฟล์นี้ปรากฏอยู่, ไม่พบ, ไม่สามารถเข้าถึงได้

สำหรับคำสั่งโดยทั่วไปแล้วเอฟทีพีจะตอบกลับเพียงบรรทัดเดียวโดยมีรหัสตอบกลับนำหน้า เช่น

226 การโอนย้ายสมบูรณ์

แต่ถ้ามีการตอบกลับหลายบรรทัด เอฟทีพีจะแสดงรหัสพร้อมทั้งกำกับรหัสบรรทัดแรกด้วยเครื่องหมาย " – " ตัวอย่างเช่นเมื่อใช้คำสั่ง HELP จะมีการตอบกลับหลายบรรทัด ดังนี้

```
220 diamond FTP server (Version 1.7.212.3 Wed Jul 14 10:24:05 GMT 1999) ready.
HELP
214-The following commands are recognized (* =>'s unimplemented).
  USER  PORT  STOR  MSAM*  RNTO  NLST  MKD   CDUP  PBSZ*
  PASS  PASV  APPE  MRSQ*  ABOR  SITE  XMKD  XCUP  PROT*
  ACCT* TYPE  MLFL* MRCP*  DELE  SYST  RMD   STOU  SIZE
  SMNT* STRU  MAIL* ALLO   CWD   STAT  XRMD  AUTH* MDTM
  REIN* MODE  MSND* REST   XCWD  HELP  PWD   ADAT*
  QUIT  RETR  MSOM* RNFR   LIST  NOOP  XPWD  CCC*
214 Direct comments to ftp-bugs@diamond.
```

รูปที่ 3-5 แสดงการตอบกลับแบบหลายบรรทัดของคำสั่ง HELP

3.5.3 ตัวอย่างแสดงรหัสคำสั่งเอฟทีพี

เพื่อแสดงการส่งคำสั่งและการตอบกลับของเอฟทีพี จะขอยกตัวอย่างการใช้เอฟทีพีที่เซิร์ฟเวอร์ของภาควิชาวิศวกรรมคอมพิวเตอร์

3.5.3.1 การเชื่อมต่อ

ตัวอย่างแรกเป็นการเชื่อมต่อไปยังเอฟทีพีเซิร์ฟเวอร์โดยใช้ออพชัน –d (debug) เพื่อให้ไคลเอ็นต์แสดงคำสั่งที่ใช้ติดต่อกับเซิร์ฟเวอร์ บรรทัดที่ขึ้นต้นด้วยสัญลักษณ์ "--->" หมายถึงไคลเอ็นต์ส่งคำสั่งไปเซิร์ฟเวอร์ และบรรทัดที่ขึ้นต้นด้วยตัวเลขคือรหัสที่เซิร์ฟเวอร์ตอบกลับ

```
root@olala05:/home/kor# ftp -d 161.246.4.3
Connected to 161.246.4.3.
220 diamond FTP server (Version 1.7.212.3 Wed Jul 14 10:24:05 GMT 1999) ready.
Name (161.246.4.3:kor): s0010088
---> USER s0010088
331 Password required for s0010088.
Password:
---> PASS XXXX
230 User s0010088 logged in.
---> SYST
215 UNIX Type: L8
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> dir .ssh
---> PORT 161,246,5,215,7,17
200 PORT command successful.
---> LIST .ssh
150 Opening ASCII mode data connection for /usr/bin/ls.
total 12
drwxr-xr-x      2 s0010088   std4d      1024 Jan 30 01:12 .
drwxrwxrwx     10 s0010088   std4d      1024 Mar 14 05:10 ..
-rw-r-----      1 s0010088   std4d        17 Jan 30 01:12 authorized_keys
-rw-r--r--      1 s0010088   std4d       665 Mar  5 01:27 known_hosts
-rw-------      1 s0010088   std4d      1024 Mar 15 16:32 prng_seed
```

```
-rw-------          1 s0010088   std4d       727 Jan 30 01:12 s0010088.pub
226 Transfer complete.
ftp> quit
```

รูปที่ 3-6 ตัวอย่างการทำงานของเอฟทีพี

ในตัวอย่างนี้จะส่งคำสั่ง SYST เพื่อสอบถามระบบของเซิร์ฟเวอร์

คำสั่ง dir เป็นคำสั่งขอดูรายละเอียดของไฟล์หรือไดเรกทอรี เนื่องจากการส่งรายชื่อไฟล์จะส่งทางการเชื่อมต่อส่วนข้อมูล ไคลเอ็นต์จึงขอเปิดการเชื่อมต่อโดยคำสั่ง PORT หลังจากนั้นจึงส่งคำสั่ง LIST เพื่อขอรายชื่อไฟล์ เมื่อต้องการยกเลิกการเชื่อมต่อโดยพิมพ์คำสั่ง quit ไคลเอ็นต์จะส่งคำสั่ง QUIT และเซิร์ฟเวอร์ตอบกลับด้วยรหัส 221 จากนั้นจึงปิดการเชื่อมต่อ

ในระหว่างที่มีการเชื่อมต่อนี้หากใช้คำสั่ง netstat ตรวจสอบสถานะของทีซีพีจะพบสถานะดังนี้คือ

```
bash-2.04$ netstat -n |grep 161.246.5.215
tcp        0      0 161.246.4.3.20      161.246.5.215.1809     ESTABLISHED
tcp        0      0 161.246.4.3.21      161.246.5.215.1808     ESTABLISHED
```

รูปที่ 3-7 แสดงการเชื่อมต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์

ผลจากคำสั่ง netstat แสดงให้เห็นถึงไคลเอ็นต์ 161.246.5.215 พอร์ต 1808 ต่อเชื่อมกับ 161.246.4.3 ที่พอร์ตเอฟทีพี 21 และเมื่อขอดูรายชื่อไฟล์จากตัวอย่างข้างต้น (การโอนย้ายข้อมูล) ไคลเอ็นต์จะส่งคำสั่ง PORT 161,246,5,215,7,17 เพื่อสร้างส่วนเชื่อมต่อข้อมูลด้วยซ็อกเก็ต 161.246.5.215, 1809

3.5.3.2 การโอนย้าย

ตัวอย่างต่อไปนี้แสดงการโอนย้ายไฟล์ข้อมูลด้วยคำสั่ง get จะพบว่าที่ไคลเอ็นเริ่มติดต่อการกำหนดแบบการถ่ายโอนแบบไบนารี (I) และสร้างการเชื่อมโยงข้อมูลและส่งคำสั่ง RETR เพื่อขอแฟ้มข้อมูล Search.class

```
ftp> cd /home/std4d/s0010088/www/spider
---> CWD /home/std4d/s0010088/www/spider
250 CWD command successful.
ftp> dir
---> PORT 161,246,5,215,7,246
200 PORT command successful.
---> LIST
```

```
150 Opening ASCII mode data connection for /usr/bin/ls.
total 60
-rwxr-xr-x   1 s0010088   std4d       2090 Feb 20 00:23 MakeIndex.class
-rwxr-xr-x   1 s0010088   std4d       2692 Feb 20 00:23 MakeIndex.java
-rwxr-xr-x   1 s0010088   std4d       2929 Feb 20 00:23 Search.class
-rwxr-xr-x   1 s0010088   std4d        600 Feb 20 00:23 Search.html
-rwxr-xr-x   1 s0010088   std4d       4443 Feb 20 00:23 Search.java
-rwxr-xr-x   1 s0010088   std4d       2160 Feb 20 00:23 Spider.class
-rwxr-xr-x   1 s0010088   std4d       5334 Feb 20 00:23 Spider.java
-rwxr-xr-x   1 s0010088   std4d       1356 Feb 20 00:23 WebSpider.class
-rwxr-xr-x   1 s0010088   std4d       3867 Feb 20 00:23 WebSpider.java
226 Transfer complete.
ftp> get Search.class
local: Search.class remote: Search.class
---> TYPE I
200 Type set to I.
---> PORT 161,246,5,215,7,247
200 PORT command successful.
---> RETR Search.class
150 Opening BINARY mode data connection for Search.class (2929 bytes).
226 Transfer complete.
2929 bytes received in 0.142 secs (20 Kbytes/sec)
ftp>
```

รูปที่ 3-8 แสดงการโอนย้ายข้อมูลในเอฟทีพี

3.5.3.3 การขัดจังหวะ

ระหว่างการโอนย้ายหากผู้ใช้ขัดจังหวะด้วยการกด Ctrl-C การยกเลิกจะเกิดขึ้น อย่างไรนั้นขึ้นอยู่กับทิศทางการถ่ายโอน หากไคลเอ็นต์ถ่ายโอนข้อมูลไปยังเซิร์ฟเวอร์การขัดจังหวะย่อมทำได้ง่ายเนื่องจากไคลเอ็นต์เพียงแต่หยุดส่งข้อมูลและส่งคำสั่ง ABOR ไปยังเซิร์ฟเวอร์ แต่ในกรณีการขัดจังหวะเมื่อโอนย้ายข้อมูลจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์จะมีขั้นตอนเพิ่มขึ้น เนื่องจากไคลเอ็นต์ต้องส่งคำสั่งไปบอกให้เซิร์ฟเวอร์หยุดการโอนย้าย

ตัวอย่างข้างล่างนี้แสดงการขัดจังหวะการโอนย้ายจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์เมื่อกดปุ่ม Ctrl-C ไคลเอ็นต์จะร้องขอการขัดจังหวะโดยส่งไอพีดาต้าแกรมเร่งด่วน (Urgent pointer) ด้วยคำสั่ง <IAC,IP, DM,A,B,O,R,\r\n> และรอการตอบกลับ เซิร์ฟเวอร์จะตอบด้วย 426 และ 226 แจ้งว่าได้รับแพ็กเก็ตเร่งด่วนแล้ว

```
ftp> get j2sdk-1_3_0-linux.rpm
---> PORT 161,246,4,3,7,19
200 PORT command successful.
---> RETR j2sdk-1_3_0-linux.rpm
150 Opening BINARY mode data connection for j2sdk-1_3_0-linux.rpm (25672625 bytes).
 receive  aborted
waiting for remote to finish abort
426 Transfer aborted. Data connection closed.
226 Abort successful
```

รูป 3-9 แสดงการขัดจังหวะการโอนย้ายจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์

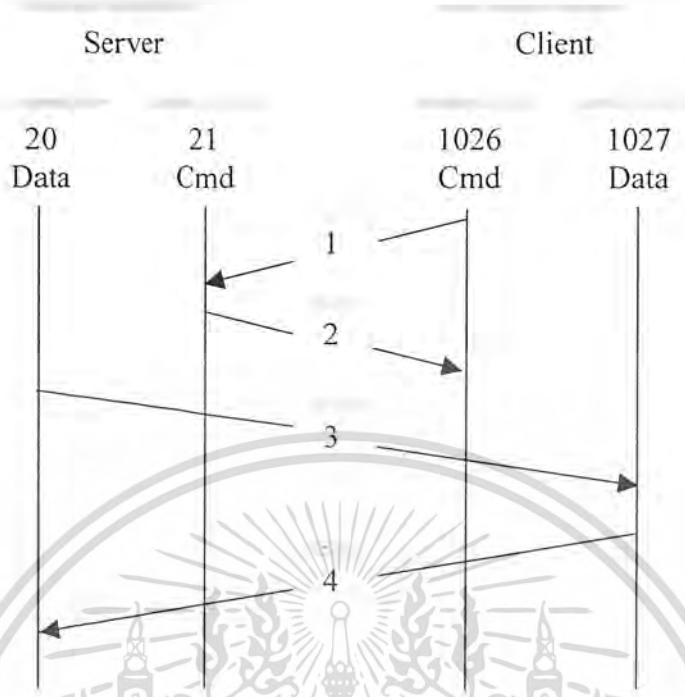
3.6 ลักษณะการทำงานของโปรแกรมเอฟทีพี

3.6.1 การทำงานแบบแอ็กทีฟ

ในการทำงานของเอฟทีพีแบบแอ็กทีฟ ฝั่งไคลเอ็นต์จะเป็นฝ่ายที่ไปสร้างการเชื่อมต่อจากค่าพอร์ตที่สุ่มขึ้นมา (N > 1024) ไปยังพอร์ต 21 ของเซิร์ฟเวอร์ ต่อจากนั้นจะเริ่มรอรับข้อมูลที่พอร์ต N+1 และส่งคำสั่งเอฟทีพี PORT N+1 ไปยังเอฟทีพีเซิร์ฟเวอร์ เครื่องเซิร์ฟเวอร์จะติดต่อเพื่อทำการเชื่อมต่อกลับไปยังพอร์ตของไคลเอ็นต์ที่ได้ระบุมากับพอร์ต 20 ของเซิร์ฟเวอร์

ไฟล์วอลล์ที่ฝั่งของเซิร์ฟเวอร์ที่จะสนับสนุนการทำงานแบบแอ็กทีฟจะต้องมีการเปิดช่องจากการติดต่อสื่อสารดังนี้

- การติดต่อมายังพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (การเริ่มการใช้งานของเครื่องฝั่งไคลเอ็นต์)
- การติดต่อจากพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์ไปยังพอร์ต N > 1024 (การติดต่อกลับไปยังพอร์ตในส่วนควบคุมของไคลเอ็นต์)
- การติดต่อจากพอร์ต 20 ของเอฟทีพีเซิร์ฟเวอร์ไปยังยังพอร์ต N > 1024 (การติดต่อไปยังพอร์ตในส่วนข้อมูลของไคลเอ็นต์เพื่อสร้างการเชื่อมต่อส่วนข้อมูล)
- การติดต่อมายังพอร์ต 20 ของเอฟทีพีเซิร์ฟเวอร์จากพอร์ต N > 1024 (การส่งสัญญาณ ACKs ไปยังพอร์ตข้อมูลของเซิร์ฟเวอร์)



รูปที่ 3-10 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบแอ็กทีฟ

จากรูปที่ 3-10 ในขั้นตอนที่ 1 เป็นการติดต่อกันจากพอร์ตควบคุมของไคลเอ็นต์กับพอร์ตส่วนควบคุมของเซิร์ฟเวอร์ และส่งคำสั่ง PORT 1027 แล้วเซิร์ฟเวอร์ก็ส่งสัญญาณ ACK ตอบกลับไปยังพอร์ตควบคุมของไคลเอ็นต์ ในขั้นที่ 2 ต่อมาในขั้นตอนที่ 3 เซิร์ฟเวอร์จะสร้างเชื่อมต่อจากพอร์ตข้อมูลของเซิร์ฟเวอร์ไปยังพอร์ตของไคลเอ็นต์ที่ได้ระบุไว้ในข้างต้น สุดท้ายไคลเอ็นต์ก็จะส่งสัญญาณ ACK ตอบกลับมายังเซิร์ฟเวอร์เพื่อยืนยันการเชื่อมต่อ

ปัญหาหลักของการทำงานแบบแอ็กทีฟ คือการที่โปรแกรมเอฟทีพีของฝั่งไคลเอ็นต์นั้นไม่จำเป็นที่จะต้องสร้างการเชื่อมต่อจริงๆ ไปยังพอร์ตข้อมูลของเซิร์ฟเวอร์ มันเพียงแต่ไปบอกเซิร์ฟเวอร์ให้ทราบว่ามีพอร์ตอะไรที่กำลังรอรับข้อมูลอยู่ และเซิร์ฟเวอร์ก็จะติดต่อกลับมายังพอร์ตที่ระบุไว้ของเครื่องไคลเอ็นต์เอง แต่ถ้าไฟล์วอลล์อยู่ฝั่งของไคลเอ็นต์แล้ว จะมีการบล็อกการติดต่อจากภายนอกไปยังเครื่องไคลเอ็นต์ที่อยู่ภายใน

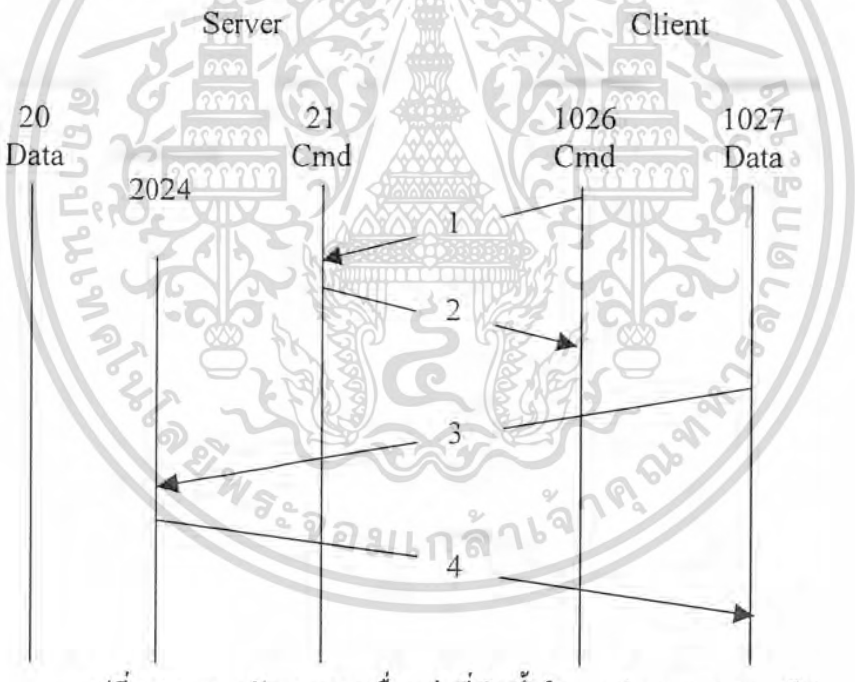
3.6.2 การทำงานแบบพาสซีฟ

ในการทำงานแบบพาสซีฟนี้ ไคลเอ็นต์จะเป็นฝ่ายที่เริ่มการติดต่อทั้ง 2 ส่วน (ส่วนควบคุมและส่วนข้อมูล) ซึ่งเป็นการแก้ปัญหาที่ไฟล์วอลล์จะบล็อกการติดต่อส่วนข้อมูลที่มาจากเซิร์ฟเวอร์ เมื่อมีเปิดการเชื่อมต่อของเอฟทีพีไคลเอ็นต์จะเปิด 2 พอร์ตที่สุ่มขึ้นมา (N > 1024 และ N+1) พอร์ตแรกจะใช้ติดต่อกับเซิร์ฟเวอร์บนพอร์ต 21 แต่แทนที่จะใช้คำสั่ง PORT ตามด้วยเลขพอร์ตข้อมูลที่จะให้เซิร์ฟเวอร์ติดต่อกลับมา ไคลเอ็นต์จะส่งคำสั่ง PASV ไปแทน ผลที่ได้ก็คือ เซิร์ฟเวอร์จะเปิดพอร์ต P > 1024 และส่งคำสั่ง

PORT P กลับมายังไคลเอ็นต์ และฝั่งไคลเอ็นต์ก็จะติดต่อจากพอร์ต N+1 ไปยังพอร์ต P ของเครื่องเซิร์ฟเวอร์เพื่อการโอนย้ายข้อมูล

ไฟล์วอลล์ที่ฝั่งของเซิร์ฟเวอร์ที่จะสนับสนุนการทำงานแบบพาสซีฟจะต้องมีการเปิดช่องจากการติดต่อสื่อสารดังนี้

- การติดต่อมายังพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (การเริ่มการใช้งานของเครื่องฝั่งไคลเอ็นต์)
- การติดต่อจากพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์ไปยังพอร์ต N > 1024 (การติดต่อไปยังพอร์ตในส่วนควบคุมของไคลเอ็นต์)
- การติดต่อมายังพอร์ต P > 1024 ของเอฟทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (ไคลเอ็นต์ติดต่อไปยังพอร์ตที่ระบุมาให้จากเซิร์ฟเวอร์เพื่อสร้างการเชื่อมต่อส่วนข้อมูล)
- การติดต่อจากพอร์ต P > 1024 ของเอฟทีพีเซิร์ฟเวอร์ไปยังพอร์ต N > 1024 (การส่งสัญญาณ ACKs ไปยังพอร์ตข้อมูลของไคลเอ็นต์)



รูปที่ 3-11 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบพาสซีฟ

ในขั้นตอนแรกนั้น ฝั่งไคลเอ็นต์จะติดต่อไปยังเซิร์ฟเวอร์ผ่านพอร์ตส่วนควบคุมและส่งคำสั่ง PASV ไป เซิร์ฟเวอร์จะตอบกลับในขั้นตอนที่ 2 ด้วยคำสั่ง PORT 2024 เป็นการบอกให้ไคลเอ็นต์ทราบถึงพอร์ตที่ได้เปิดไว้รอรับข้อมูล สำหรับการเชื่อมต่อส่วนข้อมูล ในขั้นที่ 3 ไคลเอ็นต์จะสร้างการเชื่อมต่อจากพอร์ตของไคลเอ็นต์ไปยังพอร์ตที่ได้ระบุมาของเซิร์ฟเวอร์ ในตอนสุดท้ายเซิร์ฟเวอร์ก็จะส่งสัญญาณ ACK กลับมายังพอร์ตข้อมูลของไคลเอ็นต์เพื่อยืนยันการเชื่อมต่อ

บทที่ 4

การเข้ารหัสและการคำนวณ

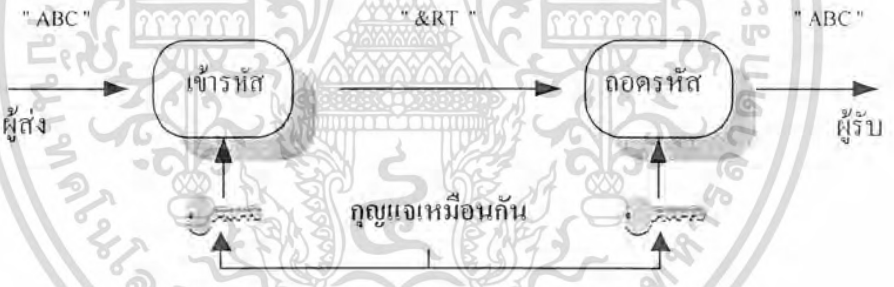
4.1 ระบบของเข้ารหัสข้อมูล (Cryptography System)

การเข้ารหัสข้อมูลเป็นการทำให้ข้อมูลที่ต้องการเป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูลปัจจุบัน โดยอาศัยหลักการของการเข้ารหัส (Encryption) และการถอดรหัส (Decryption)

- การเข้ารหัสเป็นการเปลี่ยนรูปข้อมูล โดยผ่านรูปแบบและกระบวนการแปรรูปข้อมูล ทำให้ข้อมูลที่ส่งมีรูปแบบที่ไม่เหมือนเดิม เพื่อทำให้ข้อมูลเป็นความลับ
- การถอดรหัสเป็นการแปลงข้อมูลที่ได้เข้ารหัสให้กลับมาเป็นข้อมูลเดิม ซึ่งการเข้ารหัสและการถอดรหัสจะถูกควบคุมโดยกุญแจหรือที่เรียกว่าคีย์ (Key)

4.1.1 ระบบการเข้ารหัสโดยใช้กุญแจเดียว (symmetric key)

การเข้ารหัสข้อมูลระบบนี้จะทั้งผู้รับและผู้ส่งจะต้องมีกุญแจที่เป็นความลับ ที่เหมือนกันในการเข้าและถอดรหัสข้อมูล ซึ่งหากกุญแจที่มีต่างกัน ก็จะทำให้ข้อมูลที่สื่อสารกันผิดพลาด ตัวอย่างการเข้ารหัสระบบนี้ได้แก่ การเข้ารหัสแบบ DES , 3DES , IDEA เป็นต้น



รูปที่ 4-1 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว

4.1.2 ระบบการเข้ารหัสแบบกุญแจสาธารณะ

การเข้ารหัสข้อมูลระบบนี้จะประกอบด้วยกุญแจ 2 อัน คือ

1. กุญแจส่วนตัว (private key) เป็นกุญแจที่จะต้องเก็บเป็นความลับ

2. กุญแจสาธารณะ (public key) เป็นกุญแจที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งกุญแจทั้งสองนี้จะเป็นกุญแจที่ต่างกัน การมีวิธีในการเข้าและถอดรหัส ดังรูป

รูปที่ 4-2 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ

4.2 รูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพี

สำหรับรูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพี แบ่งออกเป็น 4 ระดับ ได้แก่

- การเข้ารหัสระดับการเชื่อมโยงข้อมูล (link level encryption)
- การเข้ารหัสระดับเครือข่าย (network level encryption)
- การเข้ารหัสระดับทรานสปอร์ต (transport level encryption)
- การเข้ารหัสระดับโปรแกรมประยุกต์ (application level encryption)

4.2.1 การเข้ารหัสระดับการเชื่อมโยงข้อมูล (link level encryption)

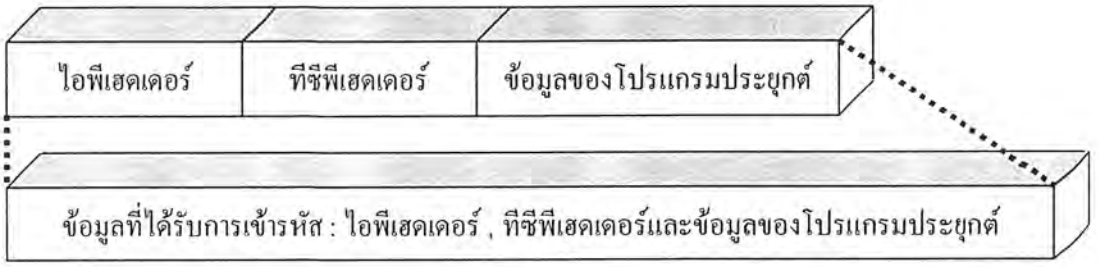
เป็นการเข้ารหัสในระดับดาต้าลิงก์ โดยทั่วไปมักจะเป็นการใช้อุปกรณ์พิเศษที่เรียกว่า กล่องสำหรับเข้ารหัส (encryption box) ซึ่งวิธีการนี้ถูกใช้ในระบบไซเฟอร์หรือการใช้ดีไวซ์ไดร์เวอร์

วิธีการนี้เป็นวิธีการที่ปลอดภัยที่สุด เนื่องจากข้อมูลถูกเข้ารหัสทั้งหมด นั่นคือทั้งข้อมูลที่ใช้งานและข้อมูลที่เป็นโครงสร้างของโพรโตคอล แต่ด้วยวิธีการนี้เป็นวิธีการที่ค่อนข้างมีค่าใช้จ่ายที่สูง เนื่องจากอุปกรณ์สำหรับการสื่อสารข้อมูลทั้งหมด จะต้องสนับสนุนการเข้ารหัสนี้ด้วย

รูปที่ 4-3 การเข้ารหัสในระดับการเชื่อมโยงข้อมูล

4.2.2 การเข้ารหัสระดับเครือข่าย (network level encryption)

เป็นการเข้ารหัสในไอพีแพ็กเก็ต (IP packet) ลักษณะดังภาพที่ 4-4



รูปที่ 4-4 การเข้ารหัสระดับเครือข่าย

4.2.3 การเข้ารหัสระดับทรานสปอร์ต (transport level encryption)

การเข้ารหัสระดับทรานสปอร์ตเป็นการเข้ารหัสในส่วนของทีซีพีเซกเมนต์ ซึ่งได้แก่ ทีซีพีเฮดเดอร์และ ข้อมูลของโปรแกรมประยุกต์

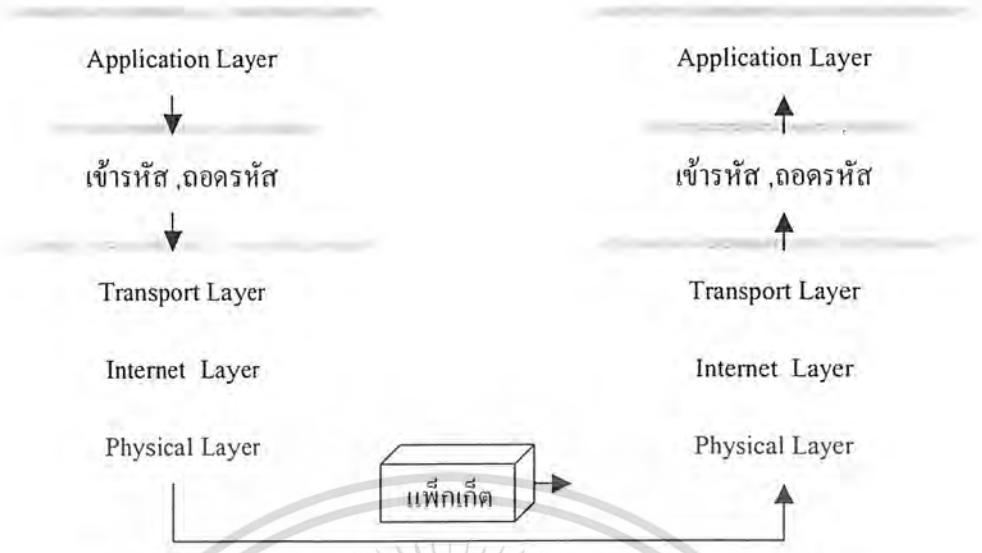
การเข้ารหัสในรูปแบบตามนี้ทำให้การส่งข้อมูลไปตามเครือข่ายสามารถทำได้โดยไม่จำเป็นต้องเปลี่ยนแปลงอุปกรณ์ใด เนื่องจากโครงสร้างในส่วนของไอพีไม่มีการเปลี่ยนแปลง

รูปที่ 4-5 การเข้ารหัสระดับทรานสปอร์ต

4.2.4 การเข้ารหัสระดับโปรแกรมประยุกต์ (application level encryption)

การเข้ารหัสระดับโปรแกรมประยุกต์ เป็นการเข้ารหัสในลักษณะของการเข้ารหัสแบบจุดต่อจุด (end-to-end encryption) นั่นคือ โปรแกรมประยุกต์จะเข้ารหัสในส่วนข้อมูลของตัวเองก่อนที่ส่งผ่านไปยังระดับล่าง ทำให้กระบวนการในระดับล่างไม่จำเป็นต้องมีการเปลี่ยนแปลงอะไร ซึ่งวิธีการนี้ถ้าต้องการนำมาใช้กับโปรแกรมประยุกต์เดิม เช่น เทลเนตหรือเอฟทีพี จะต้องแก้ไขโปรแกรมใหม่ทั้งโปรแกรมขอรับบริการและโปรแกรมให้บริการ

สำหรับการเข้ารหัสข้อมูลที่ใช้ในโครงงานนี้ การเข้ารหัสข้อมูลจะทำการเข้ารหัสข้อมูลในช่วงระหว่างชั้นแอปพลิเคชั่นและชั้นทรานสปอร์ตในระบบโอเอสไอโมเดลของโพรโตคอลทีซีพี/ไอพี ซึ่งวิธีการเข้ารหัสข้อมูลที่ใช้ คือ DES , 3DES (ระบบกุญแจเดียว) และ RSA (ระบบกุญแจสาธารณะ) ซึ่งรายละเอียดของการเข้ารหัสข้อมูลดังกล่าวจะมีดังต่อไปนี้



รูปที่ 4-6 แสดงการเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดลของทีซีพี/ไอพี

สำหรับการเข้ารหัสข้อมูลที่ใช้ในโครงงานนี้ จะใช้วิธีการเข้ารหัสข้อมูล คือ DES , 3DES (ระบบกุญแจเดียว) และ RSA (ระบบกุญแจสาธารณะ) เป็นหลักซึ่งรายละเอียดของการเข้ารหัสข้อมูลดังกล่าวจะมีดังต่อไปนี้

4.3 การเข้ารหัสแบบ DES (Data Encryption Standard)

4.3.1 ประวัติและที่มาของ DES

ในปลายทศวรรษที่ 1960 บริษัท IBM ได้จัดตั้งโครงงานวิจัยทางด้าน Computer Cryptography ซึ่งนำโดย Horst Feistel ซึ่งโครงงานนี้เสร็จสิ้นในปี 1971 ซึ่งผลงานวิจัยของโครงงานนี้คือ LUCIFER [FEIS73] โดยมีลักษณะเป็นการเข้ารหัสข้อมูลเป็นบล็อกขนาด 64 บิตและใช้คีย์ขนาด 128 บิต ซึ่งต่อมาได้ถูกพัฒนาขนาดของคีย์ให้ลดลงเหลือขนาด 56 บิต

โดยอัลกอริทึมของการเข้ารหัสข้อมูลของ Lucifer ได้ถูกพัฒนาโดย IBM สำหรับ NBS (NationalBureau of Standards) อัลกอริทึมนี้ได้เป็นที่รู้จักในนามของ DES (Data Encryption Standard) ถึงแม้ว่าชื่อจริงของมัน คือ DEA (Data Encryption Algorithm) ในสหรัฐ และ DEAI (Data Encryption Algorithm-1) ในอีกหลายๆ ประเทศ

4.3.2 รายละเอียดของ DES

เป็นวิธีการเข้ารหัสที่ใช้กันอย่างแพร่หลายที่เป็นพื้นฐานบน Data Encryption Standard (DES) ที่ได้พัฒนาขึ้นในปี 1977 โดย National Bureau of Standards ซึ่งปัจจุบันคือ Federal Information Processing Standard 46 (FIPS PUB46) สำหรับ DES ข้อมูลจะถูกเข้ารหัสเป็นบล็อกขนาด 64 บิตซึ่งใช้คีย์ขนาด 56 บิต โดยวิธีการจัดการกับข้อมูล 64 บิตที่เข้ามาเพื่อแปลงเป็น 64 บิตเข้ามูลออกไป และใช้คีย์ตัวเดียวกันนี้ในการถอดรหัส

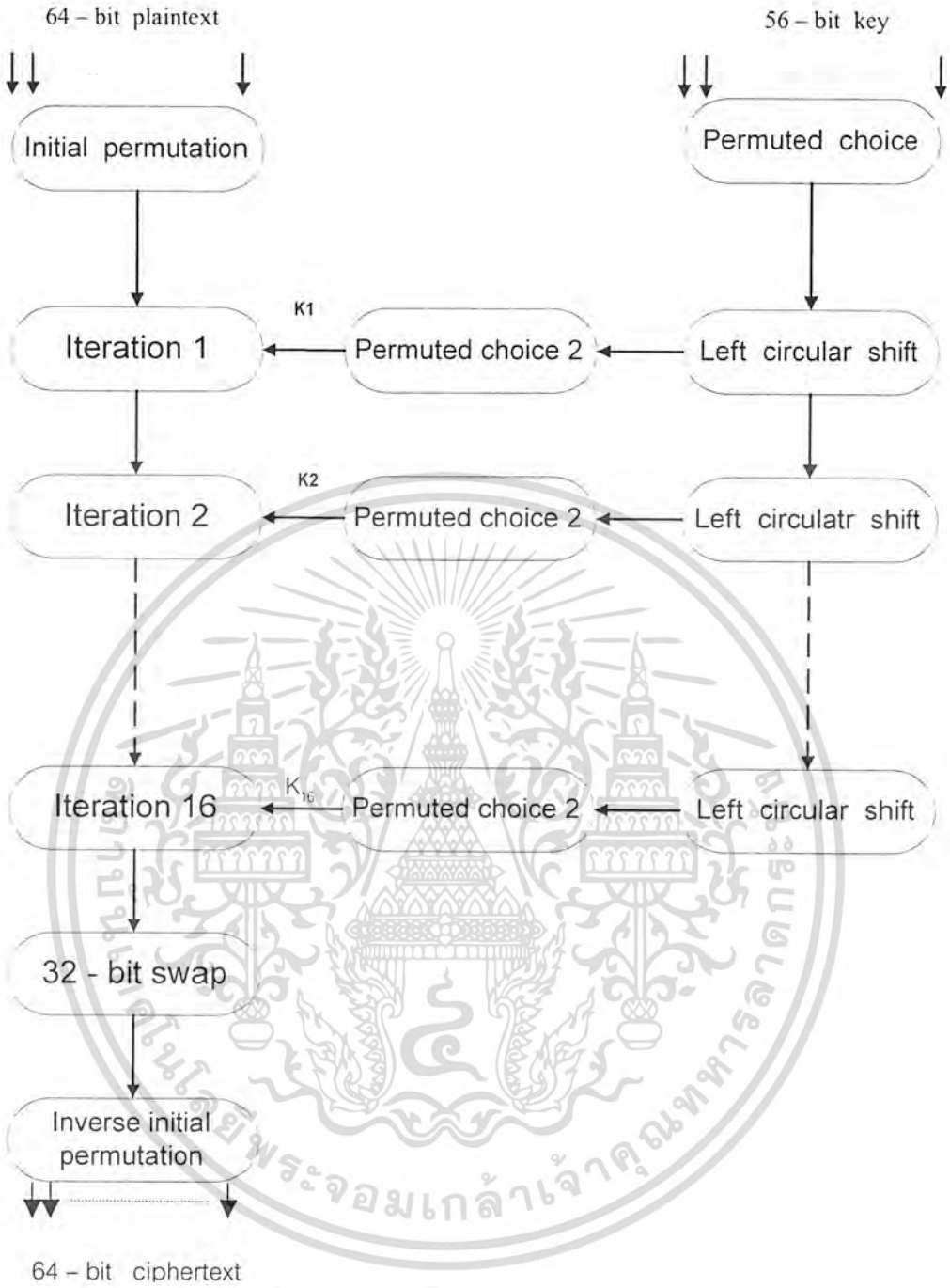
แม้ว่า DES ถูกนำมาใช้ตั้งแต่ช่วงทศวรรษที่ 70 (ค.ศ. 1960 – 1970) และได้รับการตอบรับอย่างดีจากเหล่านักวิเคราะห์รหัส (Cryptanalysis) อย่างแพร่หลาย แต่ก็เป็นข้อถกเถียงกันเป็นอย่างมากถึงเรื่อง DES นั้นจะปลอดภัยได้หรือไม่และมีความปลอดภัยมากน้อยแค่ไหน แต่จนถึงปัจจุบันเราก็ยังไม่พบช่องโหว่ของ DES ตามเอกสารที่ตีพิมพ์เป็นสาธารณะ แม้ว่าจะใช้คีย์เพียงไม่กี่บิตก็ตาม ในทางตรงกันข้ามแนวความคิดแบบ IDEA กลับใช้คีย์แบบ 128 บิต (ซึ่งขนาดกว่า 2 เท่าของ DES) และได้รับการตอบรับจากสาธารณะตั้งแต่ทศวรรษ 90 (ค.ศ. 1980 – 1990) (แต่ดีไม่เท่าตอนประกาศใช้ DES) IDEA มีความปลอดภัยมากกว่า DES และสามารถประมวลผลได้เร็วกว่า DES อย่างไรก็ตาม IDEA ยังต้องรอการตรวจสอบจากผู้เชี่ยวชาญอีกมากถึงเรื่องช่องโหว่ของความปลอดภัย

การทำงานของ DES จะมีลักษณะดังรูป ข้อมูลที่เข้ามาในส่วนฟังก์ชันของการเข้ารหัสจะมี 2 ส่วนด้วยกันคือ ข้อมูลที่ยังไม่ถูกเข้ารหัสขนาด 64 บิตและคีย์ซึ่งมีขนาด 56 บิต ซึ่งรูปในด้านซ้ายมือจะแสดงขั้นตอนจัดการกับข้อมูลที่ยังไม่เข้ารหัส โดยสามารถแบ่งย่อยๆได้อีก 3 เฟสด้วย

- เฟส 1 จะทำการจัดการกับข้อมูลที่ไม่ได้ผ่านการเข้ารหัสที่มีขนาด 64 บิตผ่านเข้าไปยังส่วนที่เรียกว่า Initial Permutation (IP) ซึ่งจะทำการเรียบเรียงบิตใหม่เพื่อผลิต "permuted input" ข้อมูลที่มีการสลับตำแหน่ง

- เฟสที่ 2 จะทำการฟังก์ชันเดียวกัน 16 ครั้งซึ่งฟังก์ชันนี้ร่วมทั้งการทำ permutation และ substitution ซึ่งผลลัพธ์ที่ได้จากการทำทั้งหมด 16 ครั้งนี้จะได้ข้อมูลขนาด 64 บิตโดยใช้ทั้งข้อมูลที่ไม่ได้ผ่านการเข้ารหัสและคีย์ในการทำ ซึ่งข้อมูลที่มีขนาด 64 บิตที่ได้นี้แบ่งเป็น 2 ด้านคือซ้ายและขวา ทั้งหมดจะถูกสวีพ เพื่อผลิต 64 บิตที่เป็น preoutput

- เฟสที่ 3 จะนำ preoutput ผ่านเข้าไปยังส่วนที่เรียกว่า "Inverse initial permutation" :IP^{-1} ซึ่งทำหน้าที่ inverse ตัว initial permutation function ซึ่งทั้งหมดจะผลิต 64 บิตที่เรียกว่า ข้อมูลที่ผ่านการเข้ารหัสแล้ว (Ciphertext)



รูปที่ 4-7 แสดงการขั้นตอนการทำงานของ DES

4.3.3 การทำ Initial Permutation

การทำ initial permutation และการทำ inverse initial permutation จะถูกอธิบายโดยใช้ตารางด้านล่างตามลำดับ ซึ่งจะเห็นได้ว่าฟังก์ชัน permution ทั้ง 2 เป็นส่วนกลับซึ่งกันและกัน โดยพิจารณาจาก 64 บิต:M ที่เข้ามา

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}	M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}	M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

M_i คือ เป็นตัวเลขฐานสอง เมื่อทำการ permutation X = IP(M) จะได้ดังนี้

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2	M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6	M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1	M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5	M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

ถ้าเราทำการ inverse permutation $Y = IP^{-1}(X) = IP^{-1}(IP(M))$ เราจะสามารถเห็นลำดับในการเรียงของบิตที่มีรูปแบบดังเดิม

4.3.4 รายละเอียดของการทำฟังก์ชันในแต่ละรอบ

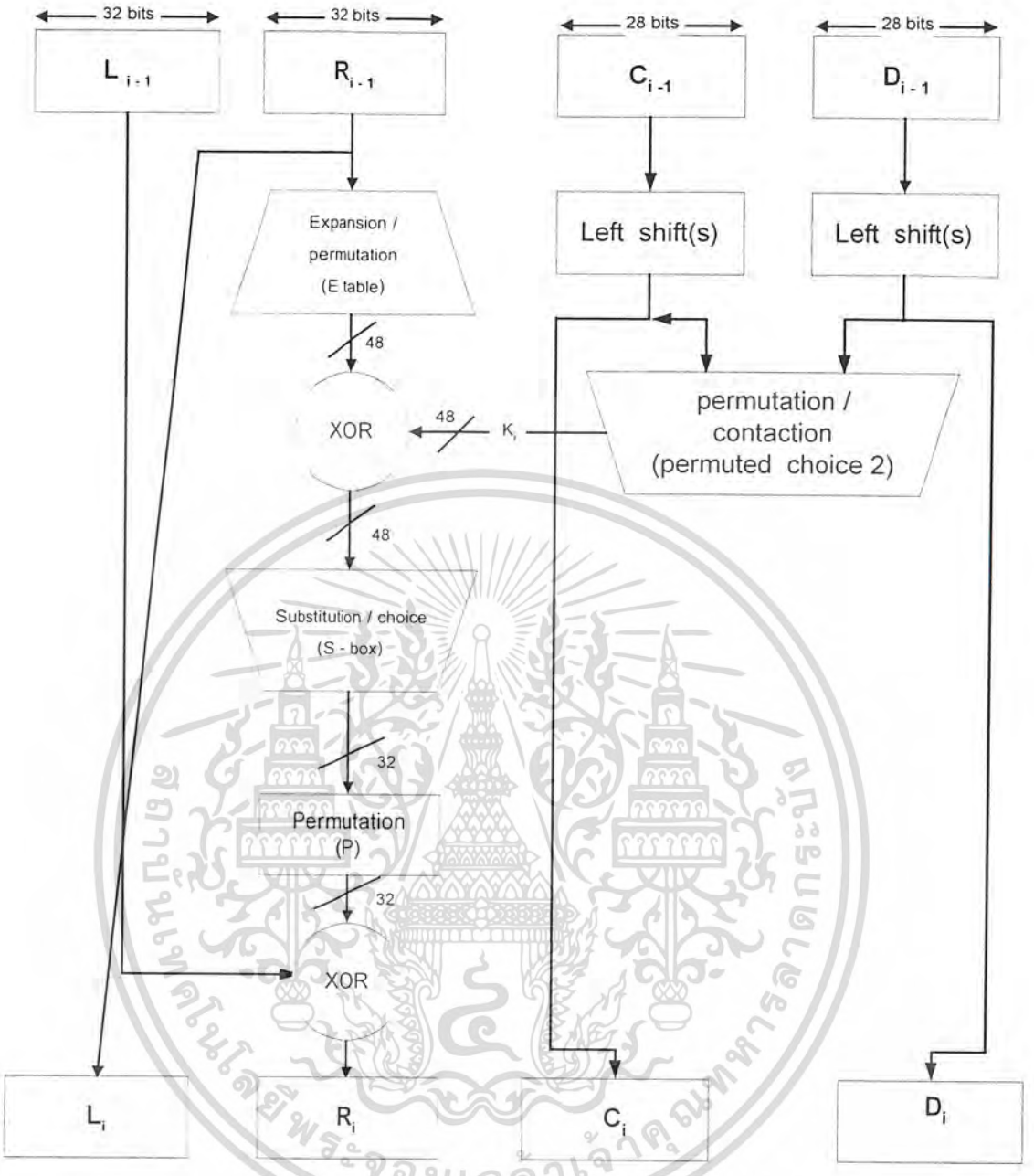
ข้อมูลที่เข้ามาที่มีขนาด 64 บิต โดยจะทำการแบ่งข้อมูลเป็น 2 ส่วนด้วยกันขนาดละ 32 บิตเท่ากัน (แบ่งเป็นซ้ายกับขวา) ซึ่งกระบวนการทำในแต่ละครั้งสามารถสรุปเป็นสูตรได้ดังนี้

$$L_i = R_{i-1}$$

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

เมื่อ $\oplus$ หมายถึง การทำ XOR function

จากสูตรจะเห็นได้ว่า 32 บิตด้านซ้ายมือ(L_i) จะเท่ากับด้านขวาของ (R_{i-1}) รอบที่ผ่านมา โดย R_i จะเท่ากับการนำ L_{i-1} มา XOR กับ $f(R_{i-1}, K_i)$ ซึ่งฟังก์ชัน f จะแสดงดังรูป 4.5



รูปที่ 4-8 แสดงการเข้ารหัส DES ในแต่ละครั้ง(ทั้งหมดทำ 16 ครั้ง)

(a) Initial Permutation (IP)

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

Output bit	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Form input bit	61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

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

(c) Expansion Permutation(E)

Output bit	1	2	3	4	5	6	7	8	9	10	11	12
Form input bit	32	1	2	3	4	5	4	5	6	7	8	9
Output bit	13	14	15	16	17	18	19	20	21	22	23	24
Form input bit	8	9	10	11	12	13	12	13	14	15	16	17
Output bit	25	26	27	28	29	30	31	32	33	34	35	36
Form input bit	16	17	18	19	20	21	20	21	22	23	24	25
Output bit	37	38	39	40	41	42	43	44	45	46	47	48
Form input bit	24	25	26	27	28	29	28	29	30	31	32	1

(d) Permutation Function (P)

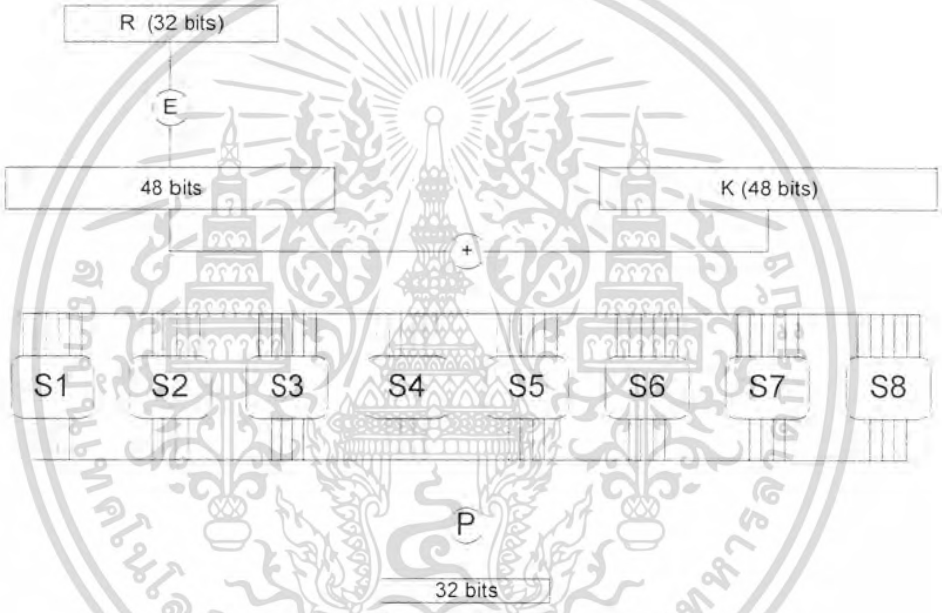
Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Form input bit	16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Form input bit	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

ตารางที่ 4-1 แสดง permutation ของ DES

โดยคีย์ K_i ที่ใช้ในแต่ละรอบจะมีขนาด 48 บิตและอินพุตด้านขวา(R)มีขนาด 32 บิต ดังนั้นจึงต้องมีการขยายขนาดจาก 32 บิตให้เป็น 48 บิต โดยใช้ตารางที่ได้มีการกำหนด permutation และ expansion ซึ่งรวมทั้งการจำลอง 16 บิตที่เพิ่มขึ้นมาของด้านขวา(R) ซึ่งจะนำผลที่ได้ที่มีขนาด 48 บิตจะถูก XOR กับ K_i โดยจะนำผลที่ได้ผ่านไปยังฟังก์ชันที่เรียกว่า Substitution และ Permutation ที่สามารถผลิตผลลัพธ์ที่มีขนาด 32 บิต

โดย Substitution จะประกอบด้วยเซตของ S – box 8 อัน (S_1 - S_8) ซึ่ง S – box จะมีอินพุตขนาด 6 บิตและผลิตเอาท์พุตขนาด 4 บิต ซึ่งตารางด้านล่างจะแสดง DES S – box โดยมีวิธีการในการแปลงอินพุตขนาด 6 บิตให้กลายเป็นเอาท์พุตขนาด 4 บิตดังนี้คือ การนำบิตแรกและบิตสุดท้ายของอินพุตมาทำเป็นตำแหน่งของแถวและนำ 4 บิตตรงกลางมาเป็นตำแหน่งของคอลัมน์ เช่น S_1 มีค่าเท่ากับ 011011 (ขนาด 6 บิต) เราจะนำบิตแรกและบิตสุดท้ายซึ่งก็คือ 0 และ 1 มาเป็นตำแหน่งของแถวจะได้แถวที่ 01 หรือถือแถว 1และ 4 บิตตรงกลางที่เหลือคือ 1101 จะได้ตำแหน่งของคอลัมน์คือ คอลัมน์ที่ 13 ดังนั้นค่าที่ตำแหน่งแถวที่ 1 และคอลัมน์ที่ 13 ในตารางคือ 0101

รูปด้านล่างจะมีรายละเอียดสำหรับ S – box operation ซึ่งในรูปจะแสดงการทำ permutation สำหรับ row 0 ของ S_1



รูปที่ 4-9 แสดงการคำนวณ f(R,K)

Column Number

Row 15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Box
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S_1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	S_2
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	

Column Number

Box

Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S_4
2	10	6	9	0	12	11	7	12	15	1	2	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S_5
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	11	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	1	
1	10	15	4	2	7	12	9	5	6	1	12	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	S_6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S_7
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	S_8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

ตารางที่ 4-2 แสดง S- box

4.3.5 การสร้างคีย์ (Key Generation)

ในรูปที่แสดงถึงการทำงานในแต่ละรอบของ DES เราจะเห็นว่าคีย์ที่ใช้มีขนาด 56 บิตซึ่งเป็นอินพุตในการทำ permutation โดยตาราง Permuted Choice One ดังรูป โดยเริ่มแรกจะทำการแบ่ง 56 บิต เป็น 2 ส่วนเท่าๆ กันส่วนละ 28 บิตโดยให้ชื่อในแต่ละส่วนว่า C กับ D ซึ่งในแต่ละรอบ (ทั้งหมด 16 รอบ) จะมีการทำ circular left shift ในแต่ละส่วนของ C และ D หรือทำ rotation โดยในแต่ละรอบจะมีการกำหนดว่าจะให้ shift ไปกี่บิตดังตาราง ซึ่งค่าที่ถูก Shift จะกลายเป็นอินพุตของการทำให้รอบถัดไปและเป็นอินพุตของการทำ Permutation Choice Two ดังในตาราง หลังจากการทำ Permutation Choice Two แล้วจะได้เอาท์พุตขนาด 48 บิต ซึ่งเป็นอินพุตของ $f(R_{I-1},K_I)$

รูปที่ 4-10 แสดงการทำ Permutation Choice

(a) Permuted Choice One(PC-1)

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

(b) Permuted Choice Two (PC-2)

Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
From input bit	14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
From input bit	26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40

Output bit	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
From input bit	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

(c) Schedule of Left Shifts

Iteration number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

ตารางที่ 4-3 แสดงการสร้างคีย์

4.3.6 การถอดรหัสข้อมูล DES

กระบวนการถอดรหัสโดยใช้ DES นั้นเหมือนกับขั้นตอนในการเข้ารหัส ซึ่งมีขั้นตอนในการทำงานดังนี้ คือ นำข้อมูลที่ผ่านการเข้ารหัสแล้ว (Ciphertext) มาเป็นอินพุตแต่จะมีการใช้คีย์ (K_i) ที่มีลำดับย้อนกลับกับคีย์ที่ใช้ในการเข้ารหัสเช่น K_{16} , K_{15} .....เป็นคีย์แรกและคีย์ถัดไปในการเข้ารหัสแทน ดังรูปด้านซ้ายมือเป็นขั้นตอนการเข้ารหัสและด้านขวามือเป็นขั้นตอนในการถอดรหัส

เราจะแสดงถึงผลลัพธ์ของขั้นตอนแรกในการกระบวนการถอดรหัส ซึ่งจะเท่ากับ 32 บิตที่ถูกสว็อพจากอินพุตของการทำทั้งหมด 16 รอบของการเข้ารหัส เริ่มจาก

$$L16 = R15$$

$$R16 = L15 \oplus f(R15,K16)$$

ในด้านการถอดรหัส

$$Ld1 = Rd0 = L16 = R15$$

$$Rd1 = Ld0 \oplus f(Rd0,K16)$$

$$= R16 \oplus f(R15,K16)$$

$$= [L15 \oplus f(R15,K16)] \oplus f(R15,K16)$$

ซึ่งคุณสมบัติของ XOR ที่สำคัญคือ

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$

$$D \oplus D = 0$$

$$E \oplus 0 = E$$

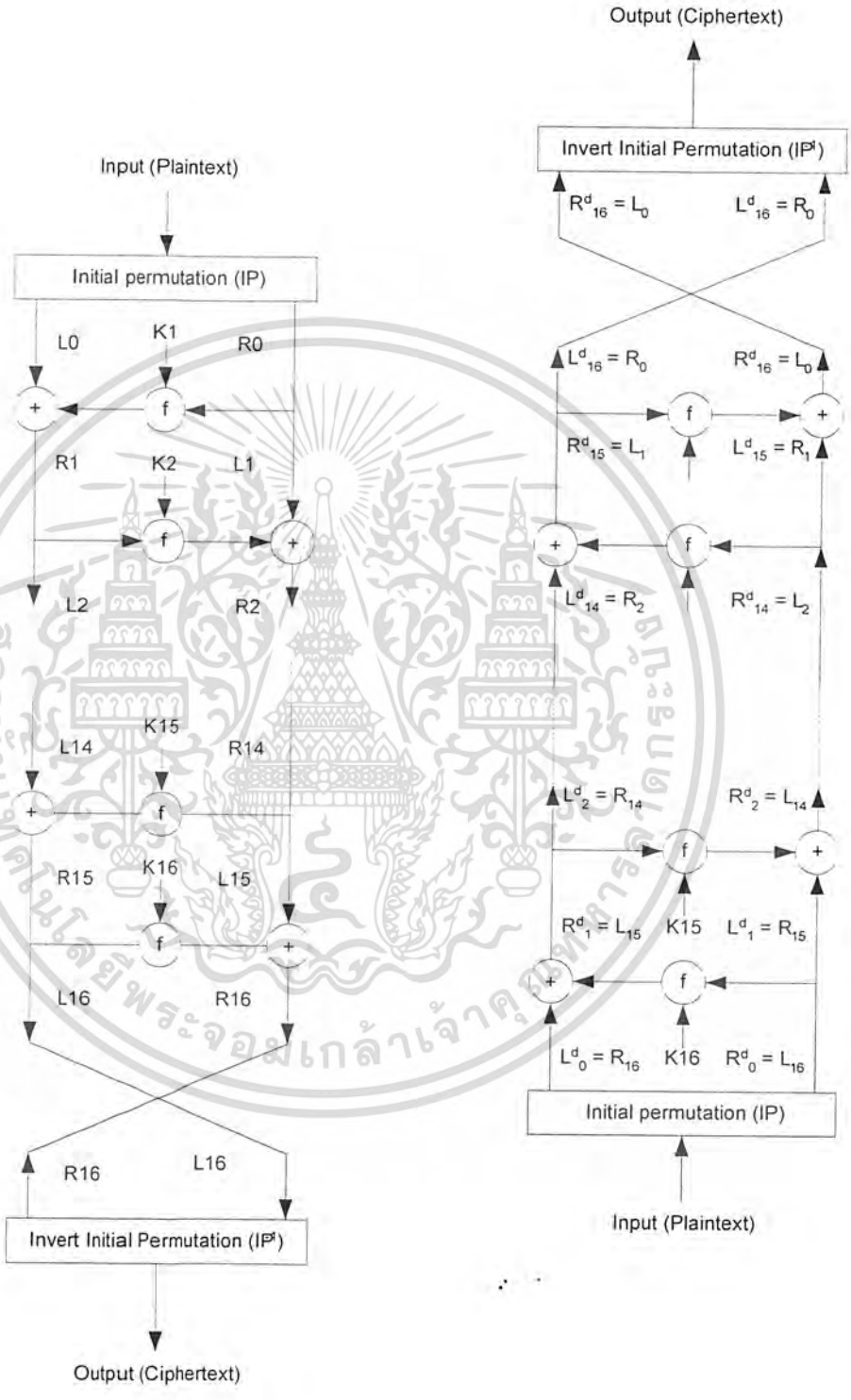
ดังนั้นเรามี Ld1 = R15 และ Rd1 = L15 จะได้เอาท์พุตของขั้นตอนแรกในการถอดรหัสคือ L15||R15 ซึ่งเราสามารถเขียนเป็นสมการในการถอดรหัสได้ดังนี้คือ

$$Ri^{-1} = Li$$

$$Li^{-1} = Ri \oplus f(Ri-1,Ki) = Ri \oplus f(Li,Ki)$$

ซึ่งผลสุดท้ายเอาท์พุตที่ได้จากขั้นตอนสุดท้ายในการถอดรหัสคือ R0||L0 และนำไปสู่ขั้นตอนในการทำ Inverse Permutation เราจะได้ข้อมูลที่ส่งมา(plaintext) ดังสมการ

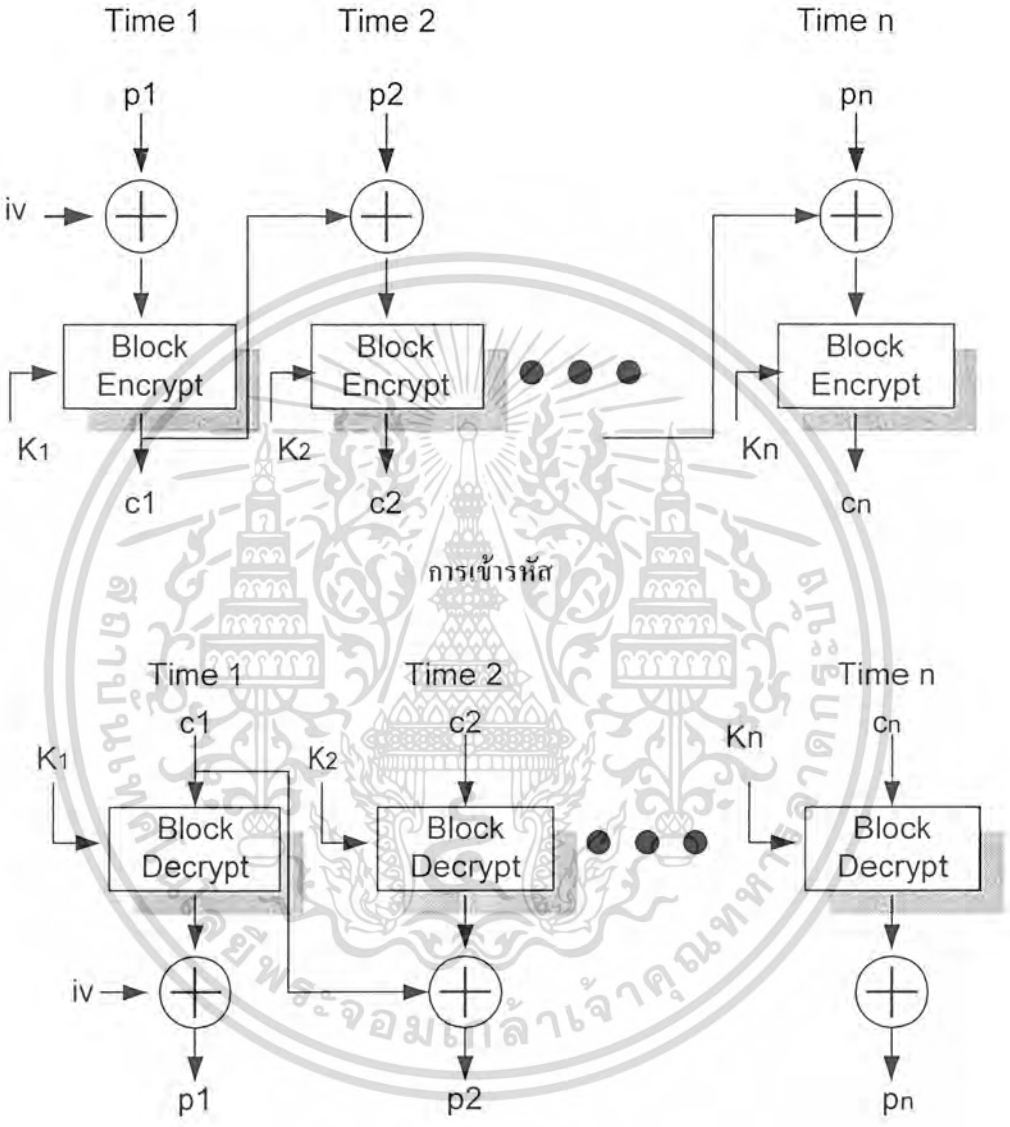
$$IP^{-1}(L0||R0) = IP^{-1}(IP(plaintext)) = plaintext$$



รูปที่ 4-11 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES

4.3.7 โหมด CBC (Cipher Block Chaining)

เป็นวิธีการเข้ารหัสที่พัฒนามาจาก DES ช่วยทำให้ข้อมูลที่ส่งยิ่งมีความปลอดภัยมากยิ่งขึ้น โดยมีวิธีการคือ จะนำข้อมูลที่ผ่านการเข้ารหัสของข้อมูลตัวก่อนมา XOR กับข้อมูลที่ยังไม่ได้เข้ารหัสของตัวถัดไปก่อนจะทำการเข้ารหัสแบบ DES ตามปกติดังรูป



รูปที่ 4-12 แสดงการเข้ารหัสและถอดรหัสของ DES CBC

ในการถอดรหัสข้อมูล ก็จะทำเช่นเดียวกับการถอดรหัสข้อมูล DES ตามปกติแต่จะนำผลที่ได้จากการทำถอดรหัสมา XOR กับข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) ตัวก่อนหน้านี้ เพื่อจะได้ข้อมูลจริงๆ(plaintext) ดังสมการ

$$Cn = Ek[Cn^{-1} \oplus Pn]$$

สมการในการถอดรหัส

$Dk[Cn] = Dk[Ek(Cn^{-1} \oplus Pn)]$

$Dk[Cn] = Cn^{-1} \oplus Pn$

$Cn^{-1} \oplus Dk[Cn] = Cn^{-1} \oplus Cn^{-1} \oplus Pn = Pn$

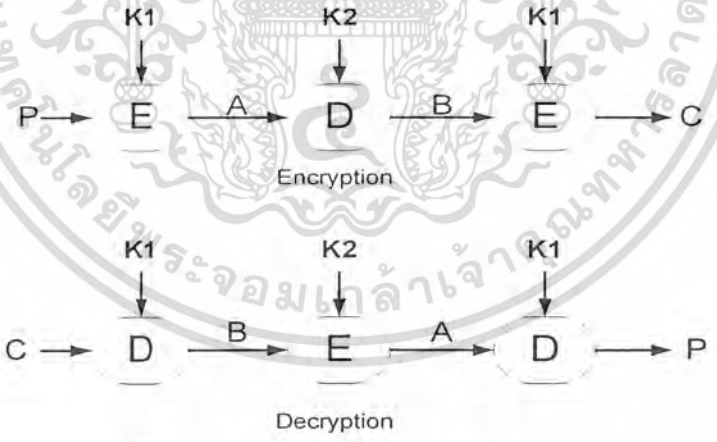
จะสังเกตเห็นได้ว่าบล็อกแรกของข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) จะมีการนำ iv (Initialization Vector) มาทำการ XOR กับข้อมูลที่ไม่ได้เข้ารหัส (plaintext) ก่อนจะมีการเข้ารหัส DES ตามปกติ และในส่วนของการถอดรหัสข้อมูลก็จะใช้ iv ในการถอดรหัสข้อมูลเช่นเดียวกัน ดังนั้นจึงจำเป็นต้องมีข้อตกลงกันระหว่างผู้ส่งกับผู้รับก่อนว่าจะใช้ iv เป็นค่าใด เพื่อให้มีความปลอดภัยสูงสุด iv ควรจะมีการป้องกันเช่นเดียวกับ Key ซึ่งในการส่งค่า iv อาจจะทำการส่งโดยใช้การเข้ารหัสแบบ ECB

4.4 การเข้ารหัสแบบ 3DES (Triple DES)

เป็นที่ทราบกันว่ายิ่งคีย์มีความยาวมากขึ้น การถอดรหัสยิ่งทำได้ยากยิ่งขึ้น นอกจากนี้เราก็สามารถเพิ่มสมรรถนะของการเข้ารหัสให้มากขึ้นได้โดย การเข้ารหัสหลายๆ ครั้ง

อย่างไรก็ตามการเข้ารหัสครั้งที่สองโดยใช้คีย์ที่ต่างจากครั้งแรกนั้น มิได้หมายความว่าข้อมูลจะมีความปลอดภัยเพิ่มขึ้นเป็น 2 เท่าถ้าการเข้ารหัสดังกล่าวเป็นการเข้ารหัสโดยเทคนิคทางคณิตศาสตร์

DES มิได้อยู่ในกลุ่มดังกล่าวแต่การที่จะทำให้ DES มีความปลอดภัยมาขึ้นเป็น 2 เท่านั้นจำเป็นที่เราจะต้องเข้ารหัสถึง 3 ครั้ง



รูปที่ 4-13 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES

ขั้นตอนของการเข้ารหัสแบบ 3 ครั้งนั้นมีดังนี้

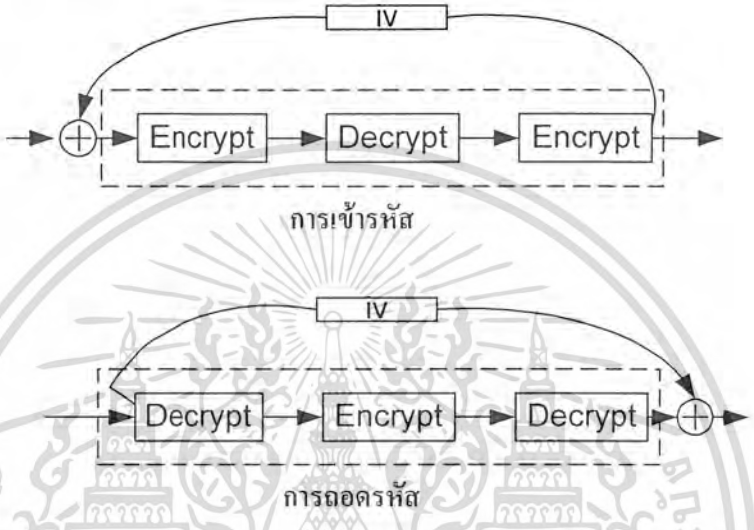
1. เข้ารหัสโดยใช้คีย์ตัวแรก
2. ถอดรหัสโดยคีย์ตัวที่สอง
3. เข้ารหัสโดยใช้คีย์ตัวที่สาม

(อาจใช้คีย์ตัวที่ 1 และ 3 เป็นตัวเดียวกันก็ได้ แต่ประสิทธิภาพของการเข้ารหัสก็จะลดลง)

4.4.1 Triple DES โหมด CBC (3DES_CBC Mode)

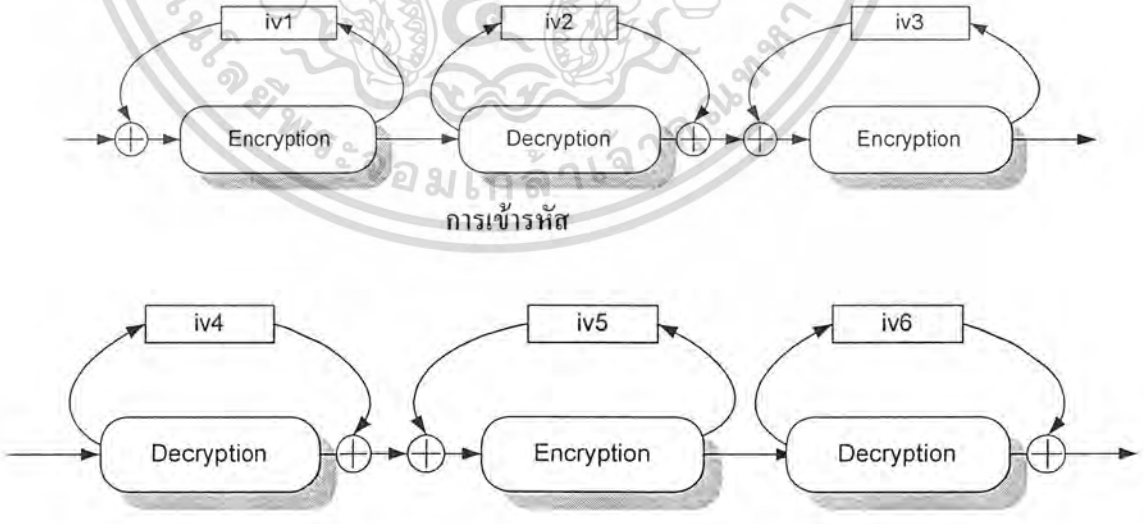
เนื่องจากการเข้ารหัสแบบ DES นั้นมีโหมดในการเข้ารหัสแบบ CBC ซึ่งการเข้ารหัส แบบ 3DES นี้ได้มีการประยุกต์มาจาก DES จึงได้มีการพัฒนา 3DES นี้ให้มีโหมด CBC ด้วย ซึ่งในการเข้ารหัสแบบ 3DES นี้ได้พัฒนาโหมดนี้ออกเป็น 2 แบบ

- **แบบ inner CBC** การเข้ารหัสแบบนี้จะมี iv เพียงตัวเดียวในการเข้ารหัสหรือถอดรหัสแบบ 3DES ในแต่ละครั้ง ดังรูป 4-14



รูปที่ 4-14 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด inner CBC

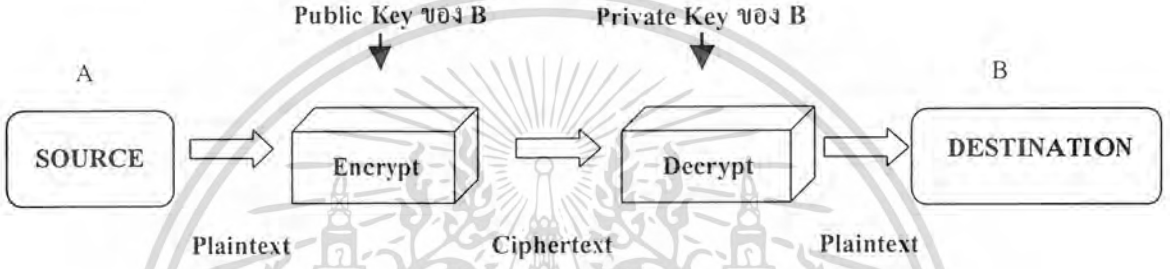
- **แบบ outer CBC** การเข้ารหัสแบบนี้แต่ละโมดูลในการเข้าหรือถอดรหัสจะมี iv เฉพาะของแต่ละโมดูล ดังรูป 4-15



รูปที่ 4-15 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด outer CBC

4.5 การเข้ารหัสแบบ RSA

RSA ถูกคิดค้นขึ้นในปี 1977 โดยที่ชื่อ RSA ได้มาจากอักษรตัวแรกของนามสกุลของผู้ร่วมกันคิดค้นคือ Ron River, Adi Shamir และ Leonard Adleman เป็นวิธีการเข้ารหัสแบบกุญแจสาธารณะที่เรียกว่าพับลิกคีย์ (Public–Key Crytosystem) เพื่อแก้ไขปัญหาการทำให้กุญแจเป็นความลับ (Secret–Key Cryptosystem) โดยสมาชิกแต่ละคนจะต้องมีกุญแจ 2 ชนิดคือ กุญแจส่วนตัวหรือไพรเวตคีย์ (Private Key) และกุญแจสาธารณะหรือพับลิกคีย์ (Public Key) โดยกุญแจส่วนตัวจะถูกเก็บไว้เป็นความลับ ส่วนกุญแจสาธารณะนั้นจะเปิดเผยให้ใครก็ได้ที่ต้องการส่งเอกสารให้แก่ตน การทำงานของรหัสลับมีหลักการว่าข้อมูลที่ถูกเข้ารหัสลับด้วยกุญแจสาธารณะของผู้ใด จะถูกถอดรหัสได้ด้วยกุญแจส่วนตัวของผู้นั้นเท่านั้น การทำงานของระบบพับลิกคีย์สามารถอธิบายได้ดังต่อไปนี้



รูปที่ 4-16 แสดงขั้นตอนการทำ Public – Key Cryptosystem

4.5.1 หลักการทำงานของ RSA

ถ้าให้ p และ q เป็นจำนวนเฉพาะที่มีค่ามาก โดยที่ n = p*q เรียกว่าโมดูลัส (modulus) จากนั้นจึงเลือก e ที่มีค่าน้อยกว่า n และไม่สามารถหาร (p-1)(q-1) ได้ลงตัว ถ้าให้ d เป็นส่วนกลับของ e ในคณิตศาสตร์ระบบมอดูโลฐาน (p-1)(q-1) นั่นคือ

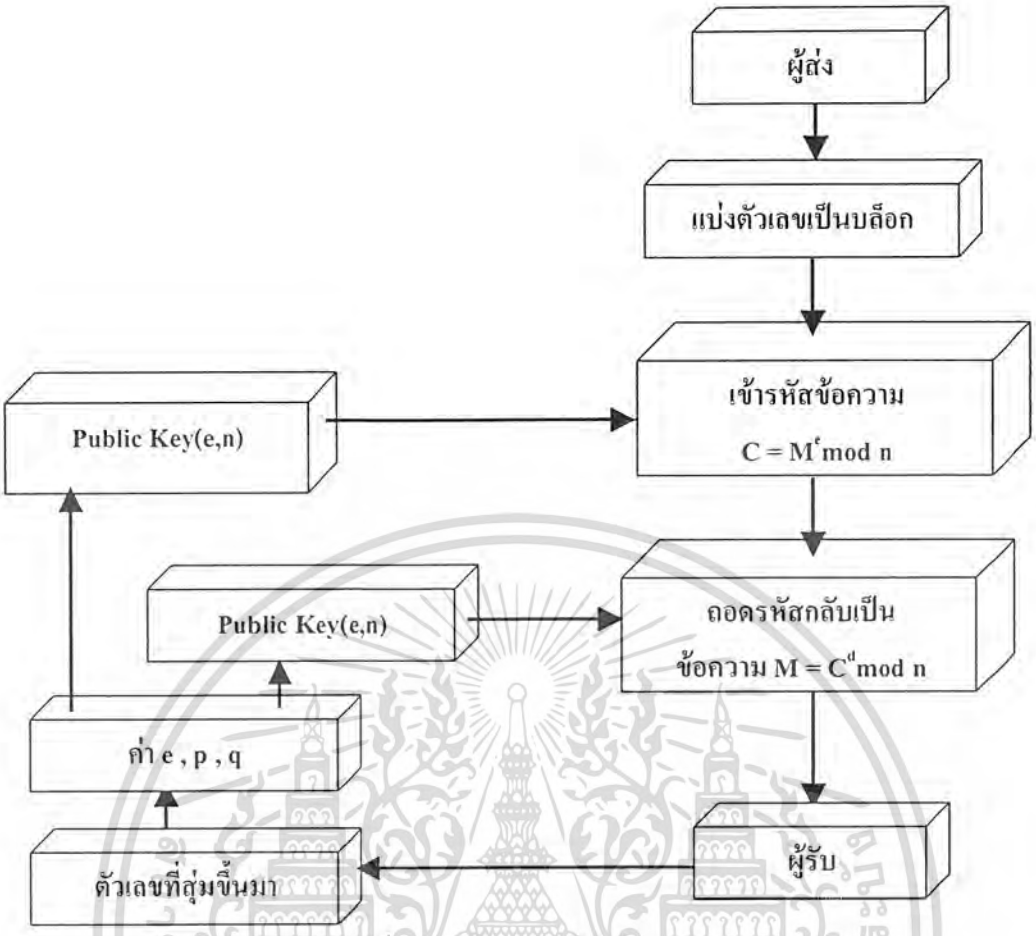
$$e*d \bmod (p-1)(q-1) = 1 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

ในรหัส RSA นั้น (n,e) คือพับลิกคีย์ (public key) ส่วน d คือไพรเวตคีย์ (private key) เมื่อได้ค่าเหล่านี้แล้ว p,q จะต้องเก็บเป็นความลับหรือถูกทำลายในทันที

ถ้าอริสาต้องการส่งข้อเอกสารส่วนตัว m ไปให้บุญมี อริสาจะสร้างรหัสลับ c ของ m ได้โดยการให้ $c = m^e \bmod n$ เมื่อ(n,e) เป็นพับลิกคีย์ของบุญมี เมื่อบุญมีได้รับเอกสารรหัสลับ c เขาจะถอดรหัสเพื่ออ่านเอกสาร m ได้ด้วย d เพราะความสัมพันธ์ในสมการ(1) ระหว่าง e,d และ n จะทำให้

$$c^d = m^{ed \bmod (p-1)(q-1)} \bmod n = m \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

และเพราะมีแต่บุญมีเท่านั้นรู้ค่า d บุญมีเท่านั้นจะถอดรหัสได้ คุณสมบัติสำคัญประการหนึ่งของ RSA คือจากสมการ (2) ในทางกลับกันถ้าเราเข้ารหัสด้วยไพรเวตคีย์ เราก็สามารถถอดรหัสด้วยพับลิกคีย์ได้ด้วยเช่นกัน คุณสมบัติข้อนี้เองที่ทำให้ RSA มีประโยชน์มากเพราะในการใช้การเซ็นระบบดิจิตอลได้อีกด้วย



รูปที่ 4-17 แสดงการเข้ารหัสแบบ RSA

ตัวอย่างขั้นตอนการเข้ารหัสแบบ RSA ซึ่งมีขั้นตอนในการหาคีย์ ดังนี้

1. เลือกจำนวนเฉพาะสองจำนวน คือ p = 7 , q = 17
2. คำนวณ n = p*q = 7 * 17 = 119
3. คำนวณ (p-1)(q-1) = 96
4. เลือก e ซึ่งมีความสัมพันธ์กับค่า (p-1)(q-1) ที่ได้กล่าวไว้ข้างต้น ในที่นี้เราจะใช้ 3
5. คำนวณค่า d ซึ่งสัมพันธ์กับสมการ e * d = 1 mod 96 ซึ่งค่าที่ถูกต้องคือ d = 77 เนื่องจาก

   77 * 5 = 385 = 4 * 96 +1

ดังนั้น Public Key คือ {5,119} และมีค่า Private Key คือ {77,119}

วิธีทำลายรหัส RSA ที่รู้จักกันดีที่สุดคือการหาค่า d นั่นเองจากสมการที่ (1) เราอาจหาค่า d ได้ หากรู้ค่า p และ q แต่เนื่องจาก p และ q เป็น prime number ที่ p*q = n ดังนั้นการทำลายรหัส RSA ขึ้นอยู่กับการแยกตัวประกอบของ n นั่นเอง แต่วิธีการแยกตัวประกอบของ n นั้นไม่ง่ายเลยสำหรับหาก n มีค่ามาก R.Rivest ได้คำนวณไว้ในปี 1992 ว่าจะต้องใช้เงินประมาณ 8.3 ล้านเหรียญสหรัฐในการแยกตัวประกอบของ n ที่มีความยาว 512 บิต ค่านี้อาจลดลงได้ในอนาคต ดังนั้นสำหรับข้อมูลที่มีความสำคัญมากกว่า อาจจำเป็นต้องใช้ n ที่มีค่ามากถึง 700 หรือ 1000 ก็ได้

จะเห็นได้ว่าไม่ว่าการเข้าหรือถอดรหัส RSA ก็จำเป็นต้องใช้การยกกำลังในระบบมอดูโล ซึ่งการยกกำลังนั้นสามารถทำได้โดยการใช้วงจรคูณระบบมอดูโลมาต่ออนุกรมกัน ดังนั้นความเร็วของทั้งการเข้าและถอดรหัสจึงขึ้นกับความเร็วของวงจรคูณระบบมอดูโลเป็นอย่างมาก นี่เองเป็นจุดอ่อนข้อหนึ่งของ RSA เมื่อเปรียบเทียบกับกุญแจเดียว เช่น DES เพราะปัจจุบันการคูณในระบบมอดูโลยังค่อนข้างยุ่งยากเมื่อเปรียบเทียบกับการแทนค่าหรือสลับตำแหน่งใน DES ได้มีการประมาณกันว่าสำหรับ n ที่มีความยาว 512 บิต การเข้ารหัสแบบ DES จะเร็วกว่า RSA ประมาณ 100 เท่า ถ้าเราทำการเข้ารหัสด้วยซอฟต์แวร์ และอาจเร็วกว่าถึง 1000 ถึง 10000 แล้วแต่ลักษณะของวงจร หากทำการเข้ารหัสด้วยฮาร์ดแวร์ โดยทั่วไปแล้วเราต้องการให้การเข้ารหัสเร็วกว่าการถอดรหัสดังนั้นเราจึงมักเลือกให้ e มีค่าน้อยกว่า d และยิ่งกว่านั้นเรามักให้ e ของสมาชิกทุกคนมีค่าเดียวกันเพื่อให้ฮาร์ดแวร์ของวงจรเข้าสำหรับสมาชิกแต่ละคนมีลักษณะคล้ายกัน

เนื่องจาก DES และ RSA มีข้อดีข้อเสียที่แตกต่างกัน จึงไม่จำเป็นว่ารหัสชนิดใดชนิดหนึ่งจะเหมาะสมในทุกสถานการณ์ โดยทั่วไปแล้ว DES จะถูกใช้ในการเข้ารหัสข้อมูลที่มีขนาดใหญ่เพราะรวดเร็วกว่าในขณะที่ RSA จะถูกใช้ในระบบสื่อสารที่ไม่ยาวนักแต่ต้องการความปลอดภัยสูงในบางครั้ง RSA ยังถูกใช้ร่วมกับ DES เพื่อเสริมจุดเด่นซึ่งกันและกัน เช่นตัวเอกสารจริงจะถูกเข้ารหัสด้วย DES โดยที่คีย์รหัส DES จะถูกเข้ารหัสด้วย RSA แล้วส่งไปด้วยกันหรือส่งไปก่อนแต่ในบางครั้ง DES อย่างเดียวก็พอแล้วหากการแลกเปลี่ยนคีย์สามารถทำได้อย่างปลอดภัยเพียงพอ หรือในกรณีที่ผู้ส่งและผู้รับเป็นบุคคลเดียวกัน เช่น ฮาร์ดดิสก์ในคอมพิวเตอร์ส่วนตัวหรือข้อมูลส่วนตัวในสมาร์ตการ์ด

4.6 การคำนวณแบบ MD5

เป็นอัลกอริทึมที่ใช้ทำ message digest จากข้อความต่างๆ เพื่อให้ออกมามีขนาด 128 บิต โดย input จะถูกจัดการทีละบล็อก ๆ ละ 512 บิต โดยมีขั้นตอนต่อไปนี้

ขั้นที่ 1 : เติม Padding Bits ข้อความจะต้องถูกต่อเติมให้มีความยาวที่ถูกมอดูโลด้วย 512 แล้วได้ 448 หากไม่ถึงหรือเกินให้เติม padding เข้าไปโดยมีลักษณะเป็นค่าบิต 1 ตามด้วย 0 จนครบ เพื่อใช้เนื้อที่ 64 บิตที่เหลือ (512 ลบ 448) ใส่ขนาดของข้อความ (จำนวนบิต)

ขั้นที่ 2 : เติมความยาว ใส่ขนาดของข้อความโดยจะมีค่าไม่เกิน 2^{64}

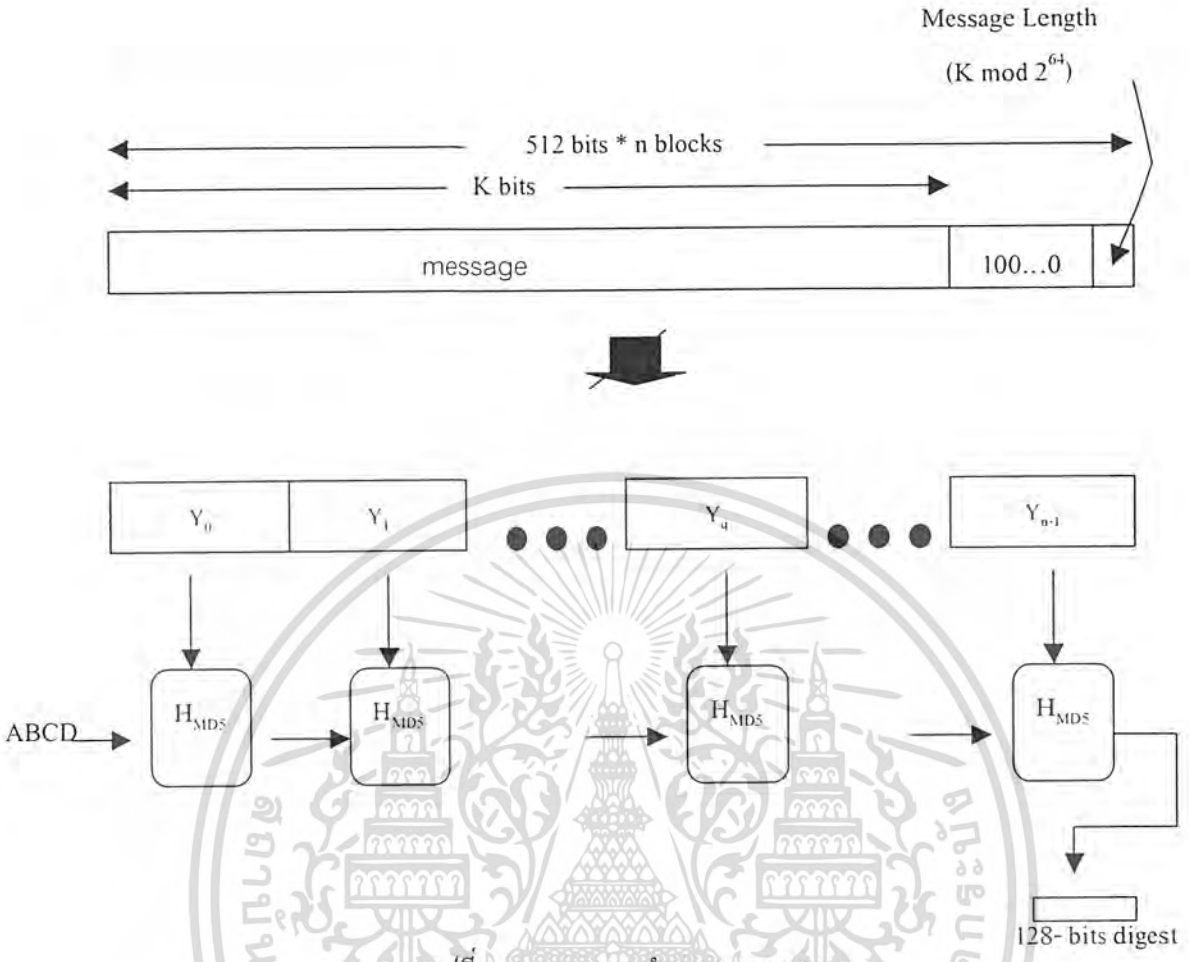
ขั้นที่ 3 : กำหนดค่าเริ่มของ MD บัฟเฟอร์ บัฟเฟอร์จะมีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการ hash โดยบัฟเฟอร์เริ่มต้นจะประกอบด้วย 32 บิตไบต์ 4 ตัว (A,B,C,D) ซึ่งมีค่าต่อไปนี้

A = 01234567

B = 89ABCDEF

C = FEDCBA98

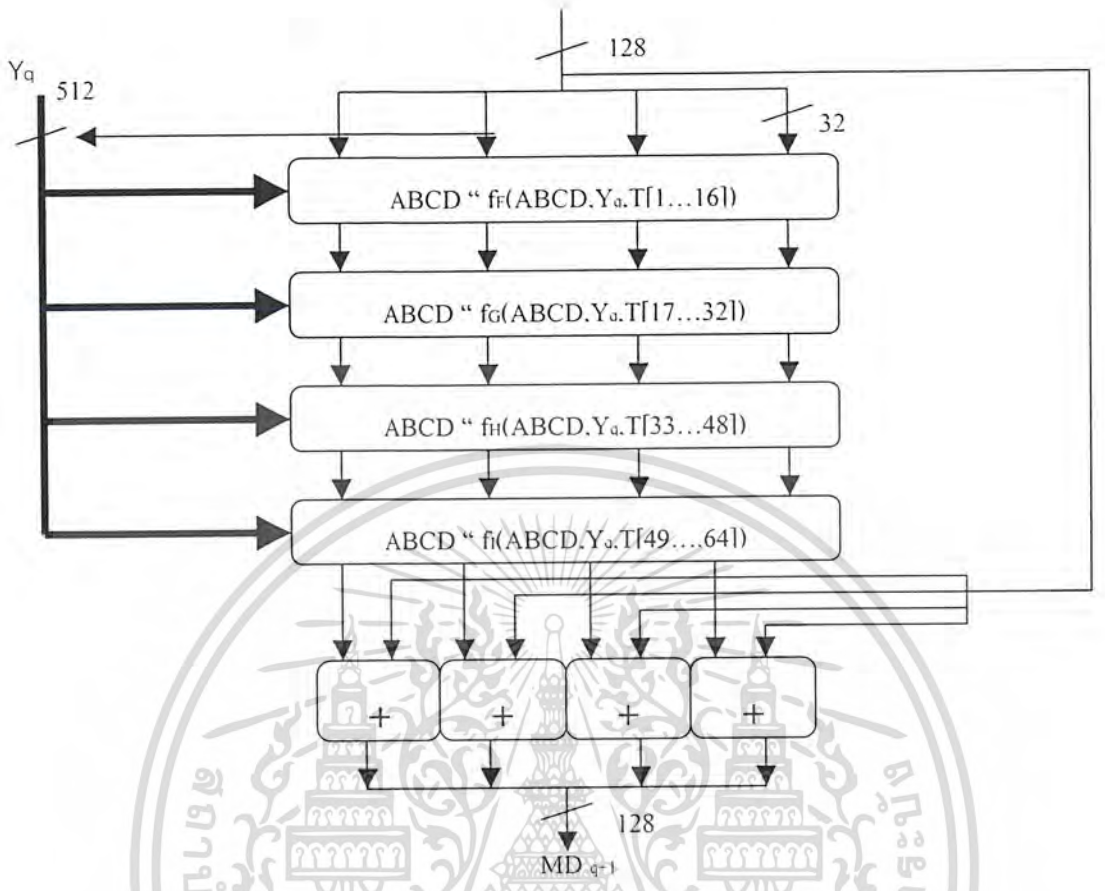
D = 76543210



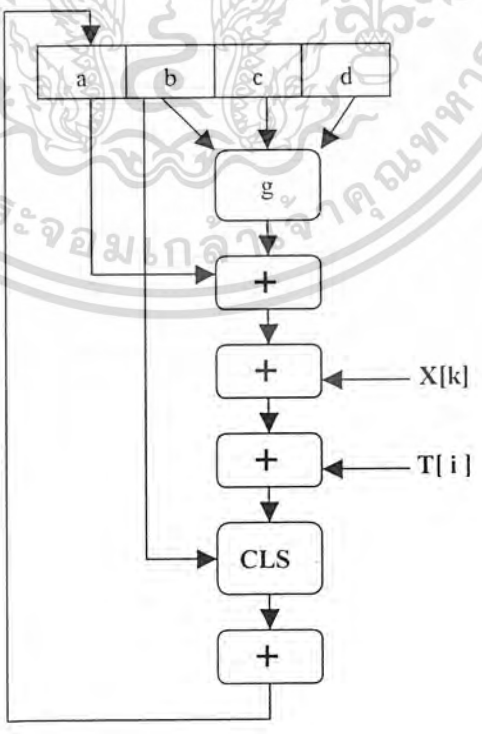
รูปที่ 4-18 แสดงการคำนวณแบบ MD5

ขั้นที่ 4 : กระบวนการจัดการข้อความ 512 บิตบล็อก เป็นหัวใจของกระบวนการทั้งหมดประกอบด้วย 4 รอบของกระบวนการที่คล้ายกันโดยแต่ละรอบจะใช้ฟังก์ชันเฉพาะของมัน โดยให้เป็น F,G,H และ I โดยในรูป f_F ,f_G ,f_H ,f_I ในการบอกว่าแต่ละรอบใช้กระบวนการที่เหมือนกันแต่ใช้ฟังก์ชันภายในที่แตกต่างกัน (F,G,H,I)

แต่ละรอบจะใช้อินพุตขนาด 512 บิตบล็อกในการจัดการ(Y_q) และ 128 บิตบัฟเฟอร์ที่มีค่า A,B,C,D และทำการอัพเดตค่าในบัฟเฟอร์ ในแต่ละรอบจะต้องมีการใช้ 1 ใน 4 ของ 64 ค่าในตาราง T[1...64] ซึ่งถูกสร้างมาจากฟังก์ชัน sine โดย T[i] จะมีค่าเป็นจำนวนเต็มของ 232 * abs(sin(i)) โดยที่ i มีหน่วยเป็น radians



รูปที่ 4-19 กระบวนการทำในหนึ่งบล็อกของ MD5



รูปที่ 4-20 แสดงการทำหนึ่ง operation ของ MD5 [abcd k s i]

โดย $a \leftarrow b + CLSs(a + g(b,c,d) + X[k] + T[i]$

และ

Round	Primitive function g	G(b ,c ,d)
FF	F(b, c ,d)	(b . c) v ($\bar{b}$. d)
FG	G(b, c ,d)	(b . d) v (c . $\bar{d}$)
FH	H(b, c ,d)	b $\oplus$ c $\oplus$ d
FI	I(b, c ,d)	c $\oplus$ (b . $\bar{d}$)

ขั้นที่ 5: Output

นำเอาท์พุตที่ได้จากการทำครั้งแรก (128 บิต) ซึ่งจะได้เป็น A,B,C,D ใหม่มาทำการโพรเซสกับบล็อกถัดไปจนครบทั้งหมด ซึ่งจะได้ผลลัพธ์ขนาด 128 บิตด้วย

บทที่ 5

เซ็กเคียวเชลล์

เซ็กเคียวเชลล์ เป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่ผู้ใช้สามารถล็อคอินเข้าใช้บริการจากเครื่องให้บริการที่อยู่ห่างไกลบนระบบเครือข่าย และสามารถทำการประมวลผล คำสั่งในเครื่องให้บริการที่อยู่ห่างไกลได้ (remote machin) พร้อมทั้งสามารถแลกเปลี่ยนข้อมูลบนระบบเครือข่ายอินเตอร์เน็ตได้อย่างปลอดภัย เนื่องจากมีวิธีการพิสูจน์ความเป็นเจ้าของที่มีประสิทธิภาพ (Strong Authentication) และระบบการติดต่อที่ปลอดภัย (Secure Communication) บนระบบเครือข่าย

5.1 ภาพรวมของโพรโตคอลเซ็กเคียวเชลล์

โพรโตคอลเซ็กเคียวเชลล์ประกอบด้วยโปรแกรมในส่วนของเครื่องที่ให้บริการ (เซิร์ฟเวอร์) และอีกส่วนหนึ่งคือส่วนของเครื่องใช้บริการ (ไคลเอ็นต์) โดยการติดต่อระหว่างเครื่องให้บริการและเครื่องใช้บริการจะติดต่อกันโดยใช้หมายเลขไอพี (IP Address) และ ทีซีพี/ไอพีซ็อกเก็ต (TCP/IP socket) ซึ่งเป็นการติดต่อสองทิศทาง (bi-directional communication)

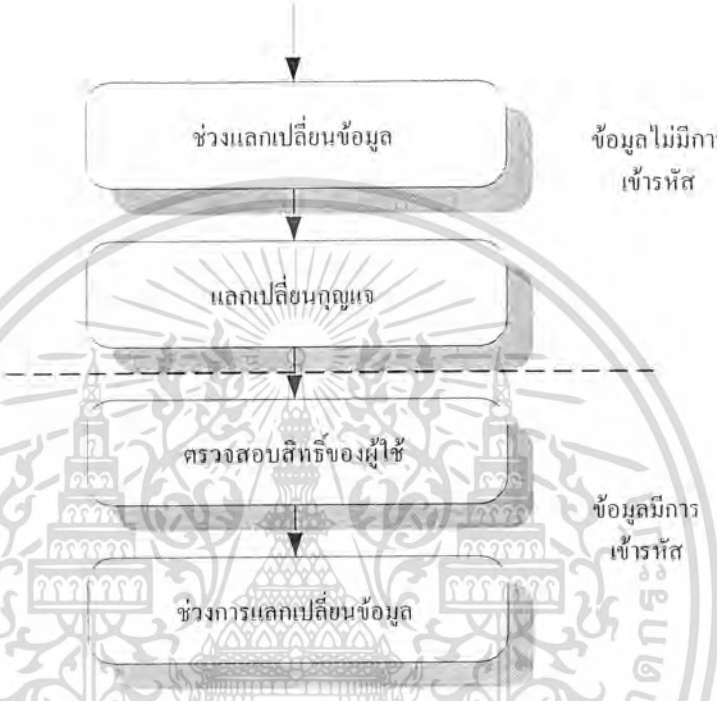
การติดต่อกันจะเริ่มการติดต่อจากเครื่องใช้บริการ โดยเครื่องให้บริการจะรอฟังจากพอร์ตที่กำหนดไว้เฉพาะ ว่ามีเครื่องที่จะใช้บริการติดต่อเข้ามายังพอร์ตนั้นหรือไม่ ซึ่งเครื่องให้บริการสามารถติดต่อกับเครื่องใช้บริการได้หลายเครื่องพร้อมๆ กัน

เมื่อเครื่องผู้ใช้บริการทำการติดต่อกับเครื่องผู้ให้บริการ หากเครื่องผู้ให้บริการยอมรับการติดต่อก็จะส่งข้อความเพื่อตรวจสอบเวอร์ชันของกันและกัน ที่เรียกว่า **Version Identification String** ของตนกลับไป เมื่อเครื่องใช้บริการได้รับก็จะทำการแปลข้อความของผู้ให้บริการ และส่งข้อความในการตรวจสอบเวอร์ชันของตนเองกลับไปเช่นกัน ซึ่งหน้าที่ของ **Version Identification String** คือ เป็นข้อความที่ใช้ในการตรวจสอบการติดต่อว่าเวอร์ชันของโปรแกรมและโพรโตคอลนั้นรองรับกันหรือไม่

หลังจากทำการตรวจสอบเวอร์ชันของกันและกันและกันแล้ว ข้อมูลในการติดต่อจะเปลี่ยนมาเป็นรูปแบบของแพ็กเก็ตซึ่งเรียกโพรโตคอลในการติดต่อนี้ว่า **binary packet protocol** โดยเครื่องผู้ให้บริการจะเริ่มส่งการติดต่อโดยส่งกุญแจสาธารณะที่เรียกว่า Host Public Key ,Server Key (สำหรับรายละเอียดเกี่ยวกับกุญแจต่างๆ จะอยู่ในบทที่ 4) และข้อมูลอื่นๆ ไปยังเครื่องผู้ใช้บริการ ตัวผู้ใช้บริการจะทำการผลิตเซสชันคีย์ (Session Key) ซึ่งใช้ในการเข้ารหัสข้อมูลที่มีขนาด 256 บิต ซึ่งจะถูกเข้ารหัสแบบ RSA 2 ครั้ง และทำการส่งไปให้เครื่องผู้ให้บริการพร้อมทั้งชนิดของการเข้ารหัสข้อมูล (cipher) ที่ใช้ หลังจากนั้นทั้ง 2 ฝั่งจะเปิดการติดต่อกันโดยข้อมูลที่ติดต่อกันนี้จะมีการเข้ารหัสโดยใช้กุญแจ (Key) และวิธีการที่ได้กำหนดไว้ข้างต้น

ต่อมาเครื่องผู้ใช้บริการจะทำการตรวจสอบสิทธิ์ของตน โดยทำการส่งหมายเลขเพื่อบ่งบอกวิธีในการพิสูจน์สิทธิ์และทำการพิสูจน์สิทธิ์ตามแบบที่ได้ส่งไปเซิร์ฟเวอร์

เมื่อทำการตรวจสอบและพิสูจน์สิทธิ์แล้วว่าถูกต้อง เครื่องผู้ใช้บริการจะทำการส่งตัวเลขเพื่อทำการเตรียมตัวสำหรับการเรียกใช้เชลล์ (Shell) และทำการเรียกใช้เชลล์เพื่อที่จะเข้าไปทำการติดต่อในส่วนอินเตอร์แอ็กทีฟเซสชันโหมด (Interactive Session Mode) ซึ่งการทำงานในโหมดนี้เป็นโหมดในการแลกเปลี่ยนข้อมูลกันและกันระหว่างผู้ที่ใช้บริการกับเครื่องผู้ให้บริการและจะสิ้นสุดการติดต่อเมื่อเครื่องผู้ให้บริการส่งคำสั่งเลิกบริการ (Exit Status) ของโปรแกรมไปยังเครื่องผู้ใช้บริการ



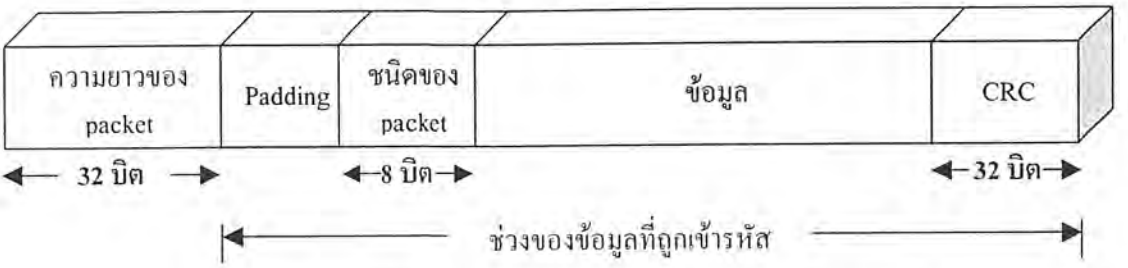
รูปที่ 5-1 แสดงขั้นตอนของเซ็กเคียวเชลล์

5.2 ลักษณะและโพรโตคอลของข้อมูล

จากภาพรวมของโพรโตคอลที่ได้กล่าวมาข้างต้น ทำให้เราทราบวิธีการทำงาน ขั้นตอนการติดต่อและรูปแบบข้อความที่ใช้ติดต่อกัน แต่ก่อนที่เราจะศึกษาขั้นตอนการทำงานอย่างละเอียด เราจะมาทำความรู้จักกับคำที่ได้กล่าวมาข้างต้นก่อนว่าแต่ละตัวคืออะไร ทำหน้าที่อย่างไร และข้อความที่ใช้มีอะไรบ้าง

5.2.1 Binary Packet Protocol

ภายหลังจากการทำการตรวจสอบเวอร์ชันของกันและกัน ทั้งเครื่องให้บริการและเครื่องผู้ใช้บริการจะทำการส่งแพ็กเก็ตที่มีรูปแบบเฉพาะ ซึ่งเรียกว่า ***Binary Packet Protocol*** รูปแบบของแพ็กเก็ตเป็นดังนี้



รูปที่ 5-2 แสดงฟิลด์ต่างๆ แพ็กเก็ตของระบบ Binary Packet Protocol

ความยาวของแพ็กเก็ต : มีขนาด 32 บิตซึ่งเป็นจำนวนเต็มบวก แบ่งออกเป็น 8 บิต/ไบต์ บิตแรกเป็นบิตที่มีค่าสูงสุด ความยาวของแพ็กเก็ต จะไม่นับรวมฟิลด์ของความยาวของแพ็กเก็ตและ padding ความยาวสูงสุดของแพ็กเก็ต คือ 26244 ไบต์

padding : เป็นข้อมูลที่สุ่มขึ้น มีขนาด 1 – 8 ไบต์ (จะมีค่าเป็น 0 ถ้าไม่มีการเข้ารหัสข้อมูล) จำนวนของ padding จะมีขนาด (8 – (ความยาวแพ็กเก็ต % 8)) ไบต์ (% คือ modulo) การที่มีการ padding ช่วยในการหาข้อมูล (plain text) ได้ยากขึ้น

ชนิดของแพ็กเก็ต : จะมีขนาด 8 บิต ค่า 255 กันไว้สำหรับใช้ในอนาคต

ข้อมูลที่จะส่ง : ข้อมูล (มีลักษณะเป็นแบบเลขฐานสอง) ที่จะส่ง โดยขึ้นกับชนิดของแพ็กเก็ต จำนวนของข้อมูลมีขนาดเท่ากับ ความยาวของแพ็กเก็ตลบ 5

CRC : เป็นข้อมูลที่ใช้ตรวจสอบ โดยมีขนาด 32 บิต โดยนำเอา padding , ชนิดของแพ็กเก็ต และฟิลด์ข้อมูล มาทำ CRC (Cyclic Redundancy Check) ด้วยโพลีโนเมียล 0xedb88320 โดย CRC นี้จะถูกคำนวณก่อนการเข้ารหัส

แพ็กเก็ตนี้ ยกเว้นฟิลด์ของความยาว จะถูกเข้ารหัสโดยอัลกอริทึมที่เลือกใช้ โดยความยาวของส่วนที่ถูกเข้ารหัส (padding + ชนิดของแพ็กเก็ต + ข้อมูล + CRC) จะเป็นจำนวนเท่าของ 8 ไบต์

5.2.2 การเข้ารหัสแพ็กเก็ต (Packet Encryption)

แพ็กเก็ตของโพรโตคอลนี้รองรับการเข้ารหัสได้หลายชนิด โดยตั้งแต่เริ่มต้นการติดต่อ ตัวเซิร์ฟเวอร์จะส่งบิตมาร์ค (bitmask) ของวิธีการเข้ารหัสแบบต่างๆ ที่เครื่องให้บริการรองรับ ตัวเครื่องผู้ใช้บริการจะเลือกวิธีการในการเข้ารหัสข้อมูล และสร้างเซสชันคีย์ ขนาด 256 บิต และทำการส่งไปให้เครื่องผู้ให้บริการ

วิธีการเข้ารหัสที่รองรับและโค้ดของวิธีการเข้ารหัสแต่ละแบบคือ

ชนิดแพ็กเก็ต	หมายเลขแทนชนิดของการเข้ารหัส	ชนิดของการเข้ารหัส
SSH_CIPHER_NONE	0	No encryption
SSH_CIPHER_IDEA	1	IDEA in CFB mode
SSH_ CIPHER_DES	2	DES in CBC mode
SSH_ CIPHER_3DES	3	Triple – DES in CBC mode
SSH_ CIPHER_TSS	4	An experimental stream cipher
SSH_ CIPHER_RC4	5	RC4

ตารางที่ 5-1 แสดงวิธีการเข้ารหัสที่เซ็กเคียวเซลล์รองรับ

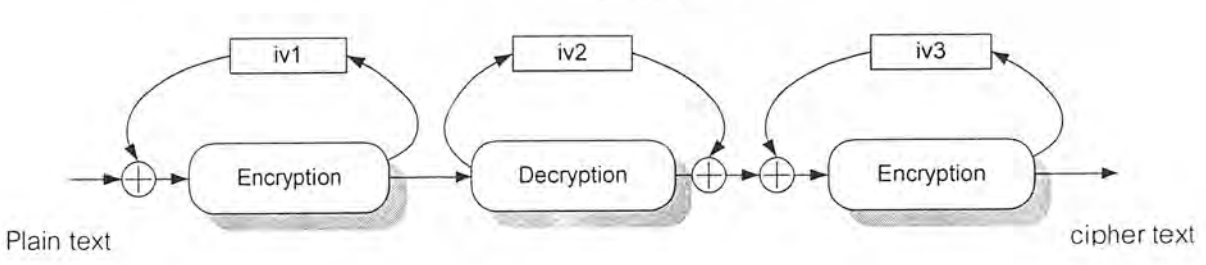
SSH_CIPHER_DES

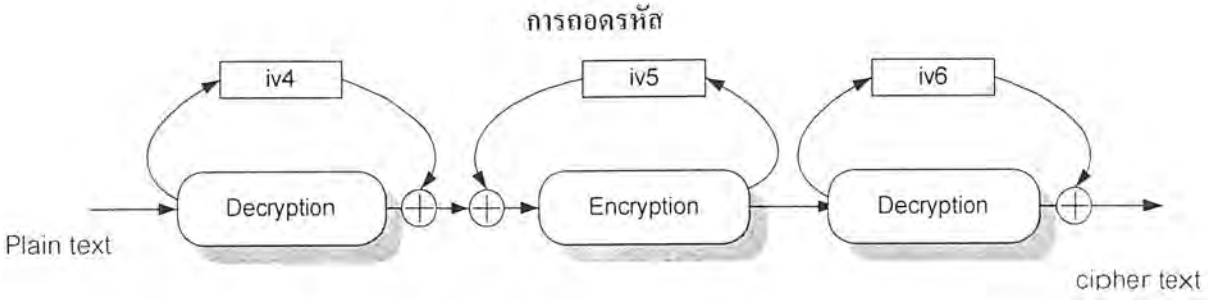
กุญแจในการเข้ารหัสนำมาจาก 8 บิตแรกของเซสชันคีย์ และ บิตที่มีค่าน้อยสุดของแต่ละไบต์จะถูกตัดทิ้งไป กุญแจที่ได้จะมีขนาด 56 บิต การเข้ารหัสแบบ DES ที่ใช้นี้จะเป็นโหมด CBC ซึ่ง iv (initialization vector) ในการเริ่มต้นถูกกำหนดให้เป็น 0 ทั้งหมด (ซึ่งได้อธิบายละเอียดในบทการเข้ารหัสข้อมูล)

รูปที่ 5-3 แสดงการเข้ารหัสและถอดรหัสแบบ DES โหมด CBC

SSH_CIPHER_3DES

วิธีการของ triple-DES คือจะมีการใช้ DES_CBC ที่ไม่เกี่ยวข้องกัน 3 ตัว ซึ่ง iv ที่ไม่เกี่ยวข้องกัน





รูปที่ 5-4 แสดงวิธีการเข้ารหัสและถอดรหัสแบบ 3DES โหมด CBC

สำหรับการเข้ารหัส ข้อมูลจะถูกเข้ารหัสด้วยไซเฟอร์แรกและถอดรหัสด้วยไซเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกเข้ารหัสอีครั้งด้วยไซเฟอร์ 3 ส่วนการถอดรหัส ข้อมูลจะถูกถอดรหัสด้วยไซเฟอร์แรกและเข้ารหัสด้วยไซเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกถอดรหัสอีครั้งด้วยไซเฟอร์ 3 โดยกุญแจที่ใช้สำหรับไซเฟอร์แรกจะถูกนำมาจาก 8 ไบต์แรกของเซสชันคีย์ และกุญแจสำหรับไซเฟอร์ที่ 2 มาจาก 8 ไบต์ถัดไปของเซสชันคีย์ และกุญแจสำหรับไซเฟอร์ที่ 3 จะเป็น 8 ไบต์ถัดไปของเซสชันคีย์เช่นกัน (ค่า iv จะไม่เกี่ยวข้องกัน)

5.2.3 ชนิดของข้อมูลที่ถูกเข้ารหัส (Data Type Encoding)

ฟิลด์ข้อมูลของแต่ละแพ็กเก็ตจะเก็บข้อมูลที่เข้ารหัสตามวิธีที่ได้อธิบายไป ซึ่งจะประกอบรายการของข้อมูลหลายๆ ตัว แต่ละรายการจะมีรหัสในแต่ละแบบและถูกนำมาใช้โดยการนำแต่ละรายการมาต่อกัน

ข้อมูลแต่ละชนิดจะถูกเก็บตามนี้คือ

- **8 bit byte** เป็นข้อมูลที่บรรจุในไบต์ๆ เดียว (มีขนาด 8 บิต)
- **32 bit unsigned integer** เก็บข้อมูลโดยแบ่งเป็น 4 ไบต์ เริ่มต้นด้วยบิตที่มีค่าสูงสุด (MSB first)
- **Arbitrary length binary string** ข้อมูล 4 ไบต์แรกจะเป็นความยาวของข้อความ (String) . เริ่มต้นด้วยบิตที่มีค่าสูงสุด (ไม่นับรวมความยาวของตัวมันเอง) ตามด้วยข้อความ (String) ซึ่งเป็นค่าของตัวอักษรเรียงกัน (ไม่นับรวมตัวอักษรปิดข้อความ)
- **Multiple – precision integer** ข้อมูล 2 ไบต์แรกเป็นจำนวนของบิตในรูปแบบของตัวเลข โดยเริ่มต้นด้วยบิตที่มีค่าสูงสุด (ตัวอย่างเช่น ค่า 0x00012345 จะมี 17 บิต) ค่า 0 จะมีจำนวน 0 บิต ซึ่งเป็นการอนุญาตให้จำนวนของบิตสามารถมากกว่าจำนวนบิตจริงของเลข ข้อมูลบอกจำนวนของบิตจะตามด้วยข้อมูลค่าของเลขจำนวนเต็มซึ่งมีการเก็บข้อมูลขนาด (จำนวนบิต + 7)/ 8 ไบต์ โดยเริ่มต้นด้วยบิตที่มีค่าสูงสุด

5.2.4 หมายเลขทีซีพี/ไอพีพอร์ต (TCP/IP Port Number & Other option)

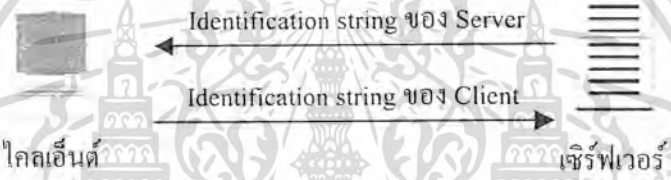
ตัวผู้ให้บริการจะรอติดต่อบนโพรโตคอลทีซีพี/ไอพีพอร์ตที่ 22 ตัวผู้ใช้บริการจะใช้พอร์ตใดของตนก็ได้ในการติดต่อเข้ากับเครื่องผู้ให้บริการ แต่ถ้าตัวเครื่องใช้บริการต้องการจะใช้การติดต่อแบบ .rhosts หรือ /etc/hosts.equiv จะต้องทำการติดต่อจากพอร์ตที่ถูกสงวนไว้ (เบอร์พอร์ตที่ต่ำกว่าเบอร์ 1024)

5.3 รายละเอียดขั้นตอนการติดต่อ

5.3.1 ช่วงตรวจสอบเวอร์ชัน (Protocol Version Identification)

เมื่อผู้ใช้บริการทำการติดต่อไปยังเครื่องให้บริการได้แล้ว เครื่องให้บริการจะส่งข้อความเฉพาะที่ใช้ในการตรวจสอบเวอร์ชัน ที่เรียกว่า version identification sting ซึ่งจะมีรูปแบบ คือ **"SSH-< protocolmajor >.<protocolminor > - <version > \n "** ซึ่ง < protocolmajor > และ <protocolminor > คือ ตัวเลขจำนวนเต็มที่บอกเวอร์ชันของโพรโตคอลที่ใช้ ส่วน < version > เป็นเวอร์ชันของซอฟต์แวร์ที่ฝั่งผู้ใช้บริการใช้ (ไม่เกิน 40 ตัวอักษร)

ส่วนผู้ใช้บริการเมื่อได้รับข้อความมาก็จะทำการแปล ถ้าไม่รองรับเวอร์ชันของโพรโตคอลก็จะทำการปิดการติดต่อ แต่ถ้ารองรับเครื่องผู้ใช้บริการก็จะส่งข้อความที่เป็นข้อมูลของตัวเองลักษณะเดียวกับที่เครื่องผู้ให้บริการส่งมากลับไป เมื่อเครื่องผู้ให้บริการได้รับก็จะตีความหมาย ถ้าเป็นเวอร์ชันของโพรโตคอลที่รองรับถ้าตรงกันไหมก็จะส่งข้อความแรกกลับไป ซึ่งข้อความจะอยู่ในรูปของ binary packet protocol



รูปที่ 5-5 แสดงการตรวจสอบเวอร์ชัน Identification

5.3.2 ช่วยการแลกเปลี่ยนกุญแจ (Key Exchange)

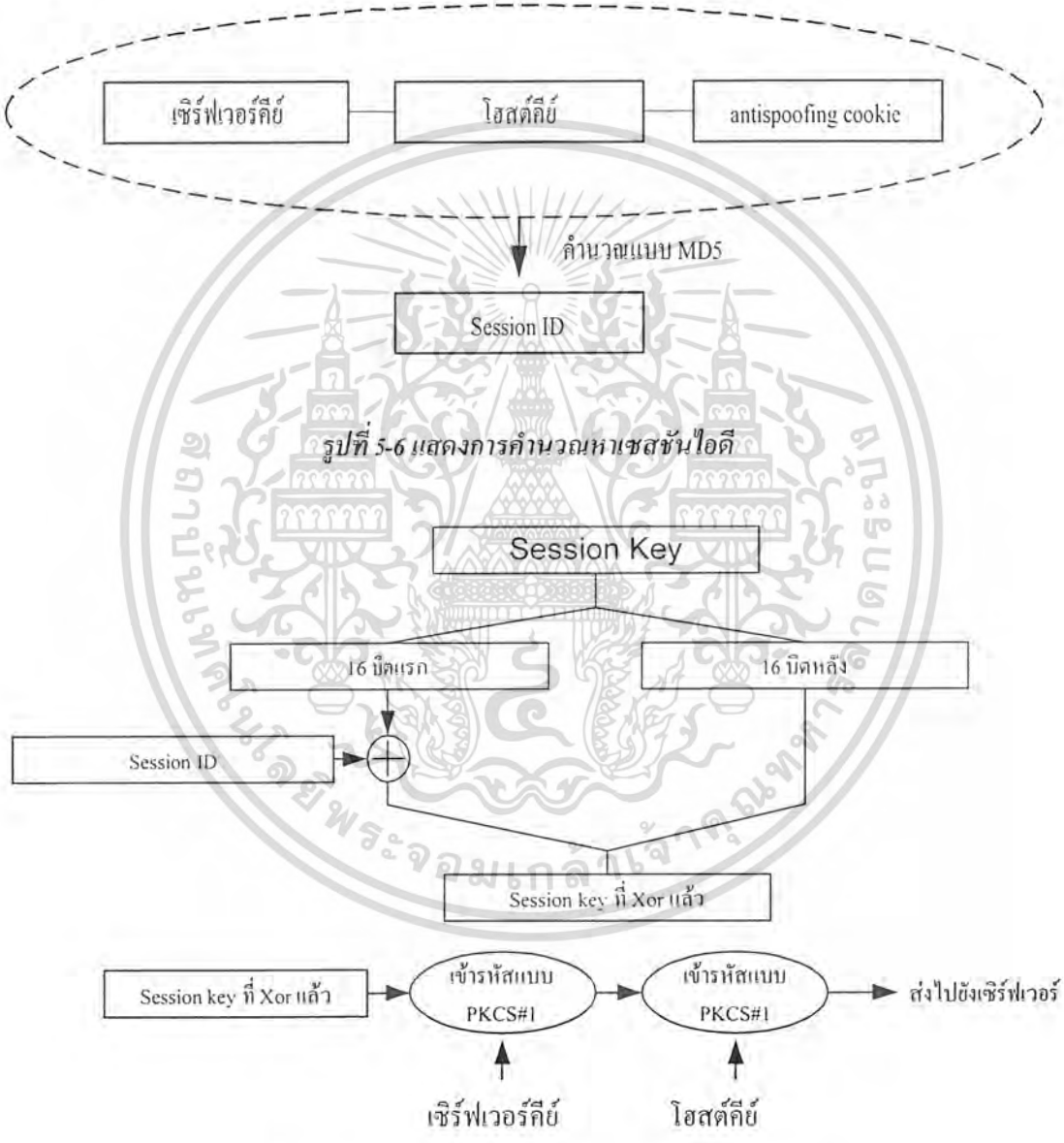
หลังจากที่มีการตรวจสอบเวอร์ชันว่ารองรับกันได้แล้ว ก็จะเข้าสู่การแลกเปลี่ยนกุญแจ เนื่องจากการเข้ารหัสของข้อมูลต่างๆ ที่ใช้ติดต่อกันทั้งสองฝ่ายจะต้องมีกุญแจเดียวกันในการเข้ารหัสและถอดรหัสของข้อมูลนั้นๆ ซึ่งจะต้องมีวิธีการทำให้ทั้งคู่มีกุญแจเดียวกัน โดยที่ผู้อื่นไม่ทราบว่ากุญแจนี้ คืออะไร ซึ่งมีขั้นตอน คือ

หลังจากที่เซิร์ฟเวอร์ได้รับ Identification String จากไคลเอ็นต์และตรวจสอบว่ารองรับแล้ว เซิร์ฟเวอร์จะส่งข้อความในรูปแบบ Binary Packet Protocol ที่เป็นชนิดของ SSH_SMSG_PUBLIC_KEY ซึ่งจะประกอบด้วย

- **ค่าที่สุ่มขึ้นมาขนาด 64 บิต (คุกกี้)** เพื่อให้ไคลเอ็นต์ส่งข้อมูลนี้กลับมา ทำให้ ยากต่อการทำการปลอมแปลง IP
- **โฮสต์คีย์ (host key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ มีขนาด 1024 บิต
- **เซิร์ฟเวอร์คีย์ (server key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ที่มีการสร้างใหม่ทุกๆ ชั่วโมง มีขนาด 768 บิต
- **การเข้ารหัสที่รองรับ (supported ciphers)** เป็นชนิดของการเข้ารหัสที่เซิร์ฟเวอร์รองรับ
- **การพิสูจน์สิทธิที่รองรับ (supported authentication method)** เป็นวิธีของการพิสูจน์สิทธิ์ที่เซิร์ฟเวอร์รองรับ

เมื่อไคลเอ็นต์ได้รับข้อความ SSH_SMSG_PUBLIC_KEY ไคลเอ็นต์จะสร้างกุญแจ (เรียกว่าเซสชันคีย์) ที่ใช้สำหรับเข้ารหัสแพ็กเก็ตต่างๆ ขึ้นมาโดยการสุ่มค่าขึ้น (กุญแจจะมีขนาด 32 ไบต์) เพื่อจะทำการส่งไปให้เซิร์ฟเวอร์ โดยก่อนที่จะทำการส่งจะนำเอาเซสชันคีย์ 16 บิตแรก มาเอ็กซ์คลูซีฟออร์กับเซสชันไอดี(มีขนาด 16 ไบต์) แล้วนำมาต่อรวมกับ 16 บิตหลังของเซสชันคีย์ จากนั้นจึงทำการเข้ารหัสแบบ PKCS#1 (ซึ่งเป็นการเข้ารหัสแบบ RSA) 2 ครั้งด้วยโฮสต์คีย์ และ เซิร์ฟเวอร์คีย์ โดยใช้กุญแจที่สั้นกว่าก่อน (ซึ่งคือ เซิร์ฟเวอร์คีย์)

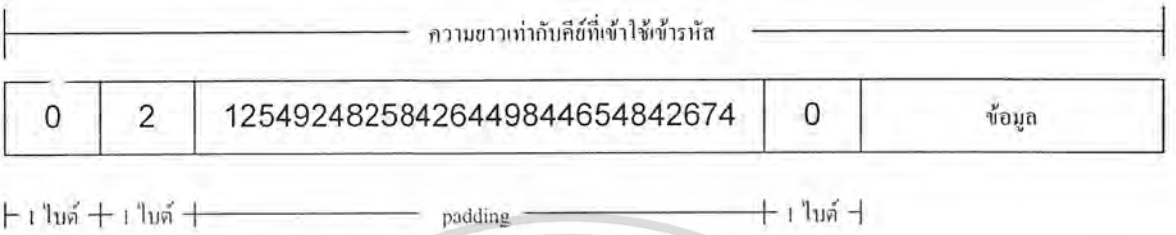
สำหรับของเซสชันไอดีเป็นค่าที่ทั้งสองฝั่งคำนวณหาได้ โดยการนำเอาโฮสต์คีย์ต่อด้วยเซิร์ฟเวอร์คีย์ต่อด้วยคุกกี้แล้วนำมาคำนวณด้วย อัลกอริทึมแบบ MD5 ซึ่งทั้งสองฝั่งจะได้ค่านี้เช่นเดียวกัน



รูปที่ 5-6 แสดงการคำนวณหาเซสชันไอดี

รูปที่ 5-7 แสดงการเข้ารหัสเซสชันคีย์ก่อนส่งให้เซิร์ฟเวอร์

สำหรับการเข้ารหัสแบบ PKCS#1 นั้นใช้มาตรฐานของ RSA มีลักษณะของการเข้ารหัสเหมือน RSA โดยให้ข้อมูลที่จะทำการเข้ารหัสนั้น ให้ไบต์แรกมีค่าเป็น 0 ไบต์ถัดมามีค่าเป็น 2 (ตามมาตรฐาน) แล้วตามด้วยค่าที่สุ่มขึ้นและไม่เท่ากับ 0 ในตำแหน่งถัดมาที่ไม่ได้ใช้ ตามด้วยไบต์ 0 และตามด้วยข้อมูลที่ต้องการจะถูกเข้ารหัส แล้วนำข้อมูลนี้มาทำการเข้ารหัสแบบ RSA (C = p^c mod n โดย C คือผลการเข้ารหัส , p คือ ข้อมูลที่จะทำการเข้ารหัส , c คือ พลับบลิคคีย์เอ็กซ์โปเนนท์ และ n คือ พลับบลิคคีย์โมดูลัส)

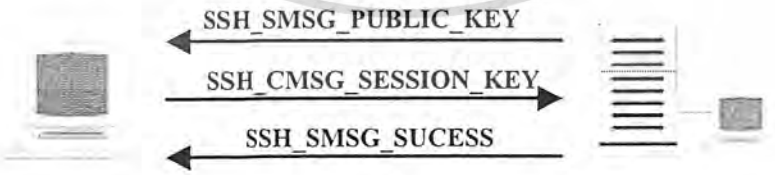


รูปที่ 5-8 แสดงตัวอย่างของข้อมูลที่จะทำการเข้ารหัส PKCS#1

หลังจากที่ได้ข้อมูลที่เป็นคีย์ (ที่ทำการเข้ารหัสข้างต้น) ก็จะส่งแพ็กเก็ตที่มีชนิดเป็น SSH_CMSG_SESSION_KEY โดยมีข้อมูล ต่างๆ ดังต่อไปนี้ ตามลำดับ

- ชนิดของการเข้ารหัส
- คุกกี้ที่ได้จากเซิร์ฟเวอร์
- เซสชันคีย์ที่ถูกเข้ารหัสข้างต้น
- โพรโตคอลเวอร์ชันที่ใช้

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเก็ต SSH_CMSG_SESSION_KEY ก็จะถอดรหัสและคำนวณหาเซสชันคีย์ออกมา หากลักษณะของข้อมูลถูกต้องก็จะส่ง แพ็กเก็ตกลับไปว่าเรียบร้อยแล้ว โดยมีชนิดว่า SSH_SMSG_SUCCESS ซึ่งข้อมูลนี้จะถูกทำการเข้ารหัสด้วยวิธีการตามที่ไคลเอ็นต์เลือกมา



รูปที่ 5-9 ขั้นตอนการส่งของแพ็กเก็ตต่างๆ ในช่วงการแลกเปลี่ยนคีย์

5.4 ช่วงตรวจสอบชื่อและพิสูจน์สิทธิ์ (Declare User Name & Authentication Phase)

ไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_CMSG_USER ไปยังเซิร์ฟเวอร์ ซึ่งจะมีชื่อของผู้ใช้ในแพ็กเก็ตนี้ เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบว่ามีผู้ใช้คนนั้นในรายการหรือไม่ และตรวจสอบการตรวจสอบสิทธิ์ที่

ต้องใช้ แล้วจะตอบกลับด้วยแพ็กเก็ตชนิด SSH_SMSG_SUCCESS ถ้าผู้ใช้คนนี้ถูกกำหนดไว้ว่าไม่ต้องการให้มีการพิสูจน์สิทธิ์ (ไม่ต้องใช้รหัสผ่าน) หรือ SSH_SMSG_FAILURE ถ้าผู้ใช้จำเป็นต้องมีการพิสูจน์สิทธิ์

สำหรับในกรณีที่ไม่มีชื่อของผู้ใช้คนนี้ เซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_FAILURE มาเช่นกัน ทำให้ผู้ใช้ไม่ทราบว่าผู้ใช้คนนี้มีอยู่หรือเปล่า

เมื่อไคลเอ็นต์ได้รับแพ็กเก็ต SSH_SMSG_FAILURE ไคลเอ็นต์จะทำการส่งวิธีการในการพิสูจน์สิทธิ์ให้กับเซิร์ฟเวอร์ ซึ่งวิธีการพิสูจน์สิทธิ์ที่รองรับ มีวิธีต่างๆ ดังนี้

- **ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv** โดยฝั่งไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_AUTH_RHOSTS
- **ตรวจสอบสิทธิ์ด้วยวิธี RSA** โดยฝั่งไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_AUTH_RSA
- **ตรวจสอบสิทธิ์จากรหัสผ่าน (password)** โดยฝั่งไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_AUTH_PASSWORD
- **ตรวจสอบสิทธิ์จากไฟล์ .rhosts และวิธี RSA** โดยฝั่งไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_AUTH_RHOSTS_RSA

ถ้าเซิร์ฟเวอร์ได้รับวิธีการพิสูจน์สิทธิ์จากไคลเอ็นต์ หากรองรับวิธีการก็จะส่งแพ็กเก็ตชนิด SSH_SMSG_SUCCESSS แต่หากไม่รองรับก็จะส่งแพ็กเก็ตชนิด SSH_SMSG_FAILURE
เวลาสำหรับการพิสูจน์สิทธิ์จะถูกกำหนดไว้ไม่ให้เกิน 5 นาที (ถ้าหากเกิน เซิร์ฟเวอร์ก็จะทำการยกเลิกการติดต่อ) สำหรับวิธีการตรวจสอบสิทธิ์วิธีต่างๆ มีรายละเอียดดังนี้

5.4.1 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RHOST ตามด้วยชื่อของผู้ใช้ฝั่งไคลเอ็นต์ เมื่อเซิร์ฟเวอร์ที่บนระบบยูนิกซ์ได้รับ ก็จะทำการตรวจสอบจากไฟล์ /etc/host.equiv และจากไฟล์ .rhosts ในไดเรกทอรีของผู้ใช้เอง การติดต่อแบบนี้จะต้องใช้พอร์ตเฉพาะพิเศษ

การวิธีการพิสูจน์สิทธิ์ด้วยระบบที่เชื่อถือฝั่งรีโมต (trusts the remote host) สามารถที่จะถูกอ่านของข้อมูลได้โดยวิธีการปลอมแปลงเลขไอพี แต่จะถูกป้องกันไว้จากโพรโตคอล (เพราะทุกๆ แพ็กเก็ตจะถูกเข้ารหัสอยู่)

5.4.2 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv และ RSA

เป็นวิธีที่เพิ่มจากวิธีการตรวจสอบสิทธ์จากไฟล์ .rhosts หรือ /etc/host.equiv โดยมีการพิสูจน์สิทธิ์แบบ RSA ด้วย

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RHOSTS_RSA ตามด้วยชื่อของผู้ใช้และกุญแจสาธารณะของไคลเอ็นต์

โดยเริ่มแรก เซิร์ฟเวอร์จะทำการตรวจสอบตามปกติเช่นเดียวกับตรวจสอบจากไฟล์ .rhosts และ /etc/host.equiv ถ้าไม่ถูกต้องก็จะส่ง SSH_SMSG_FAILURE กลับมา ตรงกันข้ามมันจะตรวจสอบกุญแจสาธารณะในตัว ถ้าหากไม่พบก็จะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อ

ถ้าเซิร์ฟเวอร์รู้โฮสต์คีย์ของเครื่องไคลเอ็นต์ มันจะทำการตรวจสอบจากโฮสต์คีย์ที่ได้มาว่าตรงกันหรือไม่ ถ้าไม่ ก็จะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อเช่นกัน

ถ้าหากถูกต้องเซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_AUTH_RSA_CHALLENGE ที่บรรจุด้วย challenge ที่ถูกเข้ารหัสสำหรับไคลเอ็นต์ โดย challenge จะมีขนาดเป็น 32 บิตที่ถูกสุ่มขึ้นมา แล้วทำการจัดข้อความและเข้ารหัสด้วย กุญแจสาธารณะของไคลเอ็นต์ (วิธีการเช่นเดียวกับที่เข้ารหัสเซสชันคีย์)

ไคลเอ็นต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับเซสชันไอดีที่มีอยู่ มาทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge +เซสชันไอดี => ทำ MD5) ถ้าถูกต้องก็จะส่ง SSH_SMSG_SUCCESS ตรงกันข้ามก็จะส่ง SSH_SMSG_FAILURE และปฏิเสธการพิสูจน์สิทธิ์นี้

5.4.3 ตรวจสอบสิทธิ์ด้วยวิธี RSA

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RSA ตามด้วยพลับบลิคคีย์โมดูลัสของไคลเอ็นต์ (ต่างจากวิธี .rhosts + RSA ที่ส่งค่ากุญแจสาธารณะทั้งหมดไป แต่วิธีนี้ส่งเฉพาะค่า n ที่เป็นโมดูลัส ของพลับบลิคคีย์)

เซิร์ฟเวอร์จะตอบกลับทันทีด้วย SSH_SMSG_FAILURE ถ้ามันไม่พบคีย์ของการพิสูจน์สิทธิ์ของไคลเอ็นต์นี้ในตัวมัน (ค่ากุญแจสาธารณะทั้งหมดของไคลเอ็นต์) ตรงกันข้ามมันจะสร้าง challenge มาทำการเข้ารหัสด้วยกุญแจสาธารณะของไคลเอ็นต์

ไคลเอ็นต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับเซสชันไอดีที่มีอยู่ มาทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge +เซสชันไอดี => ทำ MD5) ถ้าถูกต้องก็จะส่ง SSH_SMSG_SUCCESS ตรงกันข้ามก็จะส่ง SSH_SMSG_FAILURE และปฏิเสธการพิสูจน์สิทธิ์นี้

5.4.4 ตรวจสอบสิทธิ์จากรหัสผ่าน password

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_PASSWORD ตามด้วยข้อความด้วยข้อความที่เป็นรหัสผ่าน (ข้อความนี้จะยังไม่มีการทำอะไร แต่จะถูกทำการเข้ารหัสด้วยกลไกของแพ็กเก็ตโดยอัตโนมัติ)

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบรหัสผ่าน และส่ง SSH_SMSG_SUCCESS ถ้าการพิสูจน์สิทธิ์ถูกต้อง และถ้าผิดพลาดจะส่ง SSH_SMSG_FAILURE

5.5 ช่วงการเตรียมการ (Preparatory Operation)

หลังจากที่ทำการพิสูจน์สิทธิ์แล้ว เซิร์ฟเวอร์จะรอคำร้องขอจากไคลเอ็นต์ และทำการจัดการกับคำสั่งที่เข้ามา และเซิร์ฟเวอร์จะตอบกลับด้วย SSH_SMSG_SUCCESS เมื่อคำร้องขอที่เข้าถูกจัดการเสร็จสิ้น แต่ในกรณีตรงกันข้าม เมื่อคำร้องขอที่เข้ามาไม่สามารถจัดการได้หรือจัดการไม่สำเร็จก็จะส่ง SSH_SMSG_FAILURE ซึ่งข้อความช่วงนี้เป็นการเตรียมการสำหรับในช่วงต่อไป

5.6 อินเตอร์แอ็กทีฟเซสชันและการแลกเปลี่ยนข้อมูล (Interactive Session and Exchange of Data)

ในช่วงอินเตอร์แอ็กทีฟเซสชัน ข้อมูลทุกตัวจะถูกเขียนโดยเชลล์หรือคำสั่งที่ทำงานอยู่บนเครื่องเซิร์ฟเวอร์เพื่อส่งออกไปยังส่วนแสดงผลหรือส่วนแสดงข้อผิดพลาดบนเครื่องไคลเอ็นต์ ส่วนอินพุตจะมาจากส่วนรับข้อมูลบนเครื่องไคลเอ็นต์ ซึ่งจะถูกส่งไปยังโปรแกรมบนเครื่องเซิร์ฟเวอร์

การแลกเปลี่ยนข้อมูลทั้งหมดจะเป็นอะซิงโครนัส (asynchronous) โดยการส่งสามารถเกิดขึ้นได้ทุกเวลา และไม่ต้องมีการตอบสนอง (ทีซีพี/ไอพีปกติจัดการการสร้างความได้น่าเชื่อถืออยู่แล้วและแพ็กเก็ตโพรโตคอลจะทำการป้องกันการเข้ามายุ่งหรือทำการปลอมแปลงไอพี)

เมื่อไคลเอ็นต์ได้รับ EOF จากส่วนรับข้อมูล มันจะส่ง SSH_CMSG_EOF การแลกเปลี่ยนข้อมูลและอินเตอร์แอ็กทีฟโหมดจะสิ้นสุดลงเมื่อเซิร์ฟเวอร์ส่ง SSH_SMSG_EXITSTATUS เพื่อที่จะแสดงว่าการติดต่อกับตัวไคลเอ็นต์ได้สิ้นสุดลง ส่วนไคลเอ็นต์หรือเซิร์ฟเวอร์จะสามารถยกเลิกการติดต่อ เมื่อไรก็ได้โดยการส่งข้อความ SSH_MSG_DISCONNECT หรือทำการปิดการเชื่อมต่อ (close the connection)

5.7 การยกเลิกการติดต่อ (Termination of the Connection)

โดยปกติการยกเลิกการติดต่อจะเริ่มโดยเซิร์ฟเวอร์ซึ่งจะทำการส่ง SSH_SMSG_EXITSTATUS หลังจากที่โปรแกรมจบลง ตัวไคลเอ็นต์จะทำการตอบสนองกับข้อความดังกล่าว โดยการส่ง SSH_CMSG_EXIT_CONFIRMATION และทำการปิดซ็อกเก็ตของตัวเองลง ส่วนเซิร์ฟเวอร์เมื่อได้รับข้อความจึงค่อยทำการปิดซ็อกเก็ตลง เป้าหมายสำคัญสำหรับสำหรับการยืนยันในการยกเลิกคือ ในบางระบบอาจจะสูญเสียข้อมูลที่ส่งไปก่อนหน้านี้เมื่อซ็อกเก็ตถูกปิด และการยกเลิกในฝั่งไคลเอ็นต์ก่อนทำให้เกิด TCP/IP TIME_WAIT [RFC 0793] ซึ่งทำให้เซิร์ฟเวอร์รอรับคำตอบจากฝั่งไคลเอ็นต์ที่ไม่ได้ใช้ ทำให้เซิร์ฟเวอร์ต้องสูญเสียรีซอร์ส

ถ้าในระหว่างโปรแกรมมีสัญญาณที่จะทำให้ยกเลิก ฝั่งตัวเซิร์ฟเวอร์จะส่งแพ็กเก็ตชนิด SSH_MSG_DISCONNECT พร้อมกับข้อความที่เกี่ยวข้อง ถ้าการติดต่อถูกปิด file descriptor ที่ชี้ไปยังโปรแกรมจะถูกปิดลงและเซิร์ฟเวอร์จะ exit แต่ถ้าโปรแกรมรันบน tty ตัวเคอร์เนลจะส่งสัญญาณ SIGHUP เมื่อฝั่ง pty master ถูกปิด

5.8 การใช้งานการฟอร์เวิร์ดพอร์ตของเซ็กเคียวเชลล์

การฟอร์เวิร์ดพอร์ตความสามารถที่เป็นประโยชน์มากอีกอย่างของเซ็กเคียวเชลล์ ทำให้เราสามารถใช้โปรแกรมประยุกต์อื่นๆ ที่ไม่มีความปลอดภัย คือไม่มีการเข้ารหัสข้อมูล เป็นการสื่อสารที่มีความปลอดภัยด้วยการเข้ารหัสของเซ็กเคียวเชลล์ แบ่งออกเป็น 2 ชนิดคือ

- เอ็กซ์ ฟอร์เวิร์ดดิง (X forwarding)
- ทีซีพี/ไอพี พอร์ตฟอร์เวิร์ดดิง (TCP/IP port forwarding)

5.8.1 เอ็กซ์ ฟอร์เวิร์ดดิง (X forwarding)

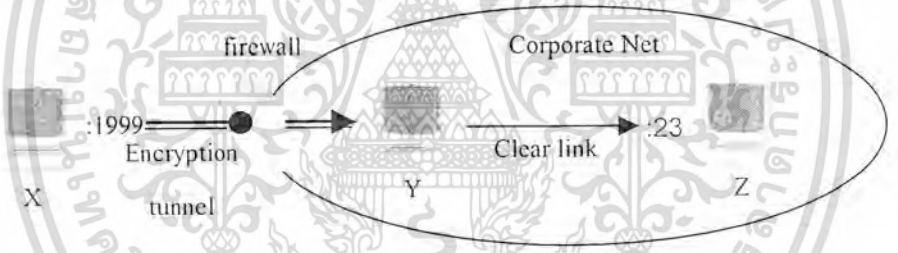
สำหรับในระบบยูนิกซ์นั้นสามารถใช้การติดต่อกับยูเซอร์แบบกราฟิกได้ จากการเรียกใช้ เอ็กซ์วินโดว์ (X window) ซึ่งเมื่อเรียกใช้เอ็กซ์วินโดว์แล้วการติดจะใช้ เอ็กซ์ 11 โพรโตคอลแทนและเซ็กเคียวเชลล์โพรโตคอลนั้นสามารถช่วยเข้ารหัสข้อมูลที่ถูกรับส่งโดยเอ็กซ์11 โพรโตคอลนี้ได้โดยใช้หลักการเดียวกับการทำทีซีพี/ไอพีฟอเวิร์ด ซึ่งจะอธิบายต่อไป

5.8.2 ทีซีพี/ไอพี พอร์ตฟอร์เวิร์ดดิง (TCP/IP port forwarding)

ปกติแล้วของโพรโตคอลที่ทำงานอยู่บนโพรโตคอลทีซีพี/ไอพี เช่น เอชทีทีพี ,เอฟทีพี ,เทลเน็ต ,เอสเอ็มทีพี นั้นจะถูกแบ่งแยกโดยหมายเลขพอร์ตและเซ็กเคียวเซลล์โพรโตคอล ก็เช่นกัน โดยมันทำงานที่พอร์ตหมายเลข 22 เมื่อเราเรียกใช้งาน เซ็กเคียวเซลล์โพรโตคอลแล้วการส่งข้อมูลจะถูกเข้ารหัสแต่ก็เฉพาะข้อมูลของเซ็กเคียวเซลล์เท่านั้น แต่ข้อมูลของโพรโตคอลอื่นๆนั้นไม่มีการเข้ารหัสเลยในการส่งข้อมูลระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ แต่ว่าเซ็กเคียวเซลล์โพรโตคอลนั้นสามารถช่วยให้โพรโตคอลต่างๆ ที่ทำงานอยู่บนโพรโตคอลทีซีพี/ไอพีมีการเข้ารหัสข้อมูลได้โดยการเรียกใช้บริการทีซีพี/ไอพีฟอเวิร์ด ซึ่งแบ่งออกได้เป็น 2 ลักษณะ คือ การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ หรือ มีการสร้างพอร์ตที่ฝั่งของเซ็กเคียวเซลล์เซิร์ฟเวอร์

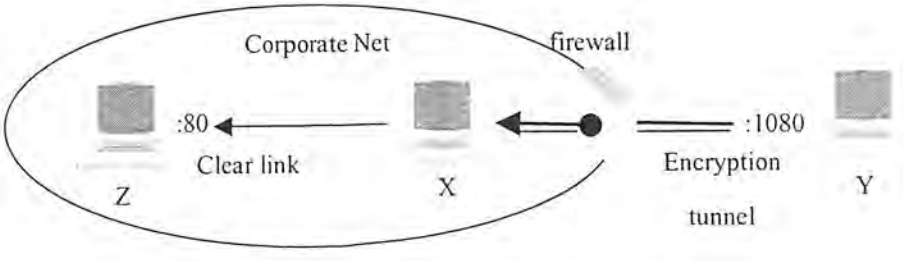
- การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ (Local) สามารถอธิบายง่ายๆ ได้ด้วยตัวอย่างดังรูปที่ 5-10 เป็นการฟอร์เวิร์ดพอร์ตจากโลคอลโฮสต์ไปยังเครื่องรีโมต คือ พิจารณาโฮสต์ แห่ง คือ x y และ z โฮสต์ x นั้นเป็นเครื่องไคลเอ็นต์ และสร้างการเชื่อมต่อกับโฮสต์ y ซึ่งเป็นเครื่องเซิร์ฟเวอร์ผ่านเซ็กเคียวเซลล์ พอร์ตบนเครื่อง x ที่ 1999 จะถูกสร้างจะส่งต่อไปยังเครื่องอีกเครื่องนั้นคือ โฮสต์ z ที่พอร์ต 23 พอร์ต 1999 นี้จะส่งผ่านช่องทางที่มีการเข้ารหัสไปยังเครื่อง y ซึ่งจะส่งต่อไปที่เครื่อง z ที่พอร์ต 23 โดยที่ไม่มีการเข้ารหัสในการเชื่อมต่อส่วนนี้

ด้วยเหตุนี้เอง ถ้าใช้คำสั่ง telnet localhost 1999 จะมีผลเหมือนกับการเชื่อมต่อเพื่อสร้างการติดต่อกับ z โดยผ่านช่องทางที่มีความปลอดภัยของเครื่อง y



รูปที่ 5-10 แสดงการทำงานของการฟอร์เวิร์ดพอร์ตแบบโลคอล

- การสร้างพอร์ตที่ฝั่งของเซ็กเคียวเซลล์เซิร์ฟเวอร์ (remote) เป็นการฟอร์เวิร์ดพอร์ตของรีโมตโฮสต์ไปยังเครื่องโลคอล คือ จะฟอร์เวิร์ดพอร์ตของเครื่องรีโมต ซึ่งคือ y พอร์ตที่ 1080 ไปยังโฮสต์ z พอร์ต 80 บนช่องทางที่มีการเข้ารหัสของ x ดังรูปที่ 5-11



รูปที่ 5-11 แสดงการทำงานของการส่งต่อพอร์ตแบบรีโมต

ซึ่งสามารถถูกมองเป็น pseudo proxy capability หรือ tunneling IP และมีผลกระทบในแง่ของจุดบกพร่องทางด้านปลอดภัยในไฟล์วอลล์ ซึ่งจะอนุญาตให้เซ็กเคียวเชลล์ผ่านไปได้ ดังนั้นไฟล์วอลล์หรืออุปกรณ์ที่ทำหน้าที่ใกล้เคียงกัน จะคิดว่ามันกำลังอนุญาตให้การติดต่อของเซ็กเคียวเชลล์ผ่านเข้าไป แต่จะมีผลที่จริงแล้วเหมือนกับการอนุญาตให้มีการติดต่อกับอย่างอิสระระหว่างระบบภายในกับระบบภายนอก

บทที่ 6
ภาษาจาวา

6.1 ประวัติของภาษาจาวา

หลายปีก่อนบางคนเชื่อว่าภาษาสำหรับโปรแกรมเป็นวิชาที่ตายแล้ว นั่นอาจเป็นเพราะว่าไม่มีอะไรที่ภาษาซีทำไม่ได้ และคงต้องอีกหลายปีที่คนส่วนใหญ่จะเข้าใจว่าคุณค่าที่แท้จริงของภาษาซีพลัสพลัส แต่เมื่อตอนต้นปี 1996 ภาษาจาวาก็มีชื่อขึ้นหน้าปกวารสารคอมพิวเตอร์ชั้นนำเกือบทุกยี่ห้อทั่วโลก และเป็นข่าวใน CNN อย่างต่อเนื่องนานนับปี ไม่เคยมีภาษาใดที่ได้รับความสนใจเช่นนี้มาก่อน ผู้เชี่ยวชาญบางคนกล่าวว่าภาษาจาวาจะเปลี่ยนวงการคอมพิวเตอร์เหมือนกับที่ล้อเลื่อนเปลี่ยนความเป็นอยู่ของมนุษยชาติ

ภาษาจาวาเริ่มเป็นที่สนใจ เมื่อบริษัท ซัน ไมโครซิสเต็มส์ (Sun Microsystems) ประกาศให้เป็นภาษาสำหรับสร้างโปรแกรมเพื่อใช้งานอินเทอร์เน็ต (Internet) โปรแกรมภาษาจาวาอาจถูกสร้างขึ้นบนคอมพิวเตอร์เครื่องหนึ่ง แล้วนำไปทำงานบนเครื่องคอมพิวเตอร์ต่างระบบได้ โดยไม่ต้องคอมไพล์โปรแกรมใหม่ ความสามารถนี้จะเปลี่ยนโฉมหน้าและบทบาทของอินเทอร์เน็ตอย่างสิ้นเชิง เมื่อเรามีโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ใดๆ ได้ อินเทอร์เน็ตจะกลายเป็นหนึ่งเดียว สิ่งที่ถูกส่งไปมาในอินเตอร์เน็ตไม่จำเป็นต้องแบ่งแยกอีกต่อไป นั่นคือเว็บเพจทั้งหลายจะไม่ใช่เอกสารที่ถูกกระทำ (Passive) แต่จะกลายเป็นเอกสารที่ทำงานได้ (Active) นอกไปจากนี้ภาษาจาวาจะส่งผลกระทบต่อทั้งผู้ผลิตและผู้ใช้ คือถึงเวลาที่บริษัทผู้ผลิตซอฟต์แวร์ทั้งหลายจะต้องเปลี่ยนแปลง หรือยุบแผนกวินโดวส์ , แมคอินทอชและยูนิกซ์ มารวมกันเสียทีเพื่อผลิตโปรแกรมที่ทำงานบนระบบใดๆก็ได้เพียงหนึ่งเดียว สำหรับผู้ใช้ลองนึกดูว่า ถ้าคอมพิวเตอร์ของเราเมื่อเปิดเครื่องเข้าสู่บราวเซอร์ จากนั้นเราสามารถเรียกโปรแกรมที่ต้องการจากเซิร์ฟเวอร์ที่ให้บริการในอินเทอร์เน็ตมาใช้งาน รูปแบบการทำงานของเราก็จะไม่เหมือนเดิมอีกต่อไป นั่นคือไม่จำเป็นต้องมีฮาร์ดดิสก์ (Hard disk) หรือแม้แต่ระบบปฏิบัติการอย่างวินโดวส์ เลยก็ได้

ผลกระทบเหล่านี้ทำให้ภาษาจาวาเป็นที่น่าสนใจอย่างยิ่ง ในขณะที่บริษัทผู้ผลิตซอฟต์แวร์ทั้งหลายก็ให้การต้อนรับภาษานี้ในทิศทางที่ดีเกินความคาดหมาย ประกอบด้วยภาษาจาวามีประสิทธิภาพในการสร้างโปรแกรมอย่างยิ่ง จึงเป็นเครื่องมือชั้นดีสำหรับเทคโนโลยีการสร้างโปรแกรมด้วยคอมโพเน็นต์และวิชวลโปรแกรมมิ่ง เชื่อได้แน่ว่าในอนาคตอันใกล้นี้ อุตสาหกรรมซอฟต์แวร์จะเปลี่ยนโฉมหน้าไปจากเดิมและการเรียนรู้ภาษาจาวาจะเป็นเรื่องที่หลีกเลี่ยงไม่ได้

ในช่วงต้น 1990s ตลาดเครื่องใช้ไฟฟ้า เช่นเครื่องซักผ้า, หม้อหุงข้าว,โทรทัศน์, ไมโครเวฟและอื่นๆ มีมูลค่าสูงกว่าตลาดคอมพิวเตอร์หลายเท่าตัว อีกทั้งระบบคอมพิวเตอร์ขนาดเล็กสำหรับควบคุมเครื่องใช้ไฟฟ้าเหล่านี้ก็ถูกพัฒนาดีขึ้นเรื่อยๆ บริษัท ซัน ไมโครซิสเต็มส์ ซึ่งประสบความสำเร็จในตลาดระบบเครือข่ายคอมพิวเตอร์อย่างมากในตอนนั้น เห็นว่าควรใช้ความได้เปรียบของบริษัท เร่งพัฒนาเทคโนโลยีเพื่อยึดครองตลาดเครื่องใช้ไฟฟ้าที่มีคอมพิวเตอร์ควบคุมไว้ก่อนคนอื่น จึงจัดทีม Green group ขึ้นในปี 1991 มี James

Gosling เป็นหัวหน้า ทำการพัฒนาระบบซอฟต์แวร์สำหรับควบคุมเครื่องใช้ไฟฟ้าขนาดเล็ก พวกเขาสร้างเครื่องต้นแบบที่เรียกว่า Star 7 เป็นระบบรีโมตคอลโทรล ขนาดมือถือ สามารถควบคุมการเคลื่อนไหวของวัตถุโดยใช้นิ้วสัมผัสบนแป้นแสดงผล ระบบนี้ถูกทดสอบใช้การแสดงควบคุมตัว The Duke (ซึ่งต่อมาเป็นตัวนำโชคของภาษาจาวา) ใช้เคลื่อนที่ผ่านกลุ่มของวัตถุในมิติสมมติ

พวกเขาใช้ภาษาซีพลัสพลัสเขียนโปรแกรมของ Star 7 ผลที่ได้เป็นโปรแกรมที่มักจะทำงานผิดพลาดและล้มเหลวบ่อยๆ จนต้องสรุปว่าภาษาซีพลัสพลัสไม่เหมาะสำหรับงานแบบนี้ เพราะมีข้อจำกัดหลายอย่าง นั่นคือเครื่องใช้ไฟฟ้าขนาดเล็กมักมีหน่วยความจำน้อย ไม่เหมาะกับโปรแกรมภาษาซีพลัสพลัส ที่มักมีขนาดใหญ่ อีกทั้งในเครื่องเหล่านี้ไม่มีระบบปฏิบัติการ โปรแกรมจึงไม่สามารถเรียกใช้บริการของระบบปฏิบัติการได้เหมือนบนเครื่องที่มีระบบปฏิบัติการ ดังนั้นความสามารถของภาษาจึงจำกัดไปอย่างมาก เราจึงต้องสร้างโปรแกรมสำหรับทำงานพื้นฐานเอง นอกจากนี้ซีพลัสพลัส ยังเป็นภาษาที่ไม่ปลอดภัย เพราะยอมให้มีการใช้พอยน์เตอร์อย่างไม่มีจำกัด และละเลยการทำการตรวจสอบชนิดข้อมูล โปรแกรมจึงมักมีความผิดพลาดซ่อนอยู่เป็นจำนวนมาก

ปัญหาที่สำคัญกว่าคือ หน่วยประมวลผลที่ใช้ในงานควบคุมมีมากหลายเบอร์ หลายยี่ห้อ และมีชุดคำสั่งไม่เหมือนกัน โปรแกรมที่ทำงานได้บนหน่วยประมวลผลรุ่นหนึ่งจะต้องถูกคอมไพล์ใหม่ จึงจะนำไปใช้งานบนหน่วยประมวลผลอีกรุ่นหนึ่งได้ ด้วยเหตุนี้พวกเขาจึงพัฒนาภาษาใหม่ชื่อโอ๊ก (ได้ชื่อจากต้นโอ๊ก ที่อยู่ข้างตึกที่ทำงาน) ให้เป็นภาษาที่ง่ายต่อการเรียนรู้และใช้งาน ไม่มีข้อผิดพลาดในตอนทำงาน (bug-free) และเหมาะที่จะทำงานในระบบที่มีหน่วยความจำน้อย พวกเขาแน่ใจว่าระบบควบคุมที่มีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆจึงให้โอ๊กเป็นภาษาเชิงวัตถุ เพื่อง่ายในการสร้างและดูแลระบบขนาดใหญ่ ปัญหาใหญ่ของการออกแบบภาษาโอ๊ก อยู่ที่ต้องการให้เป็นภาษาที่ทำงานบนหน่วยประมวลผลใดๆ ก็ได้ จึงนำเทคนิคการคอมไพล์โปรแกรมเป็นคำสั่งของหน่วยประมวลผลสมมติตัวหนึ่ง แล้วสร้างอินเทอร์พรีเตอร์ของหน่วยประมวลผลสมมติตัวนั้นให้แก่หน่วยประมวลผลที่จะทำงานโปรแกรมนั้น ด้วยวิธีนี้ โปรแกรมที่สร้างขึ้นจึงสามารถนำไปทำงานบนเครื่องที่มีหน่วยประมวลผลต่างรุ่นได้ เราเรียกคุณสมบัติอย่างนี้ว่า platform independent

ผลงานของ Green group ให้ความหวังแก่บริษัท ซัน ไมโครซิสเต็มส์ อย่างมาก จึงก่อตั้งบริษัทลูกชื่อว่า FirstPerson Inc. ในปี 1992 เพื่อร่วมมือกับบริษัท Time Warner พัฒนาระบบ video on demand ที่มีอุปกรณ์ควบคุมเครื่องรับโทรทัศน์ที่สามารถติดต่อกับผู้ใช้ในลักษณะของ Interactive TV ภาษาโอ๊กถูกใช้เขียนโปรแกรมของ set-top box ซึ่งเป็นกล่องที่ต่อพ่วงกับเครื่องโทรทัศน์ เพื่อควบคุมและติดต่อกับผู้ใช้ แม้ระบบนี้ถูกพัฒนาจนใช้งานได้ แต่บริษัท Time Warner หยุดโครงการนี้ไป บริษัทซันจึงพยายามหาลูกค้ารายใหม่ให้แก่เทคโนโลยีนี้ โดยนำไปทดสอบใช้งานอื่นๆเช่น ระบบควบคุมบัตรเครดิต ระบบควบคุมเครื่องจักรในโรงงานอุตสาหกรรม และระบบควบคุมในเครื่องซีดี-รอม แต่ก็หาลูกค้าไม่ได้สักราย น่าแปลกที่พวกเขาเป็นแนวหน้าอยู่ในตลาดระบบเครือข่าย แต่กลับไม่ได้นึกถึงอินเทอร์เน็ตอยู่เป็นเวลานานนับปี

ในปี 1993 เมื่อ ไฮเพอร์เท็กซ์และบราวเซอร์ เปลี่ยนแปลงอินเทอร์เน็ตไปอย่างมากและมีผู้ใช้มากขึ้นอย่างรวดเร็ว บริษัท ซัน ไมโครซิสเต็มส์ มองเห็นความจำเป็นที่ต้องมี ภาษาสำหรับสร้างโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ใดๆก็ได้ และเพิ่งนึกถึงคุณสมบัติ platform independent ของภาษาโอ๊ก จึงนำมาปัดฝุ่นปรับปรุงใหม่ และทดลองสร้างเว็บบราวเซอร์ซึ่งเว็บรันเนอร์ ที่ทำงานโปรแกรมภาษาโอ๊กได้ เมื่อทดลองจนได้ผล พวกเขาทราบว่ามีสิ่งที่สำคัญอย่างยิ่งในมือแล้ว แต่โอ๊กเป็นชื่อทางการค้าที่มีผู้ใช้อยู่ก่อน จึงเปลี่ยนเป็นจาวาในตอนต้นปี 1995 พร้อมกับเปลี่ยนชื่อเว็บรันเนอร์เป็น ฮอตจาวา (HotJava)

เมื่อเริ่มต้นจาวาทำงานบนระบบโซลาริสเท่านั้น แต่เพียงภายในฤดูร้อนของ 1995 ก็พัฒนาให้ทำงานได้บน วินโดวส์ เอ็นที, วินโดวส์ 95 และลีนุกซ์ พอถึงปลายปี 1995 บริษัทเน็ตสเคปก็สร้างเน็ตสเคป 2.0ให้ทำงานจาวาได้ หลังจากนั้นบริษัทไมโครซอฟต์กับไอบีเอ็ม ก็ประกาศสนับสนุนภาษาจาวากับเขาด้วย และในตอนปลายปี 1995 บริษัท ซัน ไมโครซิสเต็มส์ นำโปรแกรมชุดพัฒนาภาษาจาวา (JDK) รุ่น 1.0 ขึ้นแจกจ่ายในอินเตอร์เน็ต

6.2 คุณสมบัติของภาษาจาวา

ภาษาจาวาถูกออกแบบให้มีลักษณะดังต่อไปนี้

1.เป็นภาษาที่ง่าย (Simple) ในการเรียนรู้และใช้งาน ความหมายของคำว่า "ง่าย" อาจจะถูกตีความหมายได้หลายอย่างในแง่ต่างๆ ต่อไปนี้

- ภาษาจาวานำไวยากรณ์ภาษาส่วนใหญ่มาจากภาษาซีและซีพลัสพลัส ผู้ที่คุ้นเคยกับภาษาซีหรือซีพลัสพลัส อยู่แล้ว จะเข้าใจไวยากรณ์ภาษาจาวาได้ง่าย หรือใช้เวลาศึกษาไม่นานนัก

- ภาษาจาวามีกลไกของภาษาจำนวนไม่มากและไม่ซับซ้อน โดยตัดกลไกของภาษาซีและซีพลัสพลัสส่วนที่ทำให้ภาษายุ่งยากออกไปอย่างเช่น pointer arithmetic, default argument ,scope resolution ,protected and private inheritance และ operator overloading แต่ในขณะเดียวกันก็เพิ่มความสามารถให้คอมไพเลอร์ของภาษาจาวา ทำให้ไม่มี preprocessor commands ดังนั้นจะไม่มี macros definitions,included files , conditional compilation และ header files เมื่อจาวาเป็นภาษาเชิงวัตถุแล้ว กลไกอย่างเช่น structures ,unions ,bit fields และ enumerated types รวมทั้งการทำ typedef ก็ไม่มีความจำเป็นจึงถูกตัดออกไป ภาษาจาวาถูกออกแบบให้เป็นภาษาเชิงวัตถุอย่างรอบคอบกว่าภาษาซีพลัสพลัส ดังจะเห็นได้ว่ากลไกที่อาจทำให้เกิดความกำกวมอย่างเช่น multiple inheritance และ copy constructor และเป็นกลไกที่อาจจะทำลายแบบแผนการเขียนโปรแกรมเชิงวัตถุที่ดีอย่าง เช่น friend methods ก็ถูกตัดออกไปด้วย

- ภาษาจาวาประสบความสำเร็จอย่างมาก ในการใช้เทคนิคภาษาเชิงวัตถุ ช่วยสร้างโปรแกรมที่ยุ่งยากเป็นไปได้ง่ายขึ้น ดังจะเห็นได้ว่าหากเป็นภาษาอื่น การสร้างโปรแกรมที่เกี่ยวกับ graphic user interfaces, multitasking, network และ distributed objects ผู้ใช้โปรแกรมจะต้องมีความรู้ขั้นสูงจึงจะทำงานได้ หรือไม่ก็ต้องอาศัยโปรแกรมประเภทวิซาร์ดอย่างในวิชวลซีพลัสพลัสช่วยสร้างโปรแกรม แต่ภาษาจาวามีกลไกของ

โปรแกรมเชิงวัตถุที่ส่งเสริมการนำโปรแกรมที่สร้างไว้แล้วนำมาใช้งานใหม่ (reuse) โดยสามารถเปลี่ยนแปลงหรือเพิ่มเติมโปรแกรมบางส่วน ให้เหมาะกับงานที่ต้องการได้โดยไม่จำเป็นต้องทราบรายละเอียดของการสร้างโปรแกรมเดิมทั้งหมด จึงสามารถสร้างโปรแกรมสำหรับงานที่ยุ่งยากขึ้นจากโปรแกรมที่มีผู้สร้างไว้แล้วโดยง่าย

- ภาษาที่ ถูกเรียกว่า typed language จะทำการตรวจสอบเกี่ยวกับ type เพื่อช่วยให้โปรแกรมทำงานโดยไม่มีข้อผิดพลาดเกี่ยวกับ types แต่การเขียนโปรแกรมภาษาเช่นนี้มีข้อยุ่งยาก อย่างเช่นตัวอักษรบวกกับเลขจำนวนเต็มไม่ได้ เนื่องจากค่าที่จะนำมาทำการคำนวณ หรือกำหนดค่า (assignment)ให้แก่กัน จะต้องเป็น type ชนิดเดียวกัน ภาษาจาวาเป็น typed language แต่เพื่อใช้งานง่าย จึงสร้างกฎเกณฑ์เกี่ยวกับ automatic type conversion ซึ่งเป็นการเปลี่ยนแปลงค่าระหว่าง type ที่ต่างกัน ช่วยให้การโปรแกรมง่ายขึ้นโดยลดภาระการเปลี่ยน types ไปจากผู้โปรแกรม

2. โปรแกรมที่สร้างขึ้นด้วยภาษาจาวาจะไม่มีความผิดพลาดจากข้อบกพร่องของภาษา นั่นคือโปรแกรมจะต้องไม่ล้มเหลวลง ด้วยความผิดปรกติเพียงเล็กน้อยที่ไม่เกี่ยวกับตรรกะของโปรแกรม คุณสมบัติของภาษาอย่างนี้เรียกว่า ความคงทน (Robust) ภาษาจาวาถูกออกแบบให้มีความคงทนด้วยวิธีการดังนี้

- ภาษาจาวาเน้นการใช้กลไก exception handling เพื่อให้โปรแกรมสามารถจัดการกับความผิดปกติบางอย่างที่เกิดขึ้นในขณะโปรแกรมทำงานให้โปรแกรมทำงานต่อไปได้โดยไม่ต้องหยุดลง

- ภาษาจาวาตัดกลไกบางอย่างในภาษาซีและซีพลัสพลัส ที่อาจจะทำให้เกิดความผิดพลาด หากใช้ไม่ระวัง เช่น global variables, variable length arguments และ goto statement ภาษาจาวาตัดการอ้างถึงแอดเดรสของตัวแปร และการใช้พอยน์เตอร์สำหรับอ่านหรือเขียนข้อมูลในหน่วยความจำโดยตรง

- ภาษาจาวาไม่มีกลไกสำหรับคืน (de-allocation) หน่วยความจำที่ขอมาในขณะโปรแกรมทำงาน (dynamic memory allocation) อย่างที่มีในภาษาซีและซีพลัสพลัส คือ free() และ delete() ภาษาจาวาอาศัย automatic garbage collector ทำหน้าที่เก็บหน่วยความจำที่ไม่สามารถอ้างถึงได้แล้วกลับไปใช้งานใหม่

- ภาษาจาวาเป็นภาษาประเภท strongly typed หมายถึง ภาษาที่เน้นความถูกต้องของชนิดข้อมูล (type) ที่ใช้ในโปรแกรม คอมไพเลอร์ของภาษาประเภทนี้จะทำการตรวจสอบว่าโปรแกรมการจัดการกับชนิดข้อมูลของตัวแปรถูกต้องหรือไม่ เรียกกิจกรรมของคอมไพเลอร์นี้ว่า type checking ความผิดพลาดเกี่ยวกับชนิดข้อมูลทั้งหมดจะถูกปฏิเสธตั้งแต่ตอนคอมไพล์โปรแกรมจึงไม่มีความผิดพลาดเกี่ยวกับชนิดข้อมูลเกิดขึ้นในระหว่างที่โปรแกรมทำงาน นอกจากนั้นยังมีการตรวจสอบอีกว่า ในระหว่างโปรแกรมทำงานมีการเปลี่ยนชนิดข้อมูล (casting) ระหว่างค่าต่าง type ถูกต้องหรือไม่ และการอ้างถึงสมาชิกในอาร์เรย์หรือสตริงอยู่ในขอบเขตที่ถูกต้องหรือไม่

3. โปรแกรมภาษาจาวามักถูกส่งผ่านระบบเครือข่ายไปทำงานบนเครื่องคอมพิวเตอร์ของผู้อื่น ดังนั้นภาษาจาวาต้องมีหลักประกันให้แก่ผู้รับโปรแกรมนั้นไปทำงานว่า จะไม่ก่อให้เกิดอันตรายต่อเครื่องหรือระบบของเขา ภาษาจาวาจึงมีข้อกำหนดหลายอย่าง เพื่อให้โปรแกรมไม่สามารถทำอันตรายหรือสิ่งที่ไม่ควรสมควร

ทำ ต่อระบบที่รับโปรแกรมนั้นไปทำงาน คุณสมบัติของภาษาอย่างนี้เรียกว่า ปลอดภัย (Secure) อย่างไรก็ตาม ไม่มีภาษาใดที่ปลอดภัยร้อยเปอร์เซ็นต์ ภาษาจาวาถูกจัดว่ามีความปลอดภัยในระดับสูงเท่านั้น เพราะถูกออกแบบมาเพื่อความปลอดภัยมากกว่าภาษาอื่น โดยแบ่งการป้องกันออกเป็นหลายระดับดังนี้

- ดังที่กล่าวมาแล้วว่า ภาษาจาวาไม่ยอมให้อ้างถึงค่าในหน่วยความจำผ่านทางพอยน์เตอร์และจะทำการตรวจสอบว่าการอ้างถึงสมาชิกในอาร์เรย์อยู่ในขอบเขตหรือไม่ โปรแกรมจึงไม่สามารถเขียนหรืออ่านค่าในหน่วยความจำที่ไม่มีสิทธิ์อ้างถึง การเปลี่ยนแปลงโปรแกรมหรือค่าในหน่วยความจำ เพื่อสร้างโปรแกรมที่จะเป็นอันตรายต่อคนอื่นด้วยวิธีนี้จึงเกิดขึ้นไม่ได้

- จาวาอินเทอร์พรีเตอร์ มี byte-code verifier ทำหน้าที่ตรวจสอบโปรแกรมที่จะถูกทำงานว่ามีคำสั่งที่ผิดปกติหรือมีการทำงานที่ไม่สมควร หรือไม่หากพบก็จะถูกปฏิเสธที่จะทำงานโปรแกรมนั้น

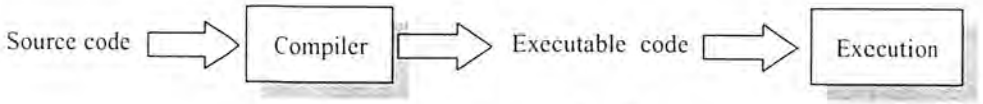
- ภาษาจาวามีระบบรักษาความปลอดภัยที่เรียกว่า sandbox model นั่นคือโปรแกรมที่ถูกนำมาจากเครื่องอื่นผ่านระบบเครือข่ายจะถือว่าเป็นโปรแกรมที่ไม่น่าไว้ใจ (un – trusted codes) และถูกเก็บอยู่ในภาวะที่เรียกว่า sandbox โปรแกรมที่อยู่ใน sandbox จะมีข้อจำกัดในการทำงานหลายอย่าง ซึ่งถูกควบคุมโดย security manager เช่นไม่สามารถอ่านหรือเขียนไฟล์ได้เป็นต้น

4. จุดมุ่งหมายสำคัญของการออกแบบภาษาจาวา คือโปรแกรมต้องสามารถทำงานบนเครื่องต่างระบบกันได้ เรียกคุณสมบัติภาษาอย่างนี้ว่า "ไม่ขึ้นกับระบบ" (architecture neutral หรือ platform independent) เพื่อให้ได้คุณสมบัตินี้ ภาษาจาวาต้องมีการแปลภาษาแบบทั้ง Compilation และ interpretation รวมทั้งการกำหนดเครื่องจักรสมมติของจาวา (JavaVirtual Machine) ดังจะกล่าวต่อไปนี้

6.3 คอมไพเลชัน และอินเทอร์พรีเทชัน (Compilation and interpretation)

ภาษาระดับสูง (high level language) อย่างเช่น ฟอร์แทรน, ปาสคาล, ซี, ซีพลัสพลัสหรือจาวาเป็นต้น มีความใกล้เคียงกับภาษาคณิตศาสตร์ เพื่อสะดวกในการคำนวณหรือแก้ปัญหาได้ง่ายกว่าการใช้คำสั่งที่หน่วยประมวลผลทำงานได้ (machine code) ภาษาระดับสูงช่วยให้การสร้างโปรแกรมมีประสิทธิภาพมากขึ้น เพราะผู้โปรแกรมสามารถคิดในระดับของการแก้ปัญหา แทนที่จะเป็นขั้นตอนการทำงานของหน่วยประมวลผลสำหรับแก้ปัญหานั้น ซึ่งหมายถึงผู้โปรแกรมไม่ต้องศึกษาทำความเข้าใจโครงสร้างและชุดคำสั่งของหน่วยประมวลผลนั้นก็สามารถสร้างโปรแกรมได้ อย่างไรก็ตาม หน่วยประมวลผลส่วนใหญ่ไม่สามารถทำงานโปรแกรมภาษาระดับสูงได้โดยตรงจึงต้องมีตัวแปลภาษา (Language translator) เปลี่ยนโปรแกรมต้นฉบับ (source code) ของภาษาระดับสูงให้เป็นโปรแกรมของคำสั่งที่หน่วยประมวลผลทำงานได้ เรียกว่า เอ็กซิคิวต์เอเบิลโค้ด โดยทั่วไปวิธีแปลภาษามีสองแบบคือ

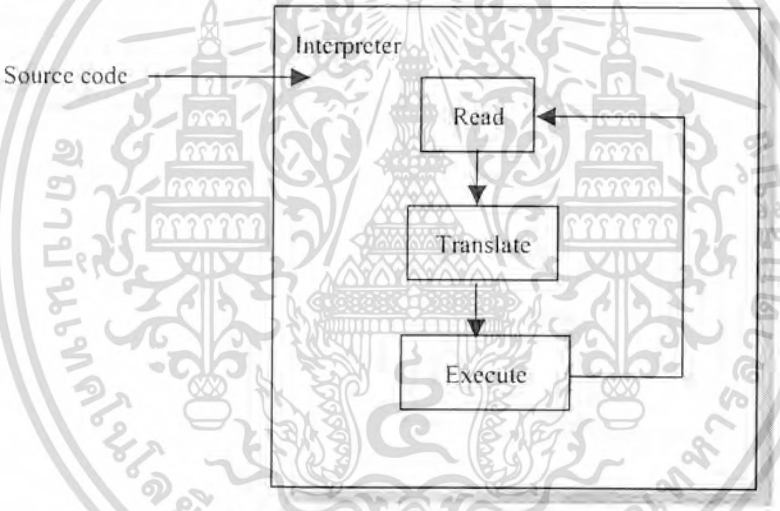
- คอมไพเลชัน ตัวแปลภาษาในกรณีนี้เรียกว่าคอมไพเลอร์ ทำหน้าที่วิเคราะห์โปรแกรมต้นฉบับแล้วสร้างเอ็กซิคิวต์เอเบิลโค้ดเป็นผลลัพธ์การกระทำดังกล่าวเรียกว่า การคอมไพล์โปรแกรม สรุปได้ดังนี้



รูปที่ 6-1 แสดงการการคอมไพล์โปรแกรม

ข้อดีของวิธีนี้คือ เอ็กซิคิวต์เอเบิลโค้ดทำงานได้เร็วมาก เนื่องจากขั้นตอนในการแปลภาษาถูกแยกออกไปทำก่อนโปรแกรมทำงาน และคอมไพเลอร์ สามารถวิเคราะห์โปรแกรมได้ทั้งโปรแกรม จึงสามารถทำการ optimization ให้โปรแกรมมีประสิทธิภาพ ในการทำงานและมีขนาดเล็ก ตัวอย่างของภาษาที่ใช้การแปลภาษาแบบคอมไพเลชันคือ ฟอร์แทรน, อัลกอล, ปาสคาล, ซีและ ซีพลัสพลัส

- อินเทอร์พรีเทชัน ตัวแปลภาษาในกรณีนี้เรียกว่า อินเทอร์พรีเตอร์ ทำหน้าที่อ่านโปรแกรมต้นฉบับทีละบรรทัด แล้วแปลโปรแกรมบรรทัดนั้นเป็นเอ็กซิคิวต์เอเบิลโค้ดและทำงาน จากนั้นก็อ่านโปรแกรมต้นฉบับบรรทัดต่อไปมาทำเช่นเดิม ไปจนกว่าจะจบโปรแกรม การทำเช่นนี้เรียกว่าอินเทอร์เพต สรุปได้ดังรูปนี้

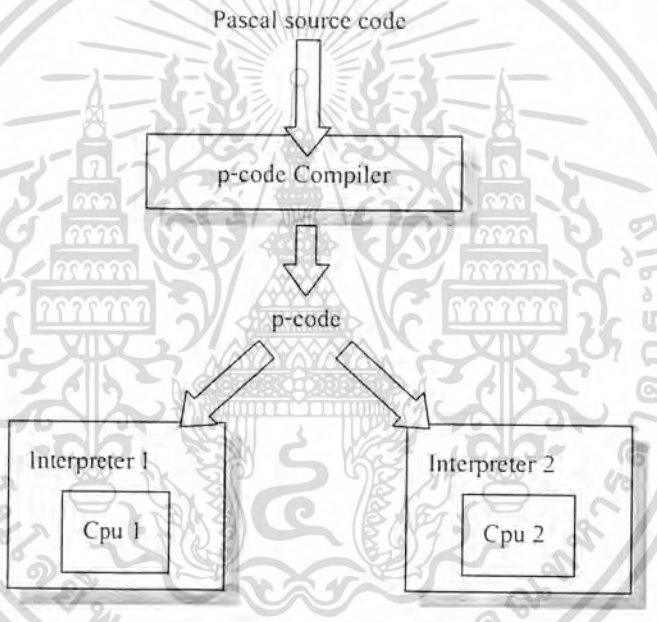


รูปที่ 6-2 แสดงการทำงานของอินเทอร์พรีเตอร์

วิธีนี้จะมีทั้งการแปลภาษาและทำงานโปรแกรมเกิดขึ้นสลับกันไป โปรแกรมจึงทำงานช้ากว่าการคอมไพเลชัน แต่มีข้อดีคือ อินเทอร์พรีเตอร์ถูกสร้างขึ้นได้ง่ายและมีขนาดเล็กกว่า เพราะการวิเคราะห์และแปลโปรแกรมทีละบรรทัดทำได้ง่ายกว่า ทำทั้งโปรแกรมและการแปลภาษาระหว่างโปรแกรมทำงานจะทำให้ได้ภาษาที่มีความยืดหยุ่นในการเขียนโปรแกรมกว่า ตัวอย่างภาษาที่ใช้การแปลภาษาแบบอินเทอร์พรีเทชัน คือ ลิปซ์, เบสิก, โปรล็อกและสมอลล์ทอร์ก

วิธีแปลภาษาทั้งสองแบบมีทั้งข้อดีและข้อเสีย โดยปกติภาษาหนึ่งจะเลือกใช้การแปลภาษาเพียงแบบใดแบบหนึ่ง แต่ก็มีภาษาที่ใช้การแปลภาษาทั้งสองแบบเพื่อประโยชน์บางประการตัวอย่างคือ

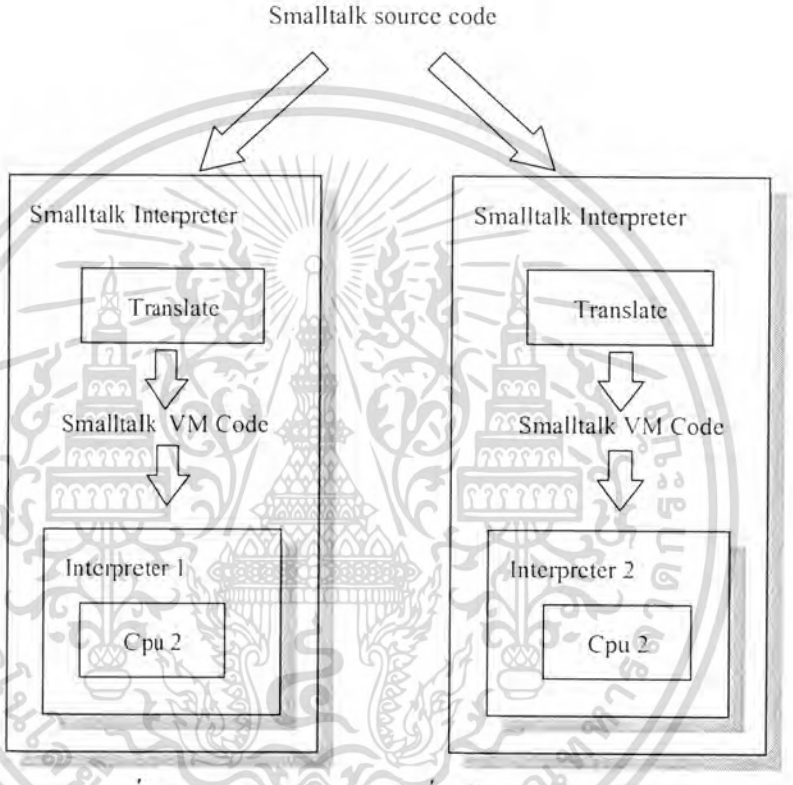
ในตอนต้น 1970s นักวิจัยที่ UC San Diego สร้างคอมไพเลอร์ที่แปลโปรแกรมภาษาปาสคาล เป็นโปรแกรมของภาษาสมมติหนึ่ง เรียกว่า p-code จากนั้นโปรแกรม p-code จะถูกทำงานโดย อินเทอร์พรีเตอร์สำหรับหน่วยประมวลผลแต่ละรุ่น เหตุที่ต้องทำเช่นนี้ ก็เพราะในสมัยนั้นการสร้างคอมไพเลอร์ที่แปลภาษาปาสคาลไปเป็นภาษาเครื่อง ถือเป็นเรื่องที่ยากมาก อีกทั้งในเวลานั้นยังไม่มีบริษัทคอมพิวเตอร์ใดครองส่วนแบ่งตลาดจำนวนมากได้อย่างปัจจุบัน จึงมีหน่วยประมวลผลหลายยี่ห้อใช้อยู่เป็นจำนวนมาก หากจะสร้างคอมไพเลอร์ภาษาปาสคาลให้แก่หน่วยประมวลผลรุ่นใดรุ่นหนึ่งก็อาจไม่คุ้ม เพราะไม่ได้ใช้อย่างแพร่หลายเขาจึงใช้วิธีคอมไพล์จากภาษาปาสคาล ไปเป็นภาษาสมมติ p-code ซึ่งเป็นภาษาที่ใกล้เคียงกับคำสั่งของหน่วยประมวลผลทั่วไป แล้วสร้างอินเทอร์พรีเตอร์ของ p-code สำหรับหน่วยประมวลผลแต่ละรุ่น ซึ่งไม่ใช่งานที่ยากนัก สรุปได้ดังรูปนี้



รูปที่ 6-3 การทำงานของโปรแกรม p-code

วิธีนี้ช่วยให้คอมพิวเตอร์รุ่นต่างๆ สามารถทำงานกับภาษาปาสคาลได้ โดยมีเพียง p-code คอมไพล์เลอร์ที่เป็นมาตรฐานกับ p-code อินเทอร์พรีเตอร์ สำหรับหน่วยประมวลผลที่ใช้ อย่างไรก็ตามวิธีนี้ไม่เป็นที่แพร่หลายเพราะโปรแกรมทำงานช้า และเมื่อเข้าสู่ยุคที่หน่วยประมวลผลของบริษัทอินเทลและโมโตโรล่า ได้รับความนิยมอย่างมาก ความหลากหลายของหน่วยประมวลผลจึงลดลง และการสร้างคอมไพเลอร์ไปสู่คำสั่งของหน่วยประมวลผลรุ่นที่มีคนนิยมใช้ จึงประสบความสำเร็จมากกว่า

ในปลายยุค 1970s ภาษาสมอลล์ทอร์ก พัฒนาความคิดนี้ไปอีกแนวหนึ่งคือ แทนที่จะกำหนดเพียงภาษาสมมติเท่านั้น แต่กำหนดไปถึงโครงสร้างและพฤติกรรมของเครื่องจักรสมมติ (virtual machine) สำหรับทำงานโปแกรมภาษาสมมตินั้นด้วย ภาษาสมอลล์ทอร์กใช้การแปลภาษาแบบอินเทอร์พรีเทชันโดยสมอลล์ทอร์กอินเทอร์พรีเตอร์ ทำหน้าที่แปลโปรแกรมทีละบรรทัดไปเป็นโปรแกรมภาษาสมมติ แล้วทำงานโดยใช้เครื่องจักรสมมติที่จำลองขึ้น สรุปได้ดังรูปนี้

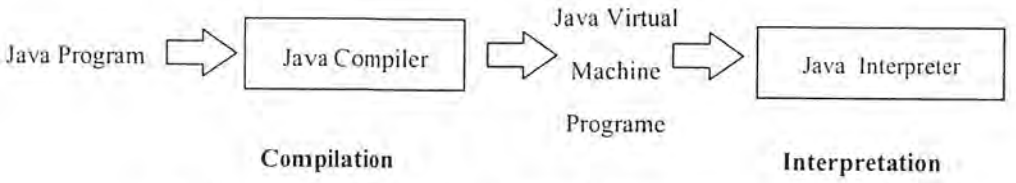


รูปที่ 6-4 แสดงการทำงานของเครื่องจักรสมมติ

วิธีการนี้ทำให้โปรแกรมภาษาสมอลล์ทอร์ก ทำงานบนเครื่องคอมพิวเตอร์ใดๆที่มี สมอลล์ทอร์กอินเทอร์พรีเตอร์ ให้ผลลัพธ์เหมือนกันหมด โดยไม่ต้องเปลี่ยนแปลงโปรแกรมส่วนใดเลย แม้ภาษาสมอลล์ทอร์กจะประสบความสำเร็จในเรื่องนี้ แต่ยังติดปัญหาที่โปรแกรมทำงานช้ามากจึงไม่ได้รับความนิยมเท่าที่ควร

6.4 เครื่องจักรสมมติของจาวา (Java Virtual Machine)

ภาษาจาวานำความคิดการสร้างเครื่องจักรสมมติมาใช้ เพื่อให้โปรแกรมทำงานไม่ขึ้นกับระบบ โดยมีคอมไพเลอร์ทำการแปลภาษาให้เป็นโปรแกรมของ (JVM) แล้วนำโปรแกรมนั้นมาทำงานด้วยเครื่องจักรสมมติที่จำลองขึ้นโดยจาวาอินเทอร์พรีเตอร์ สรุปได้ดังรูปนี้



รูปที่ 6-5 การทำงานด้วยเครื่องจักรสมมติที่จำลองโดยจาวาอินเทอร์พรีเตอร์

ในวิธีนี้โปรแกรมภาษาจาวาจะถูกคอมไพล์โดยจาวาคอมไพล์เลอร์ได้เป็นโปรแกรมของ JVM แล้วสามารถนำไปทำงานบนเครื่องใดๆ ที่มีจาวาอินเทอร์พรีเตอร์ ได้จึงมีคุณสมบัติไม่ขึ้นกับระบบ โปรแกรมของ JVM จะทำงานได้เร็วกว่าการใช้อินเทอร์พรีเตอร์เพียงอย่างเดียวแบบสมอลล์ทอร์ก เพราะขั้นตอนการทำคอมไพเลชันถูกแยกออกไปจากการเอ็กซิคิวต์ชันและด้วยการออกแบบคำสั่งของ JVM ให้ใกล้เคียงกับคำสั่งของหน่วยประมวลผลทั่วไปจาวาอินเทอร์พรีเตอร์ จึงเปลี่ยนคำสั่งของ JVM ไปสู่คำสั่งของหน่วยประมวลผลที่ใช้งานได้ง่าย การทำอินเทอร์พรีเทชันโปรแกรมของ JVM จึงเร็วกว่าการอินเทอร์พรีเทชันของภาษาระดับสูงอื่นๆ

บริษัท จาวาซอฟต์ (บริษัทลูกของ ซัน ไมโครซิสเต็มส์) เป็นผู้กำหนดชุดคำสั่งของ JVM รวมทั้งความหมาย (เรียกว่า Semantics หรือผลของการทำงาน) ของแต่ละคำสั่ง ข้อกำหนดเหล่านี้ถือเป็นมาตรฐานของภาษาจาวาที่เผยแพร่ให้แก่บุคคลทั่วไป ใครก็ตามที่สามารถสร้าง JVM ของตนเองขึ้นได้ ไม่ว่าจะใช้วิธีทางฮาร์ดแวร์หรือซอฟต์แวร์ ภายใน JVM จะมีหน่วยประมวลผลสมมติที่เรียกว่า วิชวลโพรเซสเซอร์ (virtual processor) ทำหน้าที่ประมวลผลคำสั่งของ JVM ปัจจุบัน JVM ยังอยู่ในระหว่างการพัฒนา ดังนั้น JVM เกือบทั้งหมดที่ใช้กันอยู่ในตอนนี้ จึงเป็นโปรแกรมที่จำลองการทำงานของ JVM บนเครื่องคอมพิวเตอร์ทั่วไป โดยปกติวิชวลโพรเซสเซอร์ของ JVM ที่จำลองขึ้นบนเครื่องคอมพิวเตอร์เครื่องหนึ่ง จะแปลคำสั่งของ JVM เป็นคำสั่งของหน่วยประมวลผลในคอมพิวเตอร์เครื่องนั้น เรียกว่า เนทีฟโค้ดแล้วให้หน่วยประมวลผลทำงานคำสั่งนั้น คำสั่ง (opcode) ของ JVM มีขนาด 1 ไบต์ทุกคำสั่ง จึงเรียกโปรแกรม JVM ว่าโปรแกรมไบต์โค้ด จำนวนคำสั่งของ JVM มีได้สูงสุดเพียง 256 คำสั่ง เปรียบกับหน่วยประมวลผลทั่วไปแล้ว ดูคล้ายกับว่าจำนวนคำสั่งของ JVM มากมายเป็นอย่างยิ่งที่จริงแล้วคำสั่งของ JVM แบ่งออกเป็นเพียงไม่กี่ประเภท แต่ละประเภททำหน้าที่คล้ายๆ กัน เพียงแต่ทำกับ operands ต่างชนิดข้อมูลกันเท่านั้น

ชุดคำสั่งของ JVM ถูกออกแบบมาเพื่อสนับสนุนการทำงานของโปรแกรมเชิงวัตถุจึงมีคำสั่งเกี่ยวกับการสร้างอินสแทนซ์ (instances) และการอ้างถึงสมาชิกในอินสแทนซ์ ซึ่งไม่มีในหน่วยประมวลผลทั่วไป ภาษาจาวาเป็นภาษาที่เน้นความถูกต้องเกี่ยวกับชนิดของข้อมูล (type) จึงมีคำสั่งสำหรับคำนวณชนิดข้อมูลพื้นฐานแต่ละชนิดเช่นคำสั่ง Iadd สำหรับบวกเลขจำนวนเต็มชนิด integer และคำสั่ง dadd สำหรับบวกเลขทศนิยมชนิด double เป็นต้น บางคำสั่งของ JVM จะเหมือนกับคำสั่งที่มีในหน่วยประมวลผลทั่วไป

JVM ถูกออกแบบให้สามารถจำลองได้บนประมวลผลทั่วไป แต่หน่วยประมวลผลทั่วไปมีจำนวนรีจีสเตอร์ไม่เท่ากัน บางรุ่นมีรีจีสเตอร์ หน้าที่พิเศษที่รุ่นอื่นไม่มี ผู้ออกแบบจึงตัดปัญหานี้โดยให้ JVM ไม่มี

รีจีสเตอร์และทำการคำนวณทั้งหมดบนสแต็กชุดคำสั่งของ JVM จึงเป็น stacked operations กล่าวได้ว่า JVM เป็น Stack machine

วิชวลโพรเซสเซอร์ใน JVM จะเปลี่ยนคำสั่งไบต์โค้ดไปเป็นเนทีฟโค้ดที่ทำหน้าที่เดียวกันแล้วทำงาน เนทีฟโค้ดนั้นสังเกตว่าเนทีฟโค้ดนั้นอาจเป็น Application Program Interfaces (API) ของระบบปฏิบัติการที่ใช้ หรืออาจจะเป็นไลบรารีมาตราฐานที่สร้างขึ้นสำหรับหน่วยประมวลผลนั้น ทำให้ JVM หนึ่งอาจใช้งานได้ในระบบต่างๆ โดยเปลี่ยนแปลงแต่ไลบรารีมาตราฐานเท่านั้น

โปรแกรมที่ทำงานโดยการอินเทอร์พรีเทชันย่อมช้ากว่าโปรแกรมที่ทำงานโดยตรง ปัจจุบันโปรแกรมภาษาจาวาทำงานช้ากว่าภาษาซีโดยเฉลี่ยประมาณ 10 เท่า แต่ก็เร็วกว่าภาษาเบสิก , สมอลล์ทอร์ก หรือ จาวาสคริปส์มาก มีการค้นคว้าเพื่อเพิ่มความเร็วของภาษาจาวาเช่นสร้างหน่วยประมวลผลที่สามารถทำงานคำสั่งของ JVM ได้โดยตรง และพัฒนาจาวาอินเทอร์พรีเตอร์ ซึ่งได้ไม่ทำงานคำสั่งของ JVM ทีละคำสั่ง แต่จะเปลี่ยนโปรแกรมของ JVM ทั้งโปรแกรมเป็นเนทีฟโค้ดแล้วทำงานเนทีฟโค้ดนั้น วิธีการนี้เรียกว่า Just in time เพราะคล้ายกับว่าทำการคอมไพล์โปรแกรม JVM ก่อนจะเริ่มทำงาน โปรแกรมจะทำงานเร็วขึ้นมากเพราะคำสั่งที่ถูกทำซ้ำบ่อยๆ เช่นคำสั่งที่อยู่ในประโยคทำซ้ำหรือฟังก์ชันที่ถูกเรียกใช้บ่อยๆ จะถูกแปลเพียงครั้งเดียว ไม่ใช่ทุกครั้งที่จะถูกทำงาน

บทที่ 7
การออกแบบและการสร้าง

การออกแบบโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัยจะทำได้โดยการนำเอาโพรโตคอลที่มีการเข้ารหัสเข้ามาช่วย ซึ่งได้ใช้โพรโตคอลเซ็กเคียวเชลล์มาช่วยในการเข้ารหัสในส่วนของข้อมูลที่เกี่ยวข้องในการติดต่อสื่อสารกับเซิร์ฟเวอร์ซึ่งก็รวมไปถึงการเข้ารหัสชื่อผู้ใช้, รหัสผ่านและข้อมูลต่างๆ นอกจากนี้ยังช่วยเพิ่มความปลอดภัยในการพิสูจน์ตนเพื่อป้องกันการโจมตีประเภทอื่นๆ อีกด้วย

ในการพัฒนานั้นจะต้องทำความเข้าใจทั้งในส่วนของโพรโตคอลเอฟทีพี และโพรโตคอลเซ็กเคียวเชลล์อย่างละเอียด ทำให้สามารถสร้างโปรแกรมที่นำเอาโพรโตคอลทั้งสองอย่างมาประยุกต์ใช้ร่วมกันได้อย่างถูกต้อง พร้อมทั้งศึกษาภาษาที่จะนำมาใช้ในการพัฒนาซึ่งก็คือภาษาจาวาเพื่อให้สร้างโปรแกรมที่สมบูรณ์และสามารถแก้ไข พัฒนาเพิ่มเติมได้ง่าย

7.1 ภาพรวมของการออกแบบ

7.1.1 โพรโตคอลเอฟทีพี

โพรโตคอลเอฟทีพีทำงานอยู่บนโพรโตคอลพื้นฐานคือทีซีพี/ไอพี การสื่อสารกันระหว่างเซิร์ฟเวอร์กับไคลเอ็นต์นั้น จะใช้ 2 ช่องทางการติดต่อเสมอ ช่องทางแรกจะใช้สำหรับส่งคำสั่งจากไคลเอ็นต์ไปเซิร์ฟเวอร์และการติดต่อกลับจากเซิร์ฟเวอร์มายังไคลเอ็นต์ (การเชื่อมต่อส่วนควบคุม : Control Connection) ช่องทางที่สองไว้สำหรับโอนย้ายแฟ้มข้อมูล (การเชื่อมต่อส่วนข้อมูล : Data Connection) ซึ่งช่องทางแรกจะมีความอันตรายของการถูกดักจับมาก เพราะเป็นช่องทางที่ใช้สำหรับส่งรหัสผู้ใช้กับและรหัสผ่าน

โพรโตคอลเอฟทีพีใช้ช่องทางการติดต่อช่องทางที่สองสำหรับการรับส่งแฟ้มข้อมูล โดยทั่วไป การสร้างช่องทางการติดต่อสำหรับการรับส่งแฟ้มข้อมูล ตัวของเซิร์ฟเวอร์เองจะเป็นตัวเริ่มสร้างติดต่อไปที่ไคลเอ็นต์เอง ลักษณะนี้เรียกว่าเซิร์ฟเวอร์ทำงานแบบแอ็กทีฟ อย่างไรก็ตามมักจะเป็นปัญหากับเซิร์ฟเวอร์ที่มีการติดตั้งไฟร์วอลล์ ซึ่งมักจะไม่ยอมให้ทำงานแบบแอ็กทีฟเพื่อเหตุผลทางด้านความปลอดภัยบางประการ

เหตุผลที่ไฟร์วอลล์ไม่ยอมให้เซิร์ฟเวอร์ทำงานแบบแอ็กทีฟ

การส่งแฟ้มข้อมูลในการทำงานแบบแอ็กทีฟ ในส่วนของไคลเอ็นต์จะส่งคำสั่ง PORT พร้อมกับพารามิเตอร์ที่บอกถึงแอดเดรสและพอร์ตของไคลเอ็นต์ที่รอการติดต่อจากเซิร์ฟเวอร์ หลังจากนั้นฝั่งของเซิร์ฟเวอร์ก็จะเริ่มการติดต่อไปยังแอดเดรสและพอร์ตที่ได้รับมา แต่ผู้บุกรุกใช้จุดอ่อนตรงนี้โดยพารามิเตอร์ที่ส่งไปอาจเป็นแอดเดรสและพอร์ตของเครื่องอื่นที่ต้องการโจมตี เรียกการโจมตีแบบนี้ว่า Bounce FTP Attack ซึ่งสามารถนำไปใช้ประโยชน์ในการทำสิ่งอื่น ตัวอย่างเช่น

1. สแกนพอร์ต (port scanning) ผู้บุกรุกจะใช้เซิร์ฟเวอร์เป็นตัวที่จะสแกนพอร์ตกับเครื่องที่ต้องการได้ โดยสามารถซ่อนที่อยู่จริงของคนที่ทำได้

2. ความสามารถในการทะลุทะลวงผ่านไฟล์วอลล์ได้ เช่น สมมติว่าที่ไซต์แห่งหนึ่งได้เปิดบริการเอฟทีพีแบบไม่ออกนาม (anonymous FTP) ไว้ ที่อยู่ข้างหลังไฟล์วอลล์ ผู้บุกรุกก็ทำการเริ่มการเชื่อมต่อไปยังเอฟทีพีเซิร์ฟเวอร์ และส่งคำสั่ง PORT ตามด้วยพารามิเตอร์ที่บอกถึงที่อยู่และพอร์ตที่จะติดต่อ สมมติเป็น 8080 ซึ่งถูกบล็อกจากไฟร์วอลล์ ซึ่งทำให้ผู้บุกรุกสามารถติดต่อกับพอร์ตที่ต้องการได้

การแก้ปัญหาจากการที่ไม่สามารถทำงานใน แอ็กทีฟกับเอฟทีพีเซิร์ฟเวอร์ที่มีไฟร์วอลล์

การแก้ปัญหานี้ทำได้โดยไคลเอ็นต์ เพียงแต่ส่งคำสั่ง PASV เพื่อเปลี่ยนให้เซิร์ฟเวอร์ทำงานแบบ พาสซีฟแทน แล้วให้ ไคลเอ็นต์ เป็นตัวติดต่อไปยังเซิร์ฟเวอร์เพื่อสร้างช่องทางสำหรับส่งแฟ้มข้อมูลแทน

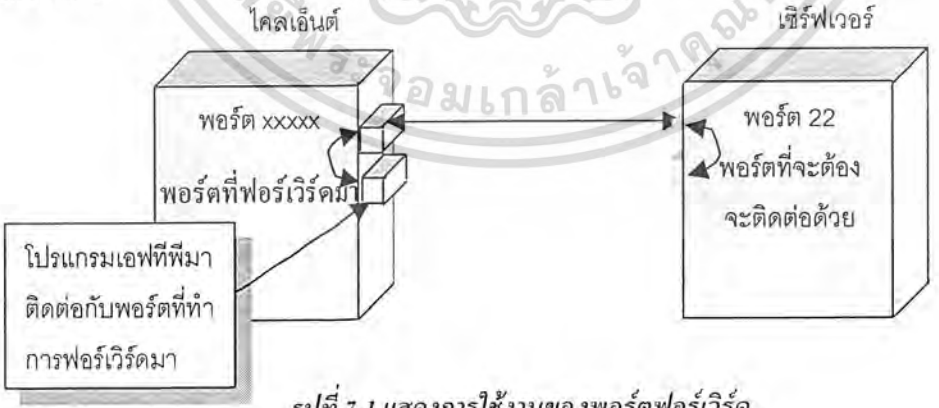
ดังนั้นจึงเป็นข้อสรุปที่ทำการพัฒนาโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย ที่สร้างขึ้นมานั้นจะออกแบบมาให้ใช้กับพาสซีฟเป็นหลัก แต่ก็สามารถทำงานแบบแอ็กทีฟได้ในโหมดที่ไม่ต้องการความปลอดภัยได้ด้วย

7.1.2 โพรโตคอลเซ็กเคียวเซลล์

โพรโตคอลเซ็กเคียวเซลล์ที่นำมาใช้กับเอฟทีพี นี้จะใช้การเข้ารหัสแบบ 3DES เป็นหลัก และได้ออกแบบส่วนของการเรียกใช้งานให้เป็นโมดูลเพื่อที่สามารถเรียกใช้การเข้ารหัสแบบอื่น ๆ ได้ง่าย

คุณสมบัติของโพรโตคอลเซ็กเคียวเซลล์ ที่สำคัญมากที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมเอฟทีพี ที่พัฒนาขึ้นมาก็คือ การฟอร์เวิร์ดพอร์ต (Port Forwarding)

การทำงานของการฟอร์เวิร์ดพอร์ตของเซ็กเคียวเซลล์นั้น เป็นการส่งการเชื่อมต่อจากเครื่องๆหนึ่งไปยังพอร์ตของอีกเครื่องหนึ่ง มีหลักการง่ายๆ คือระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ จะมีการสร้างการเชื่อมต่อที่มีความปลอดภัยไว้ การส่งข้อมูลผ่านระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะมีการเข้ารหัสตลอดโปรแกรมเอฟทีพีเพียงแค่ติดต่อไปยังพอร์ตที่ทำการฟอร์เวิร์ดมาของไคลเอ็นต์ เท่านั้น การส่งข้อมูลก็จะถูกส่งต่อจากพอร์ตดังกล่าวผ่านการช่องทางการติดต่อสื่อสารที่มีการสร้างความปลอดภัยไว้แล้วโดยเซ็กเคียวเซลล์ไปยังเซิร์ฟเวอร์โดยอัตโนมัติ



รูปที่ 7-1 แสดงการใช้งานของพอร์ตฟอร์เวิร์ด

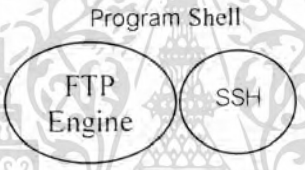
7.2 การออกแบบโปรแกรม

โปรแกรมที่ได้พัฒนามานี้มีชื่อว่า IsagFTP มีลักษณะในมุมมองของการใช้งานของผู้ใช้จะแบ่งออกเป็น 3 ส่วน เกิดจากประยุกต์เอาการใช้งานของโพรโตคอลเซ็กเคียวเซลล์และโพรโตคอลเอฟทีพีเข้าด้วยกัน

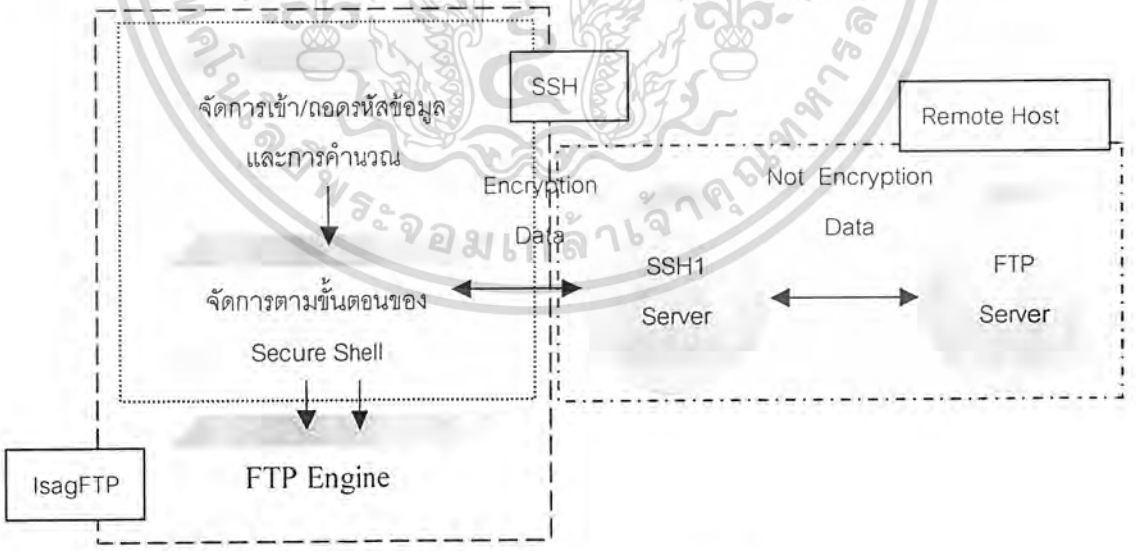
FTP Engine เป็นส่วนที่ประยุกต์ใช้โพรโตคอลเอฟทีพี มีหน้าที่จัดการเกี่ยวกับการโอนย้ายข้อมูล คำสั่งต่าง ๆและการดูแลส่วนต่าง ๆ ที่เกี่ยวข้อง เช่น การเชื่อมต่อไฟล์ที่ยังโอนย้ายไม่ครบ การรับส่งข้อมูลแบบรีเคอร์ซีฟการใช้งานร่วมกับไวด์การ์ดซึ่งเป็นฟังชันพื้นฐานที่ได้เพิ่มเติมเข้าไปในส่วนนี้

SSH เป็นส่วนที่ทำหน้าที่ในการติดต่อกับเซ็กเคียวเซลล์ของเซิร์ฟเวอร์ ทำหน้าที่ได้เหมือนกับเซ็กเคียวเซลล์ไคลเอ็นต์ทั่วไป ทั่ว ๆไป แต่ยังไม่ค่อยสมบูรณ์มากนัก จะเน้นที่การทำพอร์ตฟอร์เวิร์ด ซึ่งใช้ในการสร้างช่องทางการติดต่อสื่อสารที่ปลอดภัยให้กับโปรแกรมเอฟทีพีเสียมากกว่า

Program Shell เป็นส่วนที่สามารถสร้างขึ้นมาครอบ ตัว FTP Engine และ SSH ได้เพื่อให้มีการทำงานเป็นไปตามที่เราต้องการ เช่น จะทำการอินเทอร์เฟซกับผู้ใช้แบบกราฟิก



รูปที่ 7-2 ภาพรวมของโปรแกรมมองจากผู้ใช้



รูปที่ 7-3 รูปแบบของโปรแกรมมองจากการใช้งาน

7.3 The Socket API

หัวใจสำคัญของการติดต่อสื่อสารกับเครื่องผู้ให้บริการในระบบทีซีพี/ไอพี จำเป็นต้องทราบถึงอุปกรณ์ที่ใช้ในการติดต่อ ซึ่งอุปกรณ์นี้เรียกรวม ๆ ว่า Application Program Interface (API) ซึ่งกลุ่มของ API ที่ใช้ในการติดต่อผ่านเครือข่ายที่สำคัญคือ ซ็อกเก็ต (Socket) โดย ซ็อกเก็ตในแอพพลิเคชันนี้แบ่งออกเป็น 2 ลักษณะคือ เซิร์ฟเวอร์ซ็อกเก็ตและไคลเอ็นต์ซ็อกเก็ต

เซิร์ฟเวอร์ซ็อกเก็ต (Server Socket) ทำหน้าที่รอการติดต่อจากเครื่องของผู้ขอใช้บริการ ตัวของเซิร์ฟเวอร์ซ็อกเก็ตนี้จะฝังอยู่ทางฝั่งของเซิร์ฟเวอร์ เมื่อมีผู้ขอใช้บริการร้องขอก็จะทำการค้นหาพอร์ตว่าง และจับจองไว้สำหรับเป็นช่องทางการติดต่อ ตามขั้นตอนคร่าว ๆ 4 ขั้นตอนดังนี้

ขั้นตอนที่ 1 ฝั่งของผู้ใช้บริการทำการเปิดซ็อกเก็ต โดยระบุถึงชื่อของโฮสต์และพอร์ตที่ให้บริการ ในขั้นตอนนี้ขอยกตัวอย่างของโค้ดภาษาจาวา

```
Socket ClientSocket;
ClientSocket = new Socket(Host,Port);
```

จากตัวอย่างของโค้ดข้างบนจะทำให้เกิดการทำงานดังนี้

รูปที่ 7-4 ผู้ใช้บริการทำการเปิดซ็อกเก็ต

ผู้ให้บริการจะเปิดซ็อกเก็ตของตัวเองรออยู่แล้ว เมื่อผู้ใช้บริการเปิดซ็อกเก็ตจะทำให้เซิร์ฟเวอร์ค้นหาพอร์ตว่าง ๆ สำหรับสร้างช่องทางการติดต่อ

ขั้นตอนที่ 2 ผู้ใช้บริการรอ (Accept) จนกว่าเซิร์ฟเวอร์หาพอร์ตว่างได้ และจัดการกับตัวของเซิร์ฟเวอร์เองให้พร้อมสำหรับให้บริการ โค้ดข้างล่างจะแสดงการรอของฝั่งผู้ใช้บริการ

```
Try
{
  if((ClientSocket = new Socket(Host,Port)) == null)  ;request contact server and Accept
}
catch(UnknownHostException e)
```

```
{
 Display("Host "+Host+" not found\n");
}
catch(IOException e)
{
 Display(" Failed to connect to host "+Host+"\n");
}
```

จากตัวอย่างของโค้ดในขั้นตอนที่ 2 ทำให้เกิดการทำงานดังนี้

รูปที่ 7-5 ผู้ให้บริการทำการค้นหาพอร์ต และทำการเปิดพอร์ต

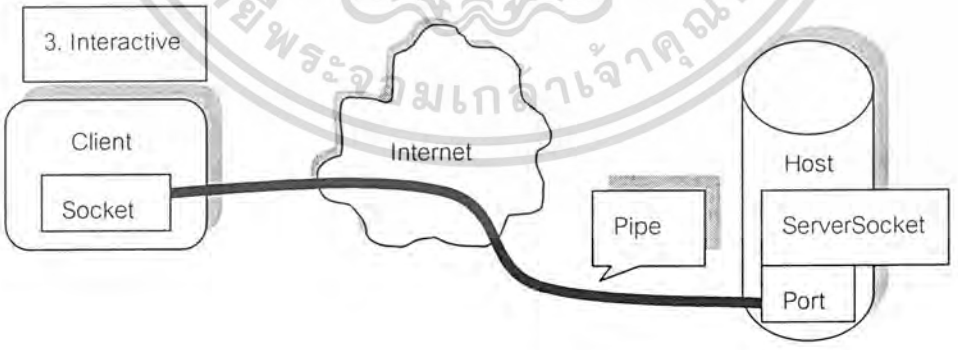
จากโค้ดข้างบนเมื่อมีการผิดพลาดประการใดจากการขอเปิดพอร์ตจะมีการดักจับความผิดพลาดด้วยเพื่อไม่ต้องการให้ระบบล้มเหลว เมื่อผู้ให้บริการพร้อมก็ส่งสัญญาณตอบรับกลับไปให้ยังผู้ใช้บริการได้ทราบเพื่อดำเนินขั้นตอนต่อไป

ขั้นตอนที่ 3 เมื่อทั้ง 2 ฝ่ายพร้อมก็มีการสร้างเส้นทางจำลองหรือท่อของการส่งถ่ายข้อมูล (Pipe line) เส้นทางนี้ข้อมูลจะถูกส่งเป็นแบบอนุกรม ประโยชน์หลัก ๆ คือต้องการให้ทั้ง 2 ฝ่ายง่ายต่อการทำให้ข้อมูลมีความสอดคล้องกันและสัมพันธ์กัน โค้ดข้างล่างแสดงการสร้างเส้นทางการส่งถ่ายข้อมูล

```
BufferedInputStream  is;
BufferedOutputStream os;
try
```

```
{
    is = new BufferedInputStream(ClientSocket.getInputStream(),8192);  ;prepare create
    pipe and input buffer
}
 catch(IOException e){
Display(" Failed to get stream from socket");
System.exit(5);
}
     try
{
    os = new BufferedOutputStream(ClientSocket.getOutputStream());  ;prepare create Pipe
    and output buffer
}
 catch(IOException e)
{
    Display(" Failed to put stream on socket");
    System.exit(5);
}
```

จากตัวอย่างของโค้ดในขั้นตอนที่ 3 ทำให้เกิดการทำงานดังนี้

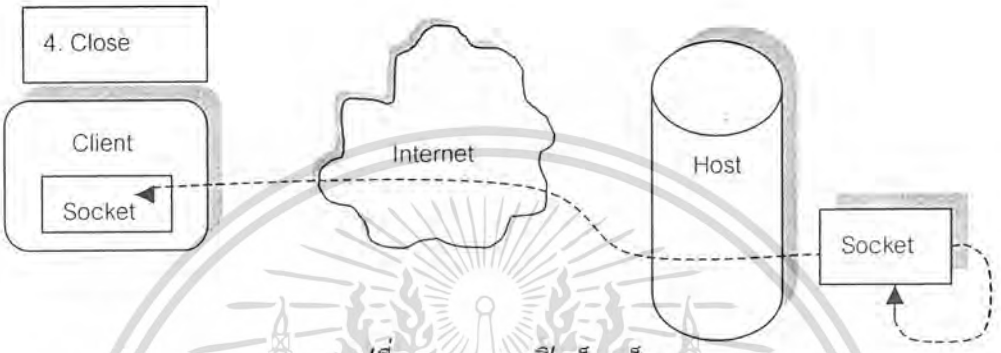


รูปที่ 7-6 แสดงการสร้างเส้นทางติดต่อ (Pipe line) และการสร้างบัฟเฟอร์

ขั้นตอนที่ 4 เป็นขั้นตอนสุดท้ายเมื่อผู้ใช้บริการต้องการปิดการติดต่อ จากโค้ดข้างล่างแสดงโปรแกรมการปิดการติดต่อ

```
ClientSocket.close();
```

จากตัวอย่างของโค้ดข้างบนทำให้เกิดการทำงานดังนี้



รูปที่ 7-7 แสดงการปิดซ็อกเก็ต

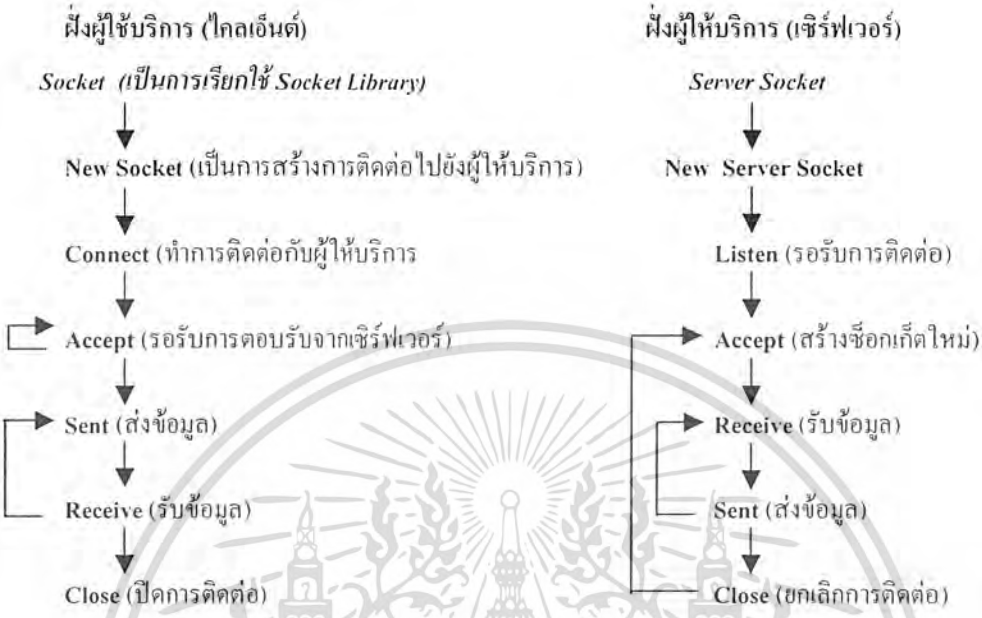
เมื่อผู้ใช้บริการทำการปิดซ็อกเก็ตตัวเองก็ส่งผลให้ผู้ให้บริการปิด Port และซ็อกเก็ตของตัวเองด้วยเช่นกัน

7.3.1 ภาพรวมของฟังก์ชันต่าง ๆ ที่ใช้สำหรับการติดต่อกับผู้ให้บริการ

เมื่อซ็อกเก็ตสร้างขึ้น สามารถที่จะเป็นได้ทั้งผู้รับการติดต่อหรือผู้สร้างการติดต่อ ซ็อกเก็ตที่ผู้ให้บริการใช้จะคอยรับการติดต่อเข้ามา เรียกว่า **Passive Socket** ขณะที่ซ็อกเก็ตที่ถูกใช้โดยผู้ใช้บริการ เรียกว่า **Active Socket** ซึ่งทั้งสองแบบนี้มีการสร้างเหมือนกัน แต่แตกต่างกันที่การใช้งาน

ในการใช้ซ็อกเก็ตในการติดต่อจะต้องมีการเรียกใช้ System Call โดยมีการส่งผ่านค่าพารามิเตอร์ที่จำเป็นลงไป ซึ่งจะใช้ฟังก์ชันของซ็อกเก็ต API ในการเรียก System Call ต่างๆ ดังต่อไปนี้

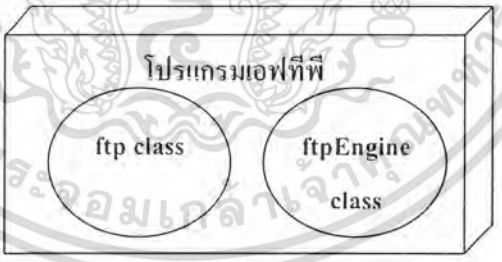
1. ***ฟังก์ชัน Socket*** เป็นคลาส Library ที่ใช้สำหรับสร้างการติดต่อกับฝั่งผู้ให้บริการ โดยจะต้องมีการระบุถึงตัวแปลต่าง ๆ เช่น หมายเลข IP ของเซิร์ฟเวอร์, พอร์ตที่ต้องการติดต่อเป็นต้น
2. ***ฟังก์ชัน Connect*** หลังจากที่สร้าง ซ็อกเก็ต แล้ว ผู้ใช้บริการจะเรียกฟังก์ชันนี้ เพื่อทำการติดต่อไปยังเซิร์ฟเวอร์ที่ต้องการ
3. ***ฟังก์ชัน Sent*** ทั้งผู้ใช้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการส่งข้อมูล โดยจะก็อบปี้ข้อมูลที่จะส่งลงในบัฟเฟอร์เพื่อที่จะส่งออกไป
4. ***ฟังก์ชัน Receive*** ทั้งผู้ใช้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการรับการข้อมูล
5. ***ฟังก์ชัน Close*** ฟังก์ชันนี้จะใช้สำหรับจบการใช้ซ็อกเก็ตในแต่ละอัน
6. ***ฟังก์ชัน Accept*** เป็นฟังก์ชันที่ผู้ใช้บริการใช้เปิดซ็อกเก็ตใหม่ เพื่อตอบรับการสร้างคู่ติดต่อของผู้ใช้บริการแต่ละคน
7. ***ฟังก์ชัน Listen*** เป็นฟังก์ชันเพื่อที่จะรอการติดต่อจากผู้ใช้บริการ
8. ***ฟังก์ชัน ServerSocket*** เป็นคลาส Library ที่ใช้สำหรับรับการร้องขอจากผู้ขอใช้บริการ



รูปที่ 7-8 แสดงลำดับการจัดการกับซ็อกเก็ตบนทีซีพี/ไอพี

7.4 รายละเอียดของการพัฒนา

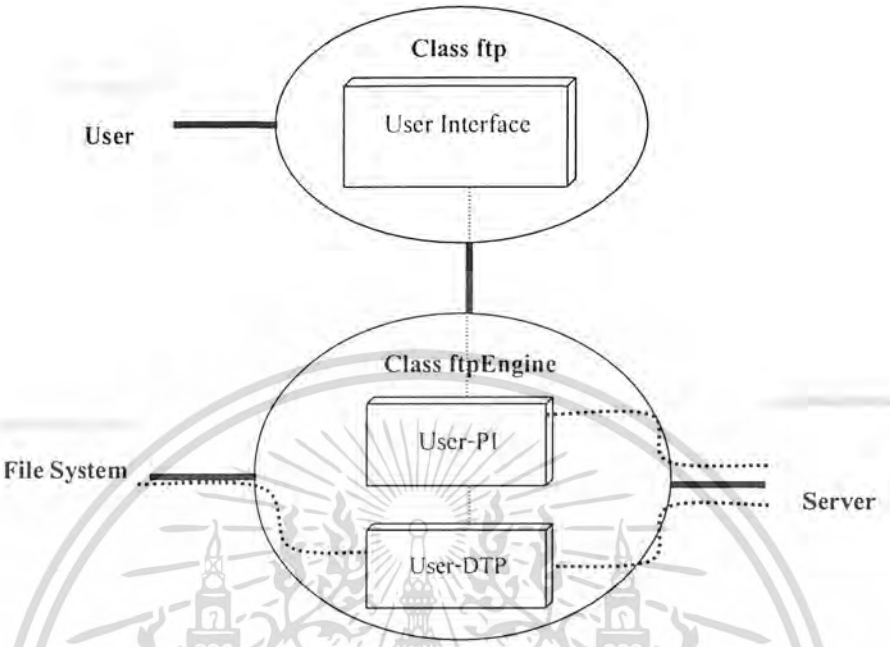
7.4.1 ขั้นตอนในการสร้างโปรแกรมเอฟทีพี



รูปที่7-9 โครงสร้างของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น

โปรแกรม FTP ที่สร้างขึ้นมามี 2 คลาสหลัก คือ ftpEngine กับ ftp

คลาส ftpEngine จะเป็นส่วนหลักที่จะจัดการทางด้านเน็ตเวิร์ก ทำหน้าที่ในการเปิดการเชื่อมต่อระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ ส่งคำสั่งเอฟทีพีมาตรฐานและรับการตอบกลับพร้อมทั้งแปลความหมาย (User-PI) และเป็นส่วนดูแลการโอนย้ายแฟ้มข้อมูล (User-DTP) ซึ่งจะถูกสั่งงานผ่านทางคลาส ftp



รูปที่ 7-10 แสดงการทำงานของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น

คลาส ftp จะเป็นคลาสที่ถูกออกแบบมาเพื่อรับคำสั่งจากผู้ใช้ทางคอมมานไลค์ และแสดงผลการตอบกลับต่างๆ ที่เกิดขึ้นในการใช้งานโปรแกรมเอฟทีพี (User Interface) โดยจะทำงานผ่านทางคลาส ftpEngine ซึ่งในจุดนี้ทำให้สามารถพัฒนาส่วนนี้ขึ้นมาใหม่เพื่อรูปแบบการใช้งานในแบบอื่นๆ เช่นอินเทอร์เฟซแบบกราฟิก โดยที่ไม่ต้องจัดการกับส่วนการโอนย้ายข้อมูลเลย

การรับข้อมูลจากผู้ใช้

คำสั่งที่ใช้มักจะเป็นคำสั่งที่เหมือนกับโปรแกรมเอฟทีพีธรรมดาทั่วไป โดยรับคำสั่งผ่านทาง standard input คือ System.in โดยจะถูกครอบด้วยคลาส InputStreamReader และ BufferdReader เพื่อให้การรับข้อมูลมีประสิทธิภาพ

ตัวอย่างนี้เป็นโค้ดที่แสดงการสร้างคลาสสำหรับใช้ในการรับข้อมูลจากผู้ใช้

```
BufferedReader  keyboardInput = new BufferedReader(new InputStreamReader(System.in));
```

การรับข้อมูลจะรับโดยการเรียกใช้เมธอด ReadLine เพื่อรอรับตัวอักษรจากผู้ใช้ ตัวอักษรที่รับมาจากผู้ใช้นั้นจะถูกนำไปแยกเอาส่วนที่เป็นคำสั่งเพื่อนำมาเปรียบเทียบกับคำสั่งที่กำหนดไว้ว่ามีคำสั่งนั้นหรือไม่ หากมีก็จะทำตามคำสั่งนั้น ส่วนที่ตามหลังคำสั่งก็จะถูกนำไปใช้เป็นข้อมูลที่คำสั่งนั้นต้องใช้ต่อไป หลังจากทำงานตามคำสั่งเสร็จก็จะรอรับคำสั่งถัดไปจากผู้ใช้เรื่อยๆ จนกว่าผู้ใช้จะออกจากการทำงาน

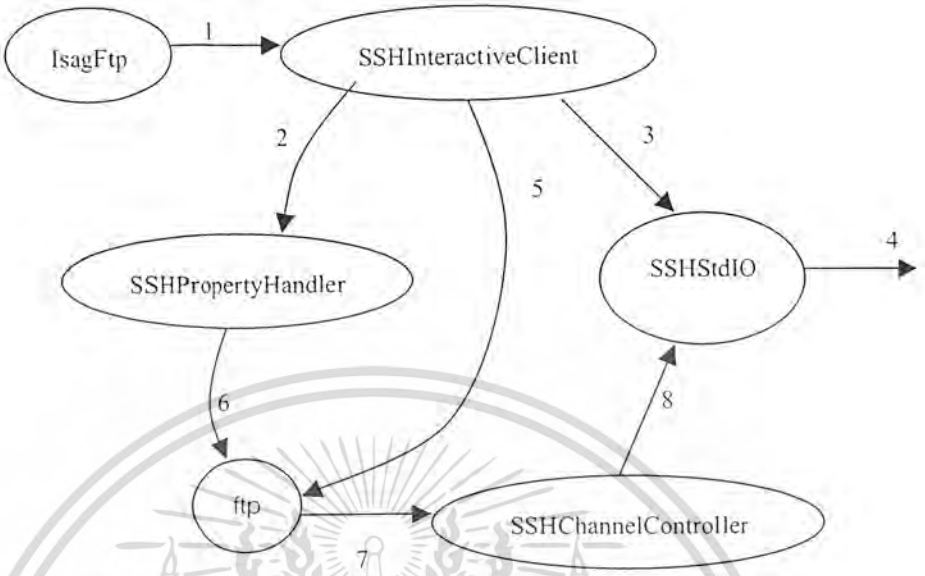
การแสดงผลทางหน้าจอ

การแสดงผลทางหน้าจอจะใช้คลาส System.out เท่านั้นเนื่องจากโปรแกรมที่พัฒนาขึ้นมามีการทำงานเป็นคอมมานไลด์โดยเรียกใช้เมธอด print หรือ println แล้วแต่กรณี

7.4.2 ขั้นตอนการสร้างโปรแกรม IsagFTP

คือการประยุกต์ใช้โพรโตคอลเซ็กเคียวเชลล์กับโปรแกรม ผู้ใช้จะพิมพ์คำสั่งผ่านอินเทอร์เฟซของโปรแกรมเซ็กเคียวเชลล์ ในตอนแรกของการทำงานจะต้องมีการตกลงกันระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ตามโพรโตคอลของเซ็กเคียวเชลล์ก่อน ดังนั้นจะไม่มีการทำงานของโปรแกรมเอฟทีพีเลย หลังจากที่มีการตกลงกันตามโพรโคคอลเซ็กเคียวเชลล์แล้ว โปรแกรมเซ็กเคียวเชลล์จะทำผ่านการทำงานให้กับโปรแกรม เอฟทีพี โปรแกรมเอฟทีพีจะทำหน้าที่เหมือนกับโปรแกรมเอฟทีพีทั่วไปเพียงแต่ข้อมูลที่ส่งออกไปยังเครือข่าย จะต้องผ่านการเข้ารหัสจากโพรโตคอลเซ็กเคียวเชลล์เสียก่อน โดยมีคลาสที่จัดการตามโพรโตคอลของเซ็กเคียวเชลล์ดังนี้

- **IsagFtp** คลาสนี้จะทำหน้าที่จัดการเริ่มต้นค่าต่างๆ จากอาร์กิวเมนต์ตอนเรียกใช้โปรแกรม หากไม่มีอาร์กิวเมนต์ก็จะนำค่าที่เป็นค่าที่กำหนดไว้ในโปรแกรมก่อนแล้วมาใช้ หลังจากที่มีการเซตค่าเป็นที่เรียบร้อยแล้วก็จะทำการเริ่มการทำงาน โดยเรียกใช้คลาส SSHInteractiveClient ต่อไป
- **SSHInteractiveClient** คลาสนี้จะเริ่มต้นการทำงานตามโพรโตคอลของเซ็กเคียวเชลล์ก่อน หลังจากนั้นจะทำการพิสูจน์ตน หากทางเซิร์ฟเวอร์ไม่มีการรองรับโพรโตคอลเซ็กเคียวเชลล์ก็จะเลิกการทำงานและผ่านการทำงานไปให้โปรแกรมเอฟทีพีธรรมดา ซึ่งจะไม่มีการเข้ารหัสแต่อย่างใด แต่ถ้าหากมีการรองรับแล้ว ก็จะเริ่มผ่านการทำงานให้โปรแกรมเอฟทีพีธรรมดาและข้อมูลที่ถูกส่งก็จะมีการเข้ารหัสตลอด
- **SSHStdIO** คลาสนี้จะรับตัวอักษรที่พิมพ์มาจากผู้ใช้ จะใช้ตอนช่วงการพิสูจน์ตนโดยการใส่ชื่อผู้ใช้และรหัสผ่านสำหรับโพรโตคอลของเซ็กเคียวเชลล์
- **SSHPropertyHandler** คลาสนี้จะเก็บค่าคุณสมบัติต่างๆ ของโปรแกรมไว้ใช้ เช่น ชื่อโฮสต์ที่ติดต่ออยู่ ชื่อผู้ใช้งานในขณะนั้น ชนิดของการเข้ารหัสที่ใช้อยู่ พอร์ตที่ใช้ในการฟอร์เวิร์ด โดยจะถูกเก็บจากคลาส IsagFtp เพื่อสามารถนำออกมาใช้ได้เมื่อเรียกใช้งานคลาสนี้
- **SSHChannelController** คลาสนี้จะจัดการกับเกี่ยวกับการรอรับการติดต่อจากโปรแกรมเอฟทีพี เพื่อนำข้อมูลมาทำการเข้ารหัส และ ทำหน้าที่ฟอร์เวิร์ดข้อมูลนั้นผ่านทางช่องทางที่สร้างขึ้นมา โดยทำหน้าที่ควบคุมการสร้างและทำลายช่องทางที่สร้างขึ้นมา



รูปที่ 7-11 แสดงการทำงานของโปรแกรม IsagFTP ที่ได้พัฒนาขึ้น

1. คลาส IsagFtp จะส่งข้อมูลไปให้ SSHInteractiveClient ช่วงนี้เป็นช่วงในการรับส่งข้อมูลระหว่างกัน ซึ่งข้อมูลต่างๆ ไม่ว่าจะเป็นชื่อผู้ใช้หรือรหัสผ่านยังไม่มีการเข้ารหัส
2. คลาส SSHPropertyHandler จะรับค่าคุณสมบัติต่างๆ เก็บไว้ใช้สำหรับโปรแกรมเอฟทีพี เช่น ชื่อผู้ใช้, รหัสผ่าน, ชื่อโฮสต์ที่ต้องการติดต่อ รวมทั้งพอร์ตที่ต้องการติดต่อด้วย
3. คลาส SSHStdIO จะรับข้อมูลเข้ามาจาก SSHInterractiveClient และพร้อมที่จะส่งออกไปและทำการพิสูจน์ตนกับโฮสต์
4. ข้อมูลถูกส่งออกสู่เครือข่าย ในช่วงการตกลงกันตามโพรโตคอลเซ็กเคียวเชลล์ จะยังไม่มีการเข้ารหัส หลังจากที่มีการตกลงกันตามโพรโตคอลเซ็กเคียวเชลล์แล้วคลาส SSHstdIO จะมีการเข้ารหัสโดยการเข้ารหัสจะเป็นแบบใด จะนำค่าคุณสมบัติที่เซตไว้ในคลาส SSHPropertyHandler มาใช้
5. การทำงานของ SSHInteractiveClient จะเปลี่ยนการติดต่อจากที่ติดต่อไปยัง SSHStdIO โดยตรงไปยังคลาส ftp
6. คลาส ftp จะเริ่มการทำงานได้โดยรับค่าคุณสมบัติที่เซตไว้ในคลาส SSHPropertyHandler มาใช้ ชื่อโฮสต์และพอร์ตที่ต้องการติดต่อจะนำมาใช้เพื่อติดต่อไปยังโฮสต์ที่ต้องการ และชื่อผู้ใช้กับรหัสผ่านจะนำมาใช้ในการพิสูจน์ตนสำหรับการใช้โพรโตคอลเอฟทีพีอีกครั้ง
7. คลาส SSHChannelController จะทำหน้าที่ในการฟอร์เวิร์ดข้อมูล โดยจะรอรับการติดต่อจากคลาส ftp
8. ข้อมูลจะถูกส่งผ่านทางเครือข่ายโดยส่งให้คลาส SSHStdIO

บทที่ 8

การทดลองและผลการทดลอง

หลังจากที่ได้พัฒนาโปรแกรมตามที่ได้ออกแบบไว้ในบทที่ 7 จึงต้องมีการทดสอบโปรแกรมที่ได้พัฒนาโดยทดลองติดตั้งโปรแกรมกับเครื่องที่มีระบบปฏิบัติการต่างๆ กัน แล้วทดลองใช้โปรแกรมติดต่อเพื่อขอใช้งานเอฟทีพีกับเซิร์ฟเวอร์ที่มีการเปิดบริการเซ็กเคียวเชลล์

8.1 จุดประสงค์ของการทดลอง

1.ทดสอบการใช้งานของโปรแกรมที่พัฒนาขึ้นตามโพรโตคอลมาตรฐานของเซ็กเคียวเชลล์ว่าสามารถทำงานได้อย่างถูกต้องหรือไม่

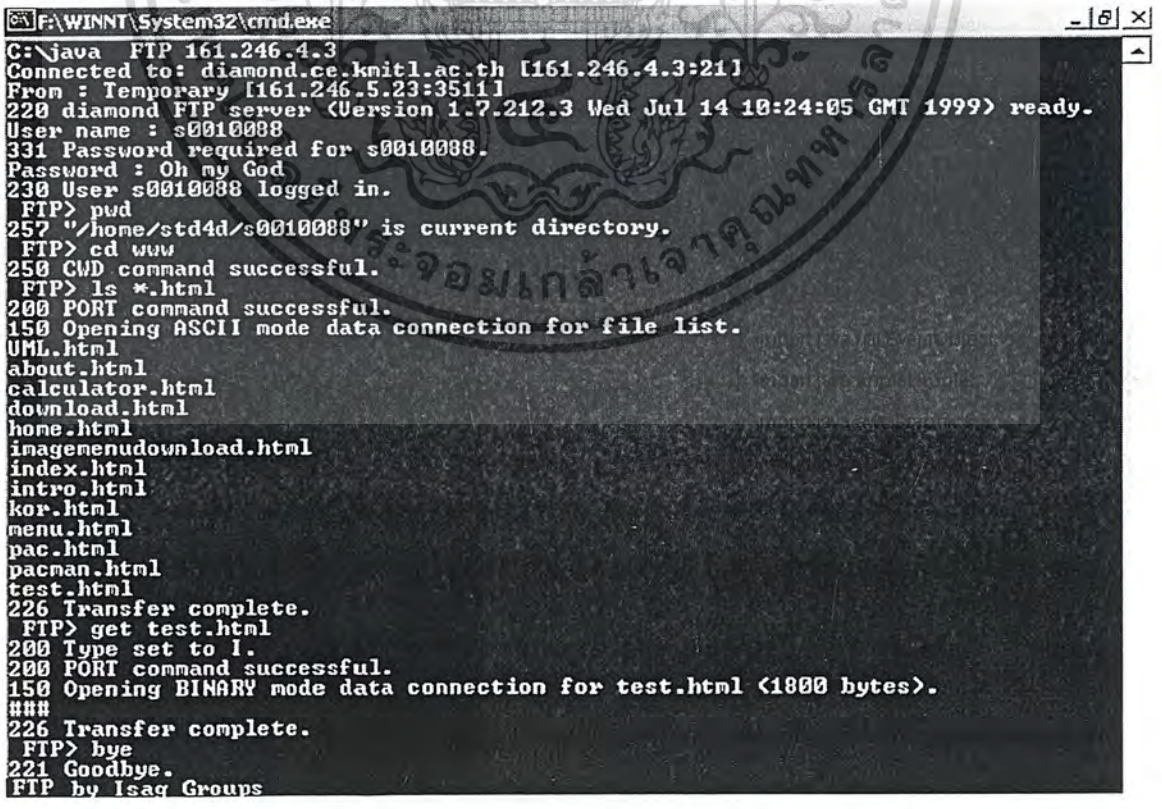
2.ทดสอบการใช้งานของโปรแกรมในแต่ระบบปฏิบัติการต่าง ๆ ที่มีการติดตั้ง JVM (Java Virtual Machine)

3.ตรวจสอบความถูกต้องของการใช้งานและขั้นตอนการทำงานของโปรแกรม IsagFTP

8.2 การทดลองการใช้งานโปรแกรม

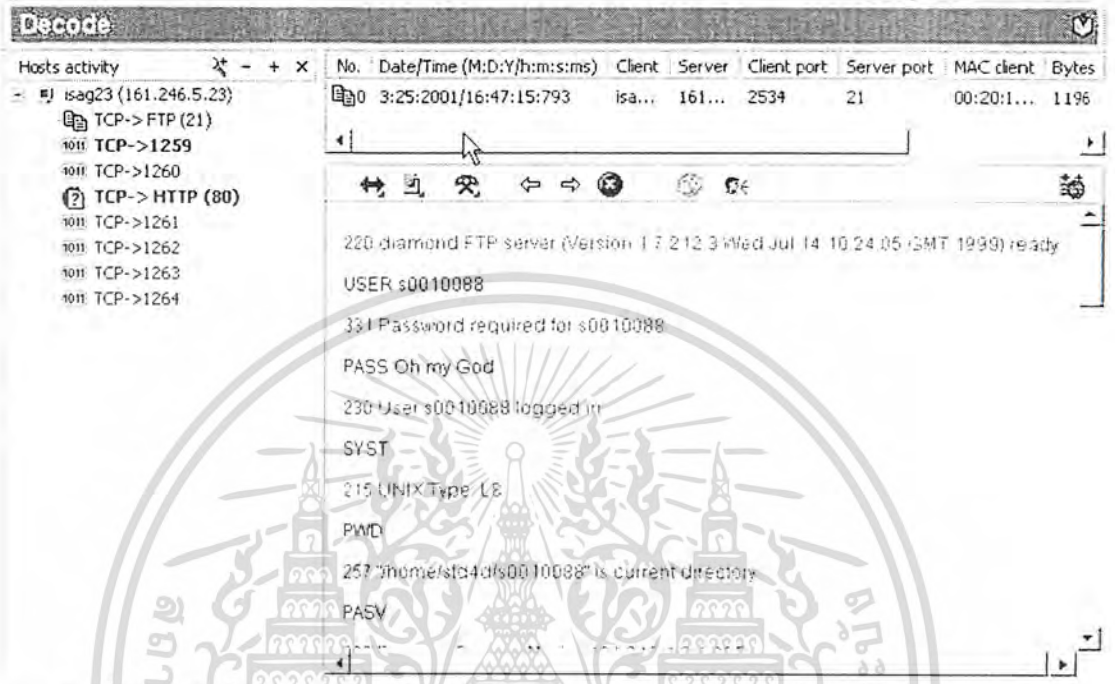
8.2.1 การใช้งานส่วนการโอนย้ายข้อมูลแต่เพียงส่วนเดียว

ในตอนแรกของการพัฒนาโปรแกรม IsagFTP นั้น ได้ทำการสร้างส่วนการโอนย้ายข้อมูลขึ้นมาก่อน ซึ่งมีลักษณะการใช้งาน เหมือนกับโปรแกรมเอฟทีพีไคลเอ็นต์ที่เป็นแบบคอมมานด์ไลด์ทั่วไป คือยังไม่มีการเข้ารหัสส่วนของข้อมูลที่มีรับส่งกันระหว่างเซิร์ฟเวอร์กับไคลเอ็นต์

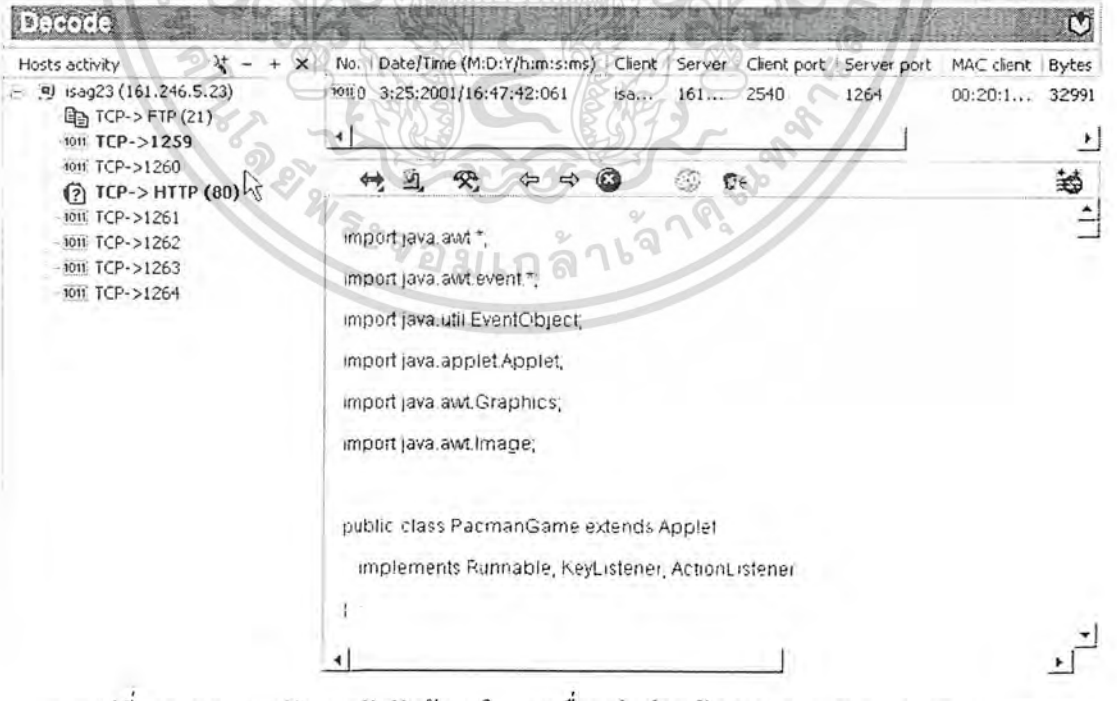


รูปที่ 8-1 แสดงการทำงานของโปรแกรมในการรับส่งข้อมูลที่พัฒนาขึ้น

ซึ่งข้อมูลที่การติดต่อสื่อสารกับเซิร์ฟเวอร์นั้นถูกลักลอบดักจับและตีความหมายได้เลย จากรูปที่ 8-2 แสดงการดักจับและตีความหมาย ข้อมูลในการเชื่อมต่อส่วนควบคุม ทั้งรหัสผู้ใช้ รหัสผ่าน คำสั่งต่าง ๆ ส่วนรูปที่8-3 แสดงการดักจับและตีความหมายของข้อมูลในการเชื่อมต่อส่วนข้อมูล ซึ่งในที่นี้เป็นการรับไฟล์ test.html มาจากเซิร์ฟเวอร์



รูปที่ 8-2 แสดงการลักลอบดักจับข้อมูลในการเชื่อมต่อส่วนควบคุม (Control Connection)

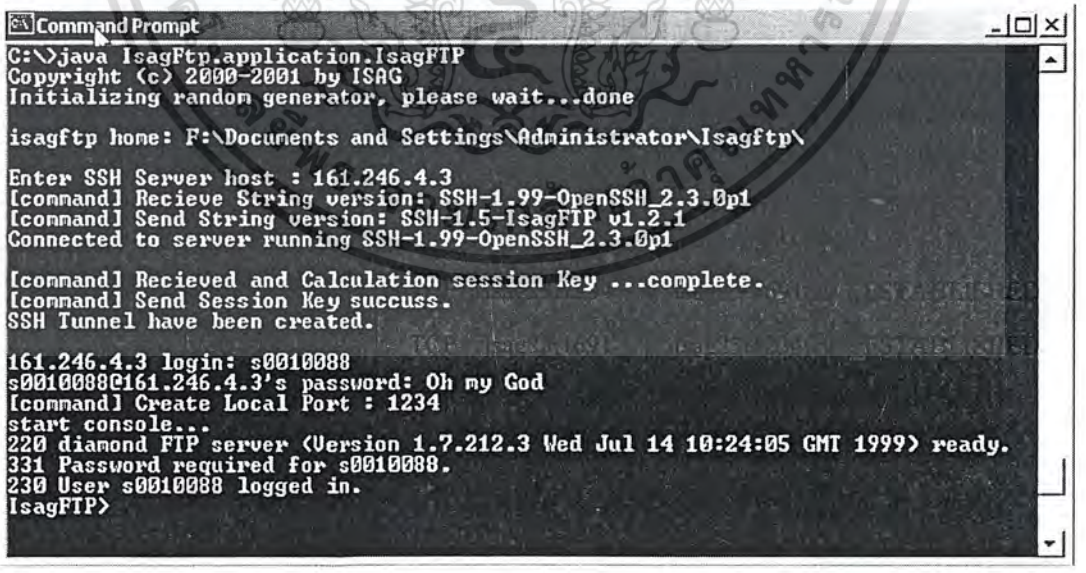


รูปที่ 8-3 แสดงการลักลอบดักจับข้อมูลในการเชื่อมต่อส่วนข้อมูล (Data Connection)

8.2.2 การทำงานโปรแกรม IsagFTP

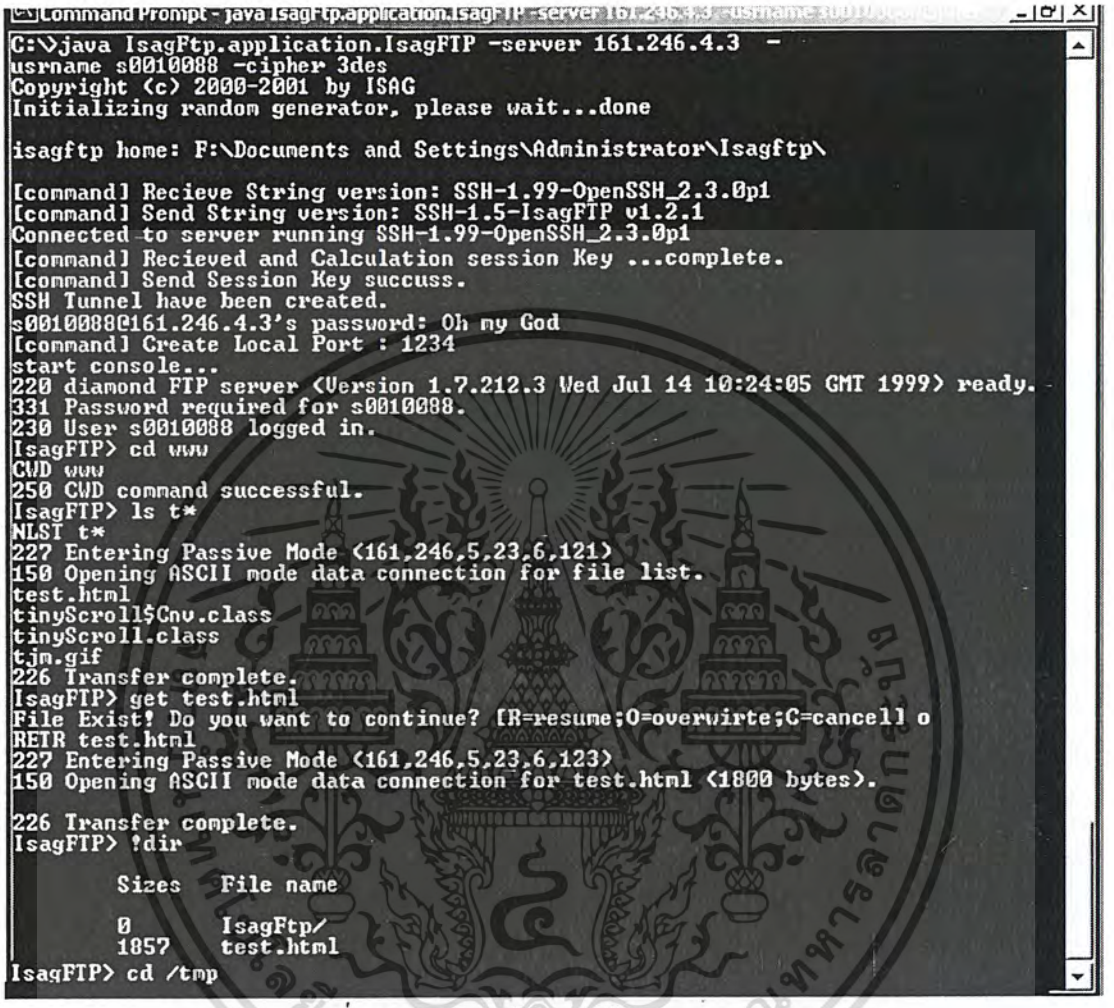
การทำงานของโปรแกรมนี้ มีลักษณะเหมือนกับการทำงานของส่วนที่ทำหน้าที่ติดต่อกับเซ็กเคียวเซลล์เดมอนบนเซิร์ฟเวอร์เพื่อที่จะทำตามขั้นตอนของโพรโตคอลเซ็กเคียวเซลล์แล้วฟอร์เวิร์ดพอร์ต มายังเครื่องที่เป็นโลคอลก่อน หลังจากนั้นจึงเอาส่วนที่จัดการรับส่งข้อมูลไปเชื่อมต่อกับพอร์ตที่ฟอร์เวิร์ดมาอีกทีหนึ่ง สามารถแบ่งเป็นขั้นตอนย่อยๆ ได้ดังนี้

- **ช่วงการตรวจสอบเวอร์ชัน** เป็นการเริ่มการติดต่อไปที่พอร์ต 22 ของเครื่องเซิร์ฟเวอร์โดยจะส่งข้อความเวอร์ชันของฝั่งไคลเอนต์และเซิร์ฟเวอร์ให้แก่กัน เพื่อให้เกิดความเข้ากันได้
- **ช่วงการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัส** เมื่อตรวจสอบเวอร์ชันของโพรโตคอลที่ใช้เรียบร้อยแล้วก็จะเข้าสู่ขั้นตอนการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัสซึ่งกันและกัน โดยจะมีพับบลิกคีย์ เซิร์ฟเวอร์คีย์และข้อมูลอื่นๆ ที่จำเป็นในการเข้ารหัส โดยในแอพพลิเคชันนี้จะใช้การแลกเปลี่ยนคีย์ด้วยวิธีการ RSA
- **ช่วงการตรวจสอบผู้ใช้และพิสูจน์สิทธิ์** หลังจากตกลงวิธีการเข้ารหัสคีย์และขั้นตอนการติดต่อเรียบร้อย จะนำข้อมูลผู้ใช้และรหัสผ่านส่งไปยังเซิร์ฟเวอร์เพื่อตรวจสอบ ซึ่งขั้นตอนนี้จะมีการเข้ารหัสแล้วตามวิธีการที่ได้ตกลงกันในขั้นตอนที่แล้ว และการตรวจสอบสิทธิ์ของผู้ใช้จะตรวจสอบจากไฟล์พาสเวิร์ดที่เซิร์ฟเวอร์
- **ช่วงการฟอร์เวิร์ดพอร์ต** หลังจากที่พิสูจน์สิทธิ์เป็นที่เรียบร้อยแล้วก็จะทำการฟอร์เวิร์ดพอร์ตที่โลคอลโฮสต์พอร์ต 1234 ไปยังพอร์ต 21 ของเซิร์ฟเวอร์ แล้วก็นำโปรแกรมส่วนการรับส่งข้อมูลมาติดต่อกับพอร์ต 1234 อีกที ซึ่งจะเสมือนกับการติดต่อกับ พอร์ต 21 โดยผ่านโพรโตคอลเซ็กเคียวเซลล์



รูปที่ 8-4 แสดงขั้นตอนการทำงานของโปรแกรม IsagFTP

- ช่วงการรับส่งคำสั่งและข้อมูล เป็นส่วนการใช้งานจริงของผู้ใช้ ในการรับส่งข้อมูล ข้อมูลในส่วนนี้ทั้งหมดไม่ว่าจะเป็นคำสั่งที่ติดต่อไปยังเอฟทีพีเซิร์ฟเวอร์ หรือการรับส่งแฟ้มข้อมูลจะถูกเข้ารหัสเองโดยอัตโนมัติ จากพอร์ต 1234 ผ่านพอร์ต 22 ไปยังพอร์ต 21



รูปที่ 8-5 แสดงการใช้งานของโปรแกรม IsagFTP

Proto	Local Address	Foreign Address	State
TCP	Isag23:1234	Isag23:1691	ESTABLISHED
TCP	Isag23:1691	Isag23:1234	ESTABLISHED
TCP	Isag23:1688	Isag23:1689	TIME_WAIT
TCP	Isag23:1690	diamond.ce.kmitl.ac.th:22	ESTABLISHED
TCP	Isag23:1692	Isag23:1693	TIME_WAIT
TCP	Isag23:1694	Isag23:1695	ESTABLISHED
TCP	Isag23:1695	Isag23:1694	ESTABLISHED

รูปที่ 8-6 แสดงการเปิดพอร์ตเพื่อใช้งานของเครื่องไคลเอ็นต์

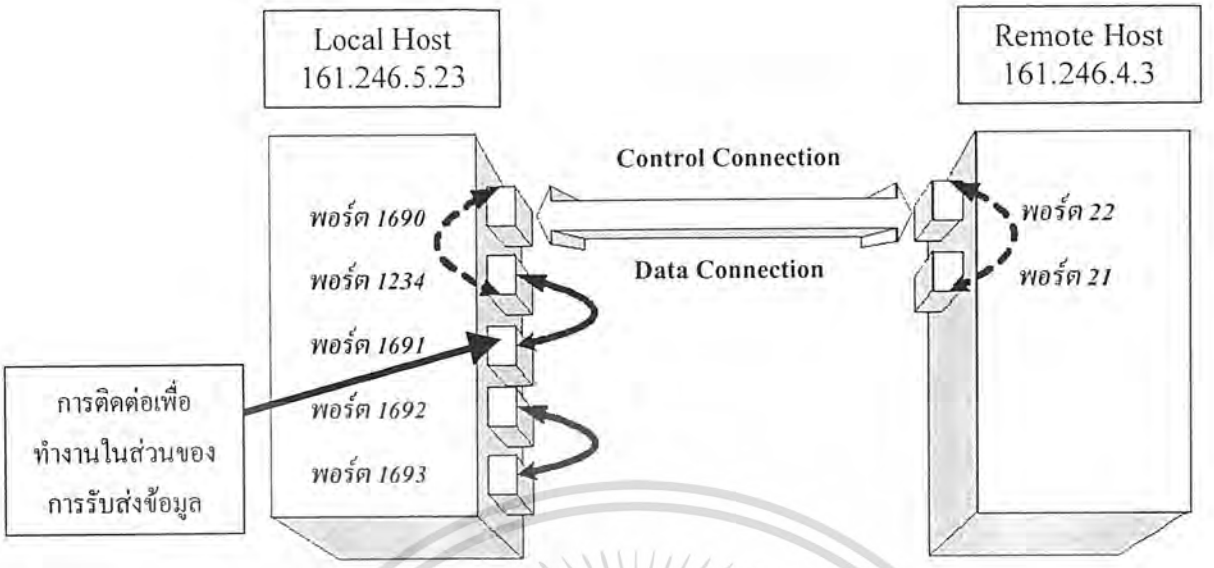
```
C:\Tool\FPortNG>fport
FPort v1.33 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com
Pid  Process        Port  Proto Path
1816 java       -> 1234  TCP   C:\JDK1.3\BIN\java.exe
1816 java       -> 1690  TCP   C:\JDK1.3\BIN\java.exe
1816 java       -> 1691  TCP   C:\JDK1.3\BIN\java.exe
1816 java       -> 1694  TCP   C:\JDK1.3\BIN\java.exe
1816 java       -> 1695  TCP   C:\JDK1.3\BIN\java.exe
```

รูปที่ 8-7 แสดงพอร์ตที่โปรแกรม IsagFTP เปิดขึ้นใช้งาน

```
bash-2.04$ netstat -a | grep isag23
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address      Foreign Address      (state)
tcp        0  32768 diamond.2726       diamond.2727         ESTABLISHED
tcp    32768      0 diamond.2727       diamond.2726         ESTABLISHED
tcp        0      0 diamond.ftp        diamond.2722         ESTABLISHED
tcp        0      0 diamond.2722       diamond.ftp          ESTABLISHED
tcp        0      0 diamond.22         isag23.1690          ESTABLISHED
```

รูปที่ 8-8 แสดงการเปิดพอร์ตของเซิร์ฟเวอร์ที่เข้าไปใช้บริการ

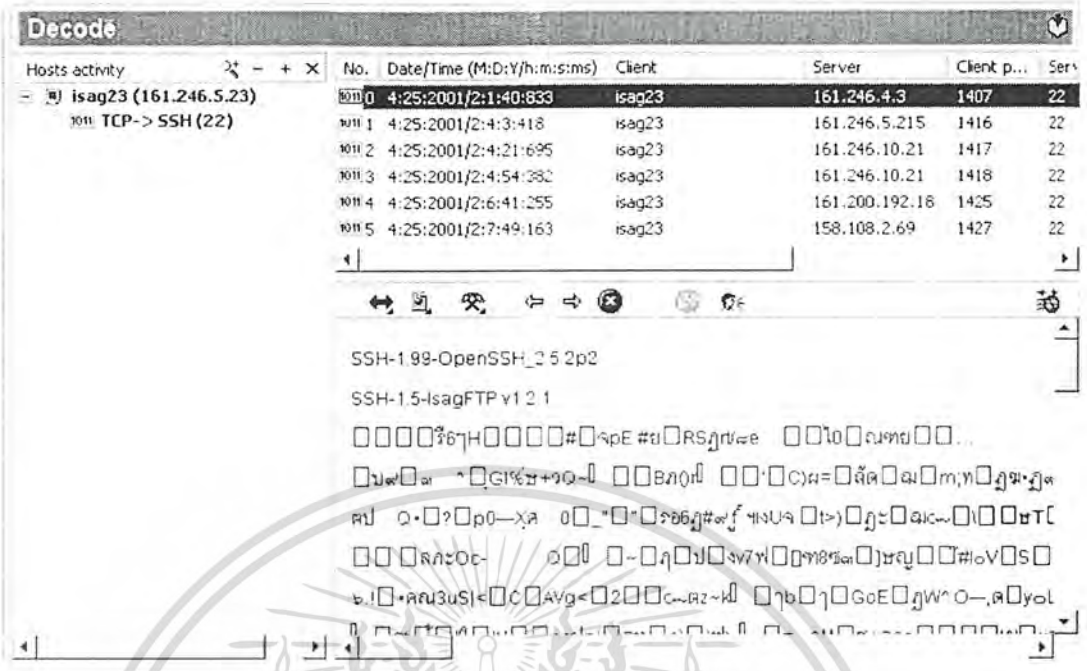
จากรูปที่ 8-6 , 8-7 และ 8-8 แสดงให้เห็นถึงการทำงานของโปรแกรม IsagFTP และการทำงานของพอร์ตฟอร์เวิร์ด ข้อมูลการเปิดพอร์ตข้างต้น เป็นตอนที่กำลังรับไฟล์มาจากเซิร์ฟเวอร์ ซึ่งต้องมีการสร้างการเชื่อมต่อส่วนข้อมูลด้วย จากรูปทั้ง 3 จะเห็นว่าข้อมูลทั้งหมดทั้งการเชื่อมต่อส่วนข้อมูลและการเชื่อมต่อส่วนควบคุมจะมีการส่งต่อผ่านไปยังพอร์ต 22 ของเซิร์ฟเวอร์ ดังนั้นข้อมูลทั้งหมดที่ติดต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์จะมีการเข้ารหัส ซึ่งจะมีลักษณะการทำงานเป็นดังรูปที่ 8-9



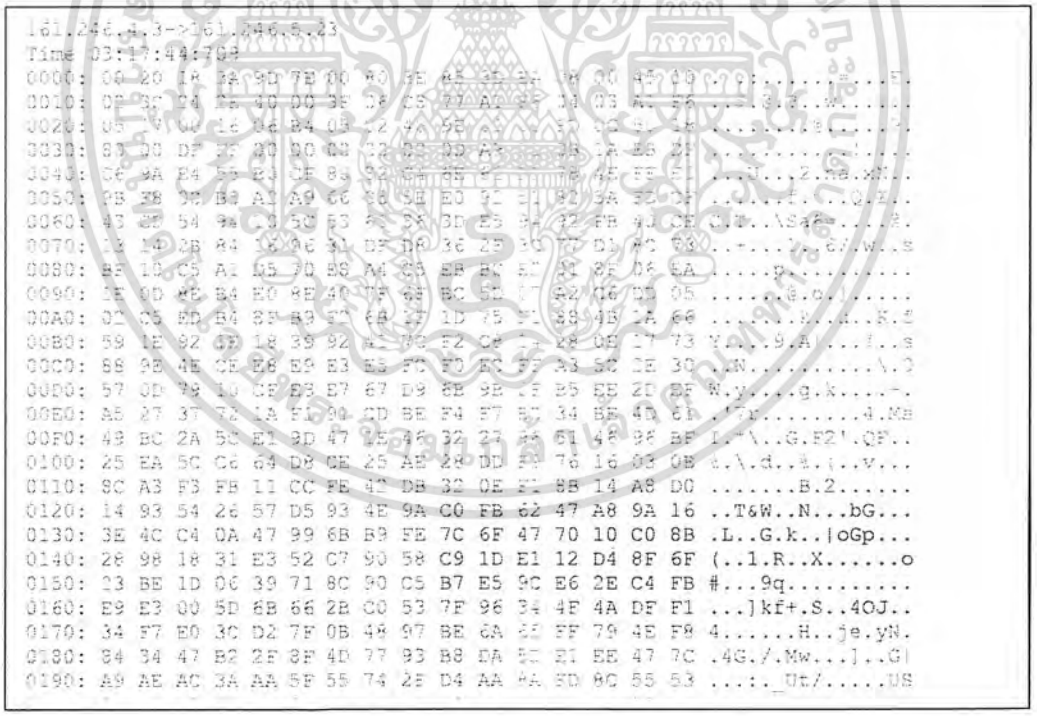
รูปที่ 8-9 แสดงรูปแบบการทำงานของโปรแกรม IsagFTP

1. เริ่มเมื่อรันโปรแกรมและระบุโฮสต์ที่จะติดต่อแล้ว จากนั้นก็จะติดต่อไปยังพอร์ต 22 ของโฮสต์ดังกล่าว เพื่อสร้างการเชื่อมต่อขึ้นและทำงานตามโพรโตคอลเซ็กเคียวเซลล์
2. เมื่อผ่านส่วนยูสเซอร์ออเทนติเคชั่นโพรโตคอลไปแล้ว ก็จะทำการฟอร์เวิร์ดพอร์ต 1234 ของโลคอลโฮสต์ (161.246.5.23) ไปยังพอร์ต 21 ของรีโมตโฮสต์ (161.246.4.3) ถ้าพอร์ต 1234 มีการใช้งานอยู่ก่อนแล้วจะเปิดพอร์ตอื่นโดยเพิ่มค่าพอร์ตทีละ 1 ไปเรื่อย ๆ จนกว่าจะพบพอร์ตที่ยังไม่มีการเปิดใช้งาน
3. จากนั้นส่วนของโปรแกรมที่ทำหน้าที่จัดการส่วนเอฟทีพีก็จะมาเชื่อมต่อกับพอร์ต 1234 ของโลคอลโฮสต์อีกที ซึ่งพอร์ตที่มีเชื่อมต่อในที่นี้คือ 1691
4. ต่อไปจะเป็นการออเทนติเคชั่นเพื่อเข้าใช้งานเอฟทีพีเซิร์ฟเวอร์ ผ่านพอร์ตที่ฟอร์เวิร์ดมาผ่านช่องทางการติดต่อสื่อสารที่มีความปลอดภัยของเซ็กเคียวเซลล์ไปยังพอร์ต 21
5. แล้วจะเข้าสู่การใช้งานของโปรแกรม ซึ่งไม่ว่าจะเป็นการเชื่อมต่อส่วนควบคุมหรือการเชื่อมต่อส่วนข้อมูล ข้อมูลทั้งหมดจะส่งต่อผ่านพอร์ต 1234 ผ่านการเชื่อมต่อของเซ็กเคียวเซลล์ไปยังเซิร์ฟเวอร์ที่ส่งต่อการทำงานของพอร์ตไปยังพอร์ต 21 ซึ่งเป็นการทำงานของเอฟทีพีเซิร์ฟเวอร์

จากรูปที่ 8-10 เป็นการแสดงการเชื่อมต่อระหว่างเครื่องไคลเอ็นต์กับเครื่องเซิร์ฟเวอร์ที่เกิดขึ้นจริง โดยมองจากผู้ที่กำลังลักลอบดักจับข้อมูลของเครื่องไคลเอ็นต์อยู่ ซึ่งจะเห็นได้ว่าการใช้โปรแกรม IsagFtp นั้นจะสร้างเชื่อมต่อมีเพียงช่องทางเดียวคือการเชื่อมต่อที่มีความปลอดภัยของเซ็กเคียวเซลล์ ทำให้ข้อมูลที่ลักลอบดักจับไปได้นั้นเป็นข้อมูลที่มีการเข้ารหัสแล้ว ซึ่งจะมีลักษณะเป็นดังรูปที่ 8-11



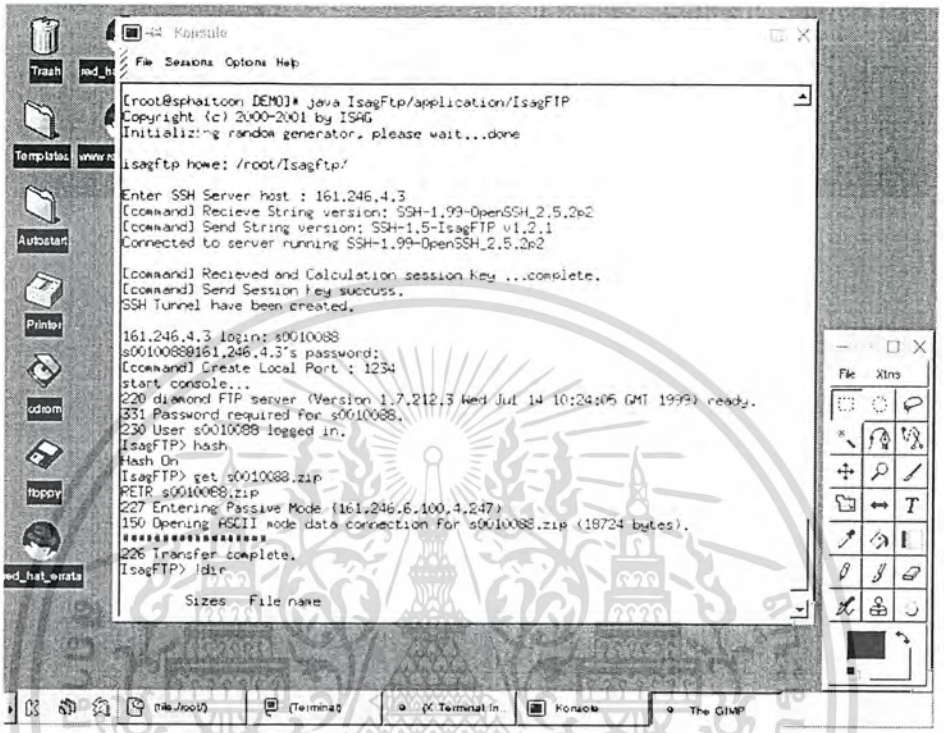
รูปที่ 8-10 แสดงถึงการเชื่อมต่อที่เกิดขึ้นระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์



รูปที่ 8-11 แสดงลักษณะของข้อมูลที่ดักจับมาได้จากการใช้งานของโปรแกรม IsagFTP

8.2.3 การทำงานในระบบปฏิบัติการต่างๆ

เนื่องจากโปรแกรม IsagFTP ได้พัฒนาขึ้นด้วยภาษาจาวา ทำให้โปรแกรมมีความสามารถในการทำงานในระบบปฏิบัติการใด ๆ ก็ได้ที่มีการติดตั้ง JVM (Java Virtual Machine) โดยที่ไม่ต้องมีการแก้ไขโปรแกรมต้นฉบับแต่อย่างใด



รูปที่ 8-12 การทำงานของโปรแกรมในระบบปฏิบัติการลีนุกซ์ Red hat 7.0

รูปที่ 8-13 การทำงานของโปรแกรมในระบบปฏิบัติการ MS Windows 2000 Professional

บทที่ 9

บทวิจารณ์และสรุป

ในการพัฒนาโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัยนี้ ได้นำโพรโตคอลเซ็กเคียวเชลล์มาประยุกต์ใช้ เพื่อสร้างความปลอดภัยในการติดต่อสื่อสารและได้สร้างโปรแกรมในส่วนของการรับส่งข้อมูล ตามมาตรฐานของโพรโตคอลเอฟทีพี ที่ได้มีการกำหนดไว้ในอาร์เอฟซี 959 (RFC 959) จากการศึกษาและพัฒนาโปรแกรมมาพอจะวิจารณ์และสรุปได้ดังนี้

9.1 การวิเคราะห์และสรุปมาตรฐานของโพรโตคอลเซ็กเคียวเชลล์

ในปัจจุบันการรักษาความปลอดภัยของข้อมูลเป็นประเด็นสำคัญ ในการติดต่อสื่อสารของระบบเครือข่าย ถึงแม้ว่าเซ็กเคียวเชลล์ยังไม่ได้เป็นโพรโตคอลมาตรฐาน ทางด้านความปลอดภัย แต่ก็เป็นโพรโตคอลที่มีคนนิยมใช้เป็นจำนานมากจนได้รับเป็นมาตรฐานดีแฟกโต (Defacto Standard) โปรแกรมประยุกต์บนทีซีพี/ไอพีที่ต้องการความปลอดภัยของข้อมูลจะเลือกใช้ และเซิร์ฟเวอร์ส่วนใหญ่มักจะติดตั้งโปรแกรมเซ็กเคียวเชลล์ไว้สำหรับผู้ใช้ที่ต้องการรักษาความปลอดภัยและความเป็นส่วนตัวของตน ข้อดีของโพรโตคอลเซ็กเคียวเชลล์ มีดังนี้

- การปลอมแปลงไอพี ไม่สามารถกระทำได้ เนื่องจากมีการตรวจสอบการเข้าใช้งานของผู้ใช้มีความปลอดภัยสูง โดยการเข้ารหัส โฮสต์คีย์และ คุกกี้ที่สุ่มขึ้นป้องกันการปลอมแปลงไอพี
- ข้อมูลผู้ใช้และรหัสผ่านของผู้ใช้นั้นจะมีความปลอดภัยในการส่งที่สูงเนื่องจากมีการเข้ารหัสที่มีประสิทธิภาพสูงด้วยกรรมวิธีของ อาร์เอสเอ (RSA) ทำให้ยากต่อการถอดรหัส
- ข้อมูลที่ติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้นมีการเข้ารหัสที่หลากหลายตามความต้องการของผู้ใช้ฝั่งไคลเอ็นต์ ทำให้ยากต่อการถอดรหัสข้อมูลที่ดักจับมาได้เป็นอย่างมาก
- การโจมตีของบุคคลระหว่างกลาง (Man in the middle attack) ไม่สามารถทำได้เนื่องจากมีการใช้โฮสต์คีย์ และเซสชันคีย์

โพรโตคอลเซ็กเคียวเชลล์นั้นมีการพัฒนาต่อเนื่องโดยมี SSH Communications Security เป็นผู้ที่พัฒนาโปรแกรมและมี IETF (The Internet Engineering Task Force) เป็นผู้ดูแลเกี่ยวกับมาตรฐานอยู่ตลอด

9.2 การวิเคราะห์และสรุปมาตรฐานของโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย (IsagFTP)

ตัวโปรแกรมนั้นมีการทำงานเป็นแบบคอมมานไลด์ และทำงานได้ทั้งแบบลักษณะที่เป็นแบตช์ไฟล์ (batch file)สามารถรับอาร์กิวเมนต์ เพื่อความสะดวกและรวดเร็วในการใช้งาน การพัฒนาโปรแกรมนั้นแบ่งออกเป็น 2 ส่วนใหญ่ ๆ คือ ส่วนที่ทำตามขั้นตอนมาตรฐานของโพรโตคอลเซ็กเคียวและส่วนจัดการรับส่งข้อมูล

9.2.1 ส่วนจัดการรับส่งข้อมูล

ได้ออกแบบตามมาตรฐานของโพรโตคอลเอฟทีพีที่ระบุไว้ในอาร์เอฟซี 959 มีความต้องการขั้นต่ำตรงกับตามที่กำหนดไว้ คือ

ชนิดของข้อมูล – แอสกี

วิธีการส่ง -- สายข้อมูล (Stream)

โครงสร้างข้อมูล --ไฟล์ , เรคอร์ด

คำสั่ง -- USER , QUIT , PORT , TYPE , MODE , STRU , RETR , STOR , NOOP

และได้เพิ่มเติมคุณสมบัติต่าง ๆ เข้าไปอีก เช่น เพิ่มคำสั่งต่าง ๆ การใช้งานร่วมกับไวด์การ์ด การเชื่อมต่อไฟล์ที่ยังรับส่งไม่สมบูรณ์ การรับส่งไฟล์แบบมัลติเพิล (Multiple put , get : mput , mget) การอัปโหลด ดาวน์โหลดเป็นไดเรกทอรี

9.2.2 ส่วนที่ทำตามขั้นตอนมาตรฐานของโพรโตคอลเซ็กเคียวเชลล์

ส่วนนี้จะเป็นเหมือน โปรแกรมไคลเอ็นที่ติดต่อไปยังเซิร์ฟเวอร์ที่ให้บริการเซ็กเคียวเชลล์ไม่เกินเวอร์ชัน 2 ซึ่งออกแบบตามอินเทอร์เน็ต-ดราฟต์ ที่เกี่ยวข้องกับเซ็กเคียวเชลล์ แต่ยังมีข้อจำกัดหลายประการ

- การเข้ารหัสในโปรแกรมยังไม่ครอบคลุมอัลกอริทึมการเข้ารหัสทั้งหมดของเซ็กเคียวเชลล์เซิร์ฟเวอร์ที่มีให้บริการ แต่ได้ออกแบบไว้รองรับการเข้ารหัสประเภทอื่น ๆ แล้ว เพียงแต่นำคลาสของการเข้ารหัสประเภทนั้นมาเรียกใช้
- ยังไม่รองรับการทำงานกับโพรโตคอลเซ็กเคียวเชลล์เวอร์ชัน 2
- มีความช้าในการเข้ารหัสส่วนข้อมูล เนื่องจากโพรโตคอลเซ็กเคียวเชลล์เวอร์ชัน 1 นั้น ไม่ได้ออกแบบมาเพื่อการเข้ารหัสข้อมูลที่มีขนาดใหญ่ (ซึ่งมีในเวอร์ชัน 2)

9.3 แนวทางการพัฒนาต่อในอนาคต

- เพิ่มอัลกอริทึมการเข้ารหัสข้อมูลที่ยังไม่ครบถ้วน เพื่อให้ยากแก่การนำข้อมูลที่ลักลอบดักจับได้ไปทำการถอดรหัส
- พัฒนาให้รองรับโพรโตคอลเซ็กเคียวเชลล์เวอร์ชัน 2 เพิ่มขึ้นด้วย
- สร้างการอินเทอร์เฟซแบบกราฟิกเพิ่มขึ้นเพื่อความสะดวกในการใช้งาน
- พัฒนาให้มีการเลือกเข้ารหัสเฉพาะการเชื่อมต่อส่วนควบคุม เพื่อความรวดเร็วในการโอนย้ายข้อมูลในการเชื่อมต่อส่วนข้อมูล

ภาคผนวก ก
รายละเอียดของแพ็กเก็ตเซ็กเคียวเชลล์

SSH Packet Description

ในการติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ในแต่ละแพ็กเก็ตที่ติดต่อกันจะมีชนิดที่แตกต่างกัน ขึ้นอยู่กับข้อมูลที่จะส่ง ซึ่งแพ็กเก็ตแต่ละชนิดจะมีเบอร์เฉพาะ เพื่อให้ทั้งเซิร์ฟเวอร์และไคลเอนต์เข้าใจตรงกัน

สำหรับข้อความที่บ่งบอกชนิดของแพ็กเก็ตในแต่ละเบอร์นี้ ข้อความที่มี _MSG_ ในชื่อเป็นข้อความที่ทั้งสองฝั่ง (ไคลเอนต์และเซิร์ฟเวอร์) สามารถส่งได้ ข้อความที่มี _CMSG_ หมายถึงไคลเอนต์เท่านั้นที่ส่งแพ็กเก็ตชนิดนี้ได้ และข้อความที่มี _SMSG_ หมายถึงเซิร์ฟเวอร์เท่านั้นที่ส่งแพ็กเก็ตชนิดนี้

0 SSH_MSG_NONE

This code is reserved. This message type is never sent.

1 SSH_MSG_DISCONNECT

string Cause of disconnection

This message may be sent by either party at any time. It causes the immediate disconnection of the connection. The message is intended to be displayed to a human, and describes the reason for disconnection.

2 SSH_SMSG_PUBLIC_KEY

8 bytes	anti_spoofing_cookie
32-bit int	server_key_bits
mp-int	server_key_public_exponent
mp-int	server_key_public_modulus
32-bit int	host_key_bits
mp-int	host_key_public_exponent
mp-int	host_key_public_modulus
32-bit int	protocol_flags
32-bit int	supported_ciphers_mask
32-bit int	supported_authentications_mask

Sent as the first message by the server. This message gives the server's host key, server key, protocol flags (intended for compatible protocol extension), supported_ciphers_mask (which is the bitwise or of (1 << cipher_number), where << is the left shift operator, for all supported ciphers), and supported_authentications_mask (which is the bitwise or of (1 << authentication_type) for all supported authentication types).The anti_spoofing_cookie is 64 random bits, and must be sent back verbatim by the client in its reply. It is used to make IP-spoofing more difficult (encryption and host keys are the real defense against spoofing).

3 SSH_CMSG_SESSION_KEY

1 byte	cipher_type (must be one of the supported values)
8 bytes	anti_spoofing_cookie (must match data sent by the server)
mp-int	double-encrypted session key
32-bit int	protocol_flags

Sent by the client as the first message in the session. Selects the cipher to use, and sends the encrypted session key to the server. The anti_spoofing_cookie must be the same bytes that were sent by the server. Protocol_flags is intended for negotiating compatible protocol extensions.

4 SSH_CMSG_USER

string	user login name on server

Sent by the client to begin authentication. Specifies the user name on the server to log in as. The server responds with SSH_SMSG_SUCCESS if no authentication is needed for this user, or SSH_SMSG_FAILURE if authentication is needed (or the user does not exist). [Note to the implementator: the user name is of arbitrary size. The implementation must be careful not to overflow internal buffers.]

5 SSH_CMSG_AUTH_RHOSTS

string	client-side user name

Requests authentication using /etc/hosts.equiv and .rhosts (or equivalent mechanisms). This authentication method is normally disabled in the server because it is not secure (but this is the method used by rsh and rlogin). The server responds with SSH_SMSG_SUCCESS if authentication was successful, and SSH_SMSG_FAILURE if access was not granted. The server should check that the client side port number is less than 1024 (a privileged port), and immediately reject authentication if it is not. Supporting this authentication method is optional. This method should normally not be enabled in the server because it is not safe. (However, not enabling this only helps if rlogind and rshd are disabled.)

6 SSH_CMSG_AUTH_RSA

mp-int	identity_public_modulus

Requests authentication using pure RSA authentication. The server checks if the given key is permitted to log in, and if so, responds with SSH_SMSG_AUTH_RSA_CHALLENGE. Otherwise, it responds with SSH_SMSG_FAILURE. The client often tries several different keys in sequence until one supported by the server is found. Authentication is accepted if the client gives the correct response to the challenge. The server is free to add other criteria for authentication, such as a requirement that the connection must come from a certain host. Such additions are not visible at the protocol level. Supporting this authentication method is optional but recommended.

7 SSH_SMSG_AUTH_RSA_CHALLENGE

mp-int encrypted challenge

8 SSH_CMSG_AUTH_RSA_RESPONSE

16 bytes MD5 of decrypted challenge

This message is sent by the client in response to an RSA challenge. The MD5 checksum is returned instead of the decrypted challenge to deter known-plaintext attacks against the RSA key. The server responds to this message with either SSH_SMSG_SUCCESS or SSH_SMSG_FAILURE.

9 SSH_CMSG_AUTH_PASSWORD

string plain text password

Requests password authentication using the given password. Note that even though the password is plain text inside the packet, the whole packet is normally encrypted by the packet layer. It would not be possible for the client to perform password encryption/hashing, because it cannot know which kind of encryption/hashing, if any, the server uses. The server responds to this message with SSH_SMSG_SUCCESS or SSH_SMSG_FAILURE.

10 SSH_CMSG_REQUEST_PTY

string TERM environment variable value (e.g. vt100)

32-bit int terminal height, rows (e.g., 24)

32-bit int terminal width, columns (e.g., 80)

32-bit int terminal width, pixels (0 if no graphics) (e.g., 480)

32-bit int terminal height, pixels (0 if no graphics) (e.g., 640)

n bytes tty modes encoded in binary

Requests a pseudo-terminal to be allocated for this command.This message can be used regardless of whether the session will later execute the shell or a command. If a pty has been requested with this message, the shell or command will run on a pty. Otherwise it will communicate with the server using pipes, sockets or some other similar mechanism.

The terminal type gives the type of the user's terminal. In the UNIX environment it is passed to the shell or command in the TERM environment variable.

The width and height values give the initial size of the user's terminal or window. All values can be zero if not supported by the operating system. The server will pass these values to the kernel if supported.

Terminal modes are encoded into a byte stream in a portable format. The exact format is described later in this document.

The server responds to the request with either SSH_SMSG_SUCCESS or SSH_SMSG_FAILURE. If the server does not have the concept of pseudo terminals, it should return

success if it is possible to execute a shell or a command so that it looks to the client as if it was running on a pseudo terminal.

---

11 SSH_CMSG_WINDOW_SIZE

32-bit int terminal height, rows

32-bit int terminal width, columns

32-bit int terminal width, pixels

32-bit int terminal height, pixels

This message can only be sent by the client during the interactive session. This indicates that the size of the user's window has changed, and provides the new size. The server will update the kernel's notion of the window size, and a SIGWINCH signal or equivalent will be sent to the shell or command (if supported by the operating system).

---

12 SSH_CMSG_EXEC_SHELL

(no arguments)

Starts a shell (command interpreter), and enters interactive session mode.

---

13 SSH_CMSG_EXEC_CMD

string command to execute

Starts executing the given command, and enters interactive session mode. On UNIX, the command is run as "<shell> -c <command>", where <shell> is the user's login shell.

---

14 SSH_SMSG_SUCCESS

(no arguments)

This message is sent by the server in response to the session key, a successful authentication request, and a successfully completed preparatory operation.

---

15 SSH_SMSG_FAILURE

(no arguments)

This message is sent by the server in response to a failed Authentication operation to indicate that the user has not yet been successfully authenticated, and in response to a failed preparatory operation. This is also sent in response to an authentication or preparatory operation request that is not recognized or supported.

---

16 SSH_CMSG_STDIN_DATA

string data

Delivers data from the client to be supplied as input to the shell or program running on the server side. This message can only be used in the interactive session mode. No acknowledgement is sent for this message.

17 SSH_SMSG_STDOUT_DATA

string data

Delivers data from the server that was read from the standard output of the shell or program running on the server side. This message can only be used in the interactive session mode. No acknowledgement is sent for this message.

18 SSH_SMSG_STDERR_DATA

string data

Delivers data from the server that was read from the standard Error of the shell or program running on the server side. This message can only be used in the interactive session mode. No acknowledgement is sent for this message.

19 SSH_CMSG_EOF

(no arguments)

This message is sent by the client to indicate that EOF has been reached on the input. Upon receiving this message, and after all buffered input data has been sent to the shell or program, the server will close the input file descriptor to the program.This message can only be used in the interactive session mode.No acknowledgement is sent for this message.

20 SSH_SMSG_EXITSTATUS

32-bit int exit status of the command

Returns the exit status of the shell or program after it has exited. The client should respond with SSH_CMSG_EXIT_CONFIRMATION when it has received this message.This will be the last message sent by the server. If the program being executed dies with a signal instead of exiting normally, the server should terminate the session with SSH_MSG_DISCONNECT (which can be used to pass a human-readable string indicating that the program died due to a signal) instead of using this message.

21 SSH_MSG_CHANNEL_OPEN_CONFIRMATION

32-bit int remote_channel

32-bit int local_channel

This is sent in response to any channel open request if the channel has been successfully opened. Remote_channel is the channel number received in the initial open request;local_channel is the channel number the side sending this message has allocated for the channel. Data can be transmitted on the channel after this message.

22 SSH_MSG_CHANNEL_OPEN_FAILURE

32-bit int remote_channel

This message indicates that an earlier channel open request by the other side has failed or has

been denied. Remote_channel is the channel number given in the original request.

23 SSH_MSG_CHANNEL_DATA

32-bit int remote_channel

string data

Data is transmitted in a channel in these messages. A channel is bidirectional, and both sides can send these messages. There is no acknowledgement for these messages. It is possible that either side receives these messages after it has sent SSH_MSG_CHANNEL_CLOSE for the channel. These messages cannot be received after the party has sent or received SSH_MSG_CHANNEL_CLOSE_CONFIRMATION.

24 SSH_MSG_CHANNEL_CLOSE

32-bit int remote_channel

When a channel is closed at one end of the connection, that side Sends this message. Upon receiving this message, the channel Should be closed. When this message is received, if the channel is already closed (the receiving side has sent this message for the same channel earlier), the channel is freed and no further action is taken; otherwise the channel is freed and SSH_MSG_CHANNEL_CLOSE_CONFIRMATION is sent in response. (It is Possible that the channel is closed simultaneously at both ends.)

25 SSH_MSG_CHANNEL_CLOSE_CONFIRMATION

32-bit int remote_channel

This message is sent in response to SSH_MSG_CHANNEL_CLOSE unless the channel was already closed. When this message is sent or received, the channel is freed.

26 (OBSOLETED; was unix-domain X11 forwarding)

27 SSH_SMSG_X11_OPEN

32-bit int local_channel

string originator_string (see below)

This message can be sent by the server during the interactive session mode to indicate that a client has connected the fake X server. Local_channel is the channel number that the server has allocated for the connection. The client should try to open a connection to the real X server, and respond with SSH_MSG_CHANNEL_OPEN_CONFIRMATION or SSH_MSG_CHANNEL_OPEN_FAILURE.The field originator_string is present if both sides specified SSH_PROTOFLAG_HOST_IN_FWD_OPEN in the protocol flags. It contains a description of the host originating the connection.

28 SSH_CMSG_PORT_FORWARD_REQUEST

32-bit int server_port

string host_to_connect

32-bit int port_to_connect

Sent by the client in the preparatory phase, this message requests that server_port on the server machine be forwarded over the secure channel to the client machine, and from there to the specified host and port. The server should start listening on the port, and send SSH_MSG_PORT_OPEN whenever a connection is made to it. Supporting this message is optional, and the server is free to reject any forward request. For example, it is highly recommended that unless the user has been authenticated as root, forwarding any privileged port numbers (below 1024) is denied.

---

29 SSH_MSG_PORT_OPEN

32-bit int local_channel

string host_name

32-bit int port

string originator_string (see below)

Sent by either party in interactive session mode, this message indicates that a connection has been opened to a forwarded TCP/IP port. Local_channel is the channel number that the sending party has allocated for the connection. Host_name is the host the connection should be be forwarded to, and the port is the port on that host to connect. The receiving party should open the connection, and respond with SSH_MSG_CHANNEL_OPEN_CONFIRMATION or SSH_MSG_CHANNEL_OPEN_FAILURE. It is recommended that the receiving side check the host_name and port for validity to avoid compromising local security by compromised remote side software. Particularly, it is recommended that the client permit connections only to those ports for which it has requested forwarding with SSH_CMSG_PORT_FORWARD_REQUEST.The field originator_string is present if both sides specified SSH_PROTOFLAG_HOST_IN_FWD_OPEN in the protocol flags. It contains a description of the host originating the connection.

---

30 SSH_CMSG_AGENT_REQUEST_FORWARDING

(no arguments)

Requests that the connection to the authentication agent be forwarded over the secure channel. The method used by clients to contact the authentication agent within each machine is implementation and machine dependent. If the server accepts this request, it should arrange that any clients run from this session will actually contact the server program when they try to contact the authentication agent. The server should then send a SSH_SMSG_AGENT_OPEN to open a channel to the agent, and the client should forward the connection to the real authentication agent. Supporting this message is optional.

---

31 SSH_SMSG_AGENT_OPEN

32-bit int local_channel.

Sent by the server in interactive session mode, this message requests opening a channel to the authentication agent. The client should open a channel, and respond with either SSH_MSG_CHANNEL_OPEN_CONFIRMATION or SSH_MSG_CHANNEL_OPEN_FAILURE

32 SSH_MSG_IGNORE

string data

Either party may send this message at any time. This message,and the argument string, is silently ignored. This message might be used in some implementations to make traffic analysis more difficult. This message is not currently sent by the implementation, but all implementations are required to recognize and ignore it.

33 SSH_CMSG_EXIT_CONFIRMATION

(no arguments)

Sent by the client in response to SSH_SMSG_EXITSTATUS. This is the last message sent by the client.

34 SSH_CMSG_X11_REQUEST_FORWARDING

string x11_authentication_protocol

string x11_authentication_data

32-bit int screen number (if SSH_PROTOFLAG_SCREEN_NUMBER)

Sent by the client during the preparatory phase, this message requests that the server create a fake X11 display and set the DISPLAY environment variable accordingly. An internet-domain display is preferable. The given authentication protocol and the associated data should be recorded by the server so that it is used as authentication on connections (e.g., in .Xauthority).The authentication protocol must be one of the supported X11 authentication protocols, e.g., "MIT-MAGIC-COOKIE-1". Authentication data must be a lowercase hex string of even length. Its interpretation is protocol dependent. The data is in a format that can be used with e.g. the xauth program. Supporting this message is optional.

The client is permitted (and recommended) to generate fake Authentication information and send fake information to the server. This way, a corrupt server will not have access to the user's terminal after the connection has terminated. The correct authorization codes will also not be left hanging around in files on the server (many users keep the same X session for months, thus protecting the authorization data becomes important).

X11 authentication spoofing works by initially sending fake (random) authentication data to the server, and interpreting the first packet sent by the X11 client after the connection has been opened.

The first packet contains the client's authentication. If the packet contains the correct fake data, it is replaced by the client by the correct authentication data, and then sent to the X server

35 SSH_CMSG_AUTH_RHOSTS_RSA

string	clint-side user name
32-bit int	client_host_key_bits
mp-int	client_host_key_public_exponent
mp-int	client_host_key_public_modulus

Requests authentication using /etc/hosts.equiv and .rhosts (or equivalent) together with RSA host authentication. The server should check that the client side port number is less than 1024 (a privileged port), and immediately reject authentication if it is not. The server responds with SSH_SMSG_FAILURE or SSH_SMSG_AUTH_RSA_CHALLENGE. The client must respond to the challenge with the proper SSH_CMSG_AUTH_RSA_RESPONSE. The server then responds with success if access was granted, or failure if the client gave a wrong response. Supporting this authentication method is optional but recommended in most environments.

36 SSH_MSG_DEBUG

string debugging message sent to the other side

This message may be sent by either party at any time. It is used to send debugging messages that may be informative to the user in solving various problems. For example, if authentication fails because of some configuration error (e.g., incorrect permissions for some file), it can be very helpful for the user to make the cause of failure available. On the other hand, one should not make too much information available for security reasons. It is recommended that the client provides an option to display the debugging information sent by the sender (the user probably does not want to see it by default). The server can log debugging data sent by the client (if any). Either party is free to ignore any received debugging data. Every implementation must be able to receive this message, but no implementation is required to send these.

37 SSH_CMSG_REQUEST_COMPRESSION

32-bit int gzip compression level (1-9)

This message can be sent by the client in the preparatory operations phase. The server responds with SSH_SMSG_FAILURE if it does not support compression or does not want to compress; it responds with SSH_SMSG_SUCCESS if it accepted the compression request. In the latter case the response to this packet will still be uncompressed, but all further packets in either direction will be compressed by gzip.

38 SSH_CMSG_MAX_PACKET_SIZE

32-bit int maximum packet size, bytes (4096-1024k)

This message can be sent by the client in the preparatory operations phase. The server responds with SSH_SMSG_FAILURE if it does not support limiting packet size, or with SSH_SMSG_SUCCESS if it has limited the maximum packet size (as determined by the value in the size field) to the specified value.

39 SSH_CMSG_AUTH_TIS

(no arguments)

This message starts TIS authentication. The server responds with SSH_SMSG_FAILURE or SSH_SMSG_AUTH_TIS_CHALLENGE.

40 SSH_SMSG_AUTH_TIS_CHALLENGE

string tis challenge

Server sends TIS challenge to user and client should show it to user and ask for response, which is sent back using SSH_CMSG_AUTH_TIS_RESPONSE message.

41 SSH_CMSG_AUTH_TIS_RESPONSE

string user response to tis challenge

When client receives SSH_SMSG_AUTH_TIS_CHALLENGE and ask users response to challenge it sends it back this message. The server answers with SSH_SMSG_FAILURE or SSH_SMSG_SUCCESS.

42 SSH_CMSG_AUTH_KERBEROS

string authentication info

Client sends authentication info to server, which replies with SSH_SMSG_AUTH_KERBEROS_RESPONSE message having correct response data encrypted with the session key.

43 SSH_SMSG_AUTH_KERBEROS_RESPONSE

string response data

Server replies to SSH_CMSG_AUTH_KERBEROS message with this message so that the response data is encrypted with session key.

44 SSH_CMSG_HAVE_KERBEROS_TGT

string kerberos credentials

Client sends kerberos credentials to server and the server replies with SSH_SMSG_SUCCESS or SSH_SMSG_FAILURE.

ภาคผนวก ข
รายละเอียดคำสั่งของเอฟทีพี

FTP COMMANDS

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD  <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
STOU <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
    [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD  <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
PWD  <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
```

ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

USER NAME (USER)

The argument field is a Telnet string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the control connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old access control parameters.

PASSWORD (PASS)

The argument field is a Telnet string specifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

ACCOUNT (ACCT)

The argument field is a Telnet string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the automation: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532 reply depending on whether it stores (pending receipt of the ACCounT command) or discards the command, respectively.

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

CHANGE TO PARENT DIRECTORY (CDUP)

This command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different syntaxes for naming the parent directory. The reply codes shall be identical to the reply codes of CWD. See Appendix II for further details.

STRUCTURE MOUNT (SMNT)

This command allows the user to mount a different file system data structure without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open. This is identical to the state in which a user finds himself immediately after the control connection is opened. A USER command may be expected to follow.

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not in progress, the server closes the control connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERs but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

An unexpected close on the control connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

Transfer Parameter Commands

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value is as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters:

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port to be used in data connection. There are defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:

```
PORT h1,h2,h3,h4,p1,p2
```

where h1 is the high order 8 bits of the internet host address.

PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a

connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:

```
                \   /
   A - ASCII |      | N - Non-print
             |->  <-| T - Telnet format effectors
   E - EBCDIC|      | C - Carriage Control (ASA)
                /   \
   I - Image

   L <byte size> - Local byte Byte size
```

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

FILE STRUCTURE (STRU)

The argument is a single Telnet character code specifying file structure described in the Section on Data Representation and Storage.

The following codes are assigned for structure:

```
   F - File (no record structure)
   R - Record structure
   P - Page structure
```

The default structure is File.

TRANSFER MODE (MODE)

The argument is a single Telnet character code specifying the data transfer modes described in the Section on Transmission Modes.

The following codes are assigned for transfer modes:

```
   S - Stream
   B - Block
   C - Compressed
```

The default transfer mode is Stream.

FTP Service Commands

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the control connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command (e.g., STOR or RETR). The data, when transferred in response to FTP service commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site, then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

STORE UNIQUE (STOU)

This command behaves like STOR except that the resultant file is to be created in the current directory under a name unique to that directory. The 250 Transfer Started response must include the name generated.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte

size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of the command. This second argument is optional, but when present should be separated from the first by the three Telnet characters <SP> R <SP>. This command shall be followed by a STORe or APPEnd command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but skips over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the old pathname of the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

ABORT (ABOR)

This command tells the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The control connection is not to be closed by the server, but the data connection must be closed.

There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated

abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "Do you really wish to delete?"), it should be provided by the user-FTP process.

REMOVE DIRECTORY (RMD)

This command causes the directory specified in the pathname to be removed as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

MAKE DIRECTORY (MKD)

This command causes the directory specified in the pathname to be created as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

PRINT WORKING DIRECTORY (PWD)

This command causes the name of the current working directory to be returned in the reply. See Appendix II.

LIST (LIST)

This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory or other group of files, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must ensure that the TYPE is appropriately ASCII or EBCDIC). Since the information on a file may vary widely from system to system, this information may be hard to use automatically in a program, but may be quite useful to a human user.

NAME LIST (NLST)

This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.) This command is intended to return information that can be used by a program to

further process the files automatically. For example, in the implementation of a "multiple get" function.

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

SYSTEM (SYST)

This command is used to find out the type of operating system at the server. The reply shall have as its first word one of the system names listed in the current version of the Assigned Numbers document [4].

STATUS (STAT)

This command shall cause a status response to be sent over the control connection in the form of a reply. The command may be sent during a file transfer (along with the Telnet IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case, the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be transferred over the control connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the control connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

The File Transfer Protocol follows the specifications of the Telnet protocol for all communications over the control connection. Since

the language used for Telnet communication may be a negotiated option, all references in the next two sections will be to the "Telnet language" and the corresponding "Telnet end-of-line code". Currently, one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the Telnet protocol will be cited.

FTP commands are "Telnet strings" terminated by the "Telnet end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and Telnet-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the control connection while a data transfer is in progress. Some servers may not be able to monitor the control and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The following ordered format is tentatively recommended:

1. User system inserts the Telnet "Interrupt Process" (IP) signal in the Telnet stream.

2. User system sends the Telnet "Synch" signal.

3. User system inserts the command (e.g., ABOR) in the Telnet stream.

4. Server PI, after receiving "IP", scans the Telnet stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

FTP COMMAND ARGUMENTS

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and
<LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any
              ASCII code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number> ::= any decimal integer 1 through 255
```

```
<form-code> ::= N | T | C
<type-code> ::= A [<sp> <form-code>]
              | E [<sp> <form-code>]
              | I
              | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code> ::= S | B | C
<pathname> ::= <string>
<decimal-integer> ::= any decimal integer
```

ภาคผนวก ค
รายละเอียดการตอบกลับของเอฟทีพี

Reply Codes by Function Groups

```
200 Command okay.
500 Syntax error, command unrecognized.
    This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
202 Command not implemented, superfluous at this site.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.
110 Restart marker reply.
    In this case, the text is exact and not left to the
    particular implementation; it must read:
         MARK yyyy = mmmm
    Where yyyy is User-process data stream marker, and mmmm
    server's equivalent marker (note the spaces between markers
    and "=").
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
    On how to use the server or the meaning of a particular
    non-standard command.  This reply is useful only to the
    human user.
215 NAME system type.
    Where NAME is an official system name from the list in the
    Assigned Numbers document.

120 Service ready in nnn minutes.
220 Service ready for new user.
221 Service closing control connection.
    Logged out if appropriate.
421 Service not available, closing control connection.
    This may be a reply to any command if the service knows it
    must shut down.
125 Data connection already open; transfer starting.
225 Data connection open; no transfer in progress.
425 Can't open data connection.
226 Closing data connection.
    Requested file action successful (for example, file
    transfer or file abort).
426 Connection closed; transfer aborted.
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.
530 Not logged in.
331 User name okay, need password.
332 Need account for login.
532 Need account for storing files.
 150 File status okay; about to open data connection.
250 Requested file action okay, completed.
257 "PATHNAME" created.
350 Requested file action pending further information.
450 Requested file action not taken.
    File unavailable (e.g., file busy).
550 Requested action not taken.
    File unavailable (e.g., file not found, no access).
451 Requested action aborted. Local error in processing.
551 Requested action aborted. Page type unknown.
452 Requested action not taken.
    Insufficient storage space in system.
```

```
552 Requested file action aborted.
    Exceeded storage allocation (for current directory or
    dataset).
553 Requested action not taken.
    File name not allowed.
```

Numeric Order List of Reply Codes

```
110 Restart marker reply.
    In this case, the text is exact and not left to the
    particular implementation; it must read:
         MA.K yyyy = mmmm
    Where yyyy is User-process data stream marker, and mmmm
    server's equivalent marker (note the spaces between markers
    and "=").
120 Service ready in nnn minutes.
125 Data connection already open; transfer starting.
150 File status okay; about to open data connection.
 200 Command okay.
202 Command not implemented, superfluous at this site.
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
    On how to use the server or the meaning of a particular
    non-standard command.  This reply is useful only to the
    human user.
215 NAME system type.
    Where NAME is an official system name from the list in the
    Assigned Numbers document.
220 Service ready for new user.
221 Service closing control connection.
    Logged out if appropriate.
225 Data connection open; no transfer in progress.
226 Closing data connection.
    Requested file action successful (for example, file
    transfer or file abort).
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
250 Requested file action okay, completed.
257 "PATHNAME" created.

331 User name okay, need password.
332 Need account for login.
350 Requested file action pending further information.

421 Service not available, closing control connection.
    This may be a reply to any command if the service knows it
    must shut down.
425 Can't open data connection.
426 Connection closed; transfer aborted.
450 Requested file action not taken.
    File unavailable (e.g., file busy).
451 Requested action aborted: local error in processing.
452 Requested action not taken.
    Insufficient storage space in system.
   500 Syntax error, command unrecognized.
    This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
502 Command not implemented.
```

```
503 Bad sequence of commands.
504 Command not implemented for that parameter.
530 Not logged in.
532 Need account for storing files.
550 Requested action not taken.
    File unavailable (e.g., file not found, no access).
551 Requested action aborted: page type unknown.
552 Requested file action aborted.
    Exceeded storage allocation (for current directory or
    dataset).
553 Requested action not taken.
    File name not allowed.
```

บรรณานุกรม

หนังสืออ้างอิง

[1] Jamie Jaworski (1996) : "*Java Developer's Guide*", Sam.net Developer 's Guide, Inc. 1996.

[2] Gary Cornell, Cay S. Horstmann (1996) : "*Core JAVA*", Sun Microsystem, Inc. 1996

[3] David Flanagan : "*JAVA in a Nutshell, Second Edition*", O'Reilly & Associates, Inc. United State of America 1997.

[4] ดร.วีระศักดิ์ ซึงถาวร : "*Fundamental of JAVA Programming Volume 1*", Sum Publishing Department, Sum System Company Limited. 1998.

[5] ดร.วีระศักดิ์ ซึงถาวร : "*Fundamental of JAVA Programming Volume 2*", Se-Education Public Company Limited. 2000.

[6] สุรศักดิ์ สงวนพงษ์ "สถาปัตยกรรมและโพรโตคอลทีซีพี/ไอพี". Se-Education Public Company Limited. 2000.

เว็บไซต์อ้างอิง

[1] http://www.employees.org/~satch/ssh/faq/ssh-faq.html

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

[3] http://www.java.sun.com/

[4] http://www.javasoft.com/

[5] http://cryp.to/publications/the_secure_shell/

[6] http://www.mindbright.se/mindterm/

[7] http://www.ietf.org/

[8] http://www.rfc-editor.org/

[9] http://www.w3.org/Protocols/rfc959/