การพัฒนาโปรแกรมจำลองเทอร์มินอลที่มีความปลอดภัย SSH (Secure Shell)

นายชาตฉกรรจ์ ไพบูลย์สิริกุล
นางสาวนภาลัย ศรีศักดิ์บางเตย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

การพัฒนาโปรแกรมจำลองเทอร์มินอลที่มีความปลอดภัย SSH (Secure Shell)

นายชาตฉกรรจ์ ไพบูลย์สิริกุล
นางสาวนภาลัย ศรีศักดิ์บางเตย

อาจารย์ที่ปรึกษา
อาจารย์ธนา หงษ์สุวรรณ
อาจารย์สุมณฑา หลิมศิริวงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาโปรแกรมจำลองเทอร์มินอลที่มีความปลอดภัย
SSH (Secure Shell)

ผู้จัดทำ

1. นายชาตฉกรรจ์ ไพบูลย์สิริกุล รหัสประจำตัว 38014118
2. นางสาวนภาลัย ศรีศักดิ์บางเตย รหัสประจำตัว 38014225

อาจารย์ที่ปรึกษา
(อาจารย์ธนา หงษ์สุวรรณ)

อาจารย์ที่ปรึกษา
(อาจารย์สุมณฑา หลิมศิริวงษ์)

SSH (Secure Shell)

ชาตฉกรรจ์ ไพบูลย์สิริกุล
นภาลัย ศรีศักดิ์บางเตย
ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
สุมณฑา หลิมศิริวงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์นี้จะนำเสนอการพัฒนาความปลอดภัยของโปรแกรมเทลเนทซึ่งใช้วิธีการพิสูจน์สิทธิ์และการเข้ารหัสข้อมูลที่มีประสิทธิภาพ

โปรแกรมที่ได้พัฒนาขึ้นดังกล่าวเป็นโปรแกรมด้านฟรอนท์เอ็นท์(front – end) ซึ่งทำงานภายใต้ระบบปฏิบัติการวินโดว์และทำการติดต่อกับเซิร์ฟเวอร์ที่มี Secure Shell ภายใต้ระบบปฏิบัติการยูนิกซ์ โดยแพคเก็ตของข้อมูลที่ถูกส่งในระบบเครือข่ายเป็นความลับเพราะทุกแพคเก็ตที่ส่งจะถูกเข้ารหัสด้วยคีย์ เพื่อให้แน่ใจว่าผู้ส่งและผู้รับนั้นเป็นผู้ที่ได้รับสิทธิอย่างแท้จริง

การพัฒนาโปรแกรมนี้กระทำภายใต้มาตรฐาน Secure Shell Protocol ซึ่งเวอร์ชันของเซิร์ฟเวอร์ที่เราเลือกใช้คือเวอร์ชัน 1.2.23 โดยเครื่องมือที่เราใช้พัฒนาโปรแกรมดังกล่าวคือ “Microsoft Studio Visual C++” ซึ่งโครงงานนี้จะเป็นประโยชน์สำหรับผู้สนใจทางด้านความปลอดภัยในการส่งข้อมูลผ่านระบบเครือข่าย

SSH(Secure Shell)

Chartchakan Paiboonsirikul
Napalai Srisakbangtoey
Thana Hongsuwan Advisor
Sumonta Limsiriwong Advisor
1998

ABSTRACT

This thesis represents the security development of telnet program which provides strong authentication and data encryption mechanism.

This program works as front-end application in Windows system which connect to secure shell server in Unix system. Data packets in network will be saved from eavesdropping because every packets are encrypted by a secret key. It makes sure that only authorized sender and receiver can understand that data.

Program was written based on SSH protocol standard. SSH version 1.2.23 was used as a server. We used " Microsoft Studio Visual C++ 5.0 " as developing tool. We hope that our project is useful for those who are interested in the security of data transferring area.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้จะไม่สามารถเสร็จสมบูรณ์ได้ถ้าไม่ได้รับคำแนะนำจากอาจารย์ที่ปรึกษาทั้ง 2 ท่าน คือ อาจารย์สุมณฑา หลิมศิริวงษ์ และอาจารย์ธนา หงษ์สุวรรณ ซึ่งทางคณะผู้จัดทำจักขอบพระคุณยิ่งสำหรับทุกสิ่งทุกอย่างที่ได้รับจากท่านทั้งสอง

นอกจากนี้ก็ต้องขอขอบพระคุณอาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ โดยเฉพาะคณาจารย์ทุกท่านในภาควิชาคอมพิวเตอร์ที่ทำให้คณะผู้จัดทำได้เป็นวิศวกรคอมพิวเตอร์อย่างเต็มภาคภูมิ

ขอขอบคุณภาควิชาวิศวกรรรมคอมพิวเตอร์ที่ช่วยอำนวยความสะดวกต่างๆ ทั้งเอื้อเฟื้อสถานที่และทางด้านอุปกรณ์ ให้คณะผู้จัดทำได้ทำการวิจัย

ขอขอบใจเพื่อน Gang 4D ทุกคนที่ช่วยเหลือคณะผู้จัดทำในการทำงานตลอดเวลา เป็นที่ปรึกษาเป็นกำลังใจอย่างดีเสมอมา

ท้ายที่สุดนี้ต้องขอขอบพระคุณ คุณบิดา มารดาที่ได้ให้กำเนิด คอยสั่งสอน และให้การศึกษา พร้อมทั้งสนับสนุนในกิจกรรมด้านต่างๆ นับเป็นพระคุณที่หาที่เปรียบมิได้ ทางคณะผู้จัดทำขอกราบขอพระคุณมา ณ ที่นี้ด้วย

ชาตฉกรรจ์ ไพบูลย์สิริกุล
นภาลัย ศรีศักดิ์บางเตย

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ที่มาของโครงงาน

ในปัจจุบันการส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ผ่านระบบเครือข่ายมีความจำเป็นมากขึ้น อีกทั้งยังมีวิธีการส่งข้อมูลและโปรแกรมที่ใช้ในการติดต่อระหว่างเครื่องคอมพิวเตอร์ผ่านระบบเครือข่ายหลายรูปแบบ เช่นโปรแกรมเทลเนท ซึ่งเป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่สามารถล็อคอินเข้ามาขอใช้บริการ จากเครื่องให้บริการที่อยู่ห่างไกลได้ (remote login) เมื่อผู้ใช้ทำการเชื่อมต่อกับเครื่องให้บริการ ผู้ใช้จะทำการส่งชื่อและรหัสผ่าน ไปให้เครื่องให้บริการเพื่อตรวจสอบ เมื่อเครื่องให้บริการตรวจสอบแล้วว่าถูกต้อง ผู้ใช้บริการก็จะสามารถเข้าไปขอให้บริการจากเครื่องให้บริการได้

ตามปกติ ข้อมูลที่ทำการส่งระหว่างเครื่องให้บริการและเครื่องใช้บริการที่ใช้โปรแกรมเทลเนทหรือโปรแกรมประยุกต์อื่นๆ เป็นข้อมูลที่ยังไม่ผ่านการเข้ารหัส ซึ่งหากมีผู้ไม่ประสงค์ดีต้องการทำการแก้ไขหรือขโมยข้อมูลก็สามารถทำได้ไม่อยากนัก

ดังนั้น ข้อมูลที่ทำการส่งผ่านระบบเครือข่ายจำเป็นต้องมีวิธีการที่จะทำให้ข้อมูลนั้นปลอดภัย ซึ่งโปรแกรมประยุกต์โปรแกรมหนึ่งที่สามารถตอบสนองความต้องการดังกล่าว คือ SSH (Secure Shell) โดย SSH นี้มีการทำการพิสูจน์สิทธิ์ และมีวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพก่อนที่จะทำการติดต่อและส่งข้อมูล ซึ่งแม้ว่าจะมีผู้ดักจับข้อมูลก็ไม่สามารถทราบได้ว่าเป็นข้อมูลอะไร เป็นเสมือนข้อมูลขยะ ทำให้ข้อมูลที่ส่งผ่านระบบเครือข่ายมีความปลอดภัยมากยิ่งขึ้น

1.2 จุดประสงค์ของโครงงาน

1. เพื่อเพิ่มความปลอดภัยให้กับข้อมูลที่ใช้ส่งผ่านระบบเครือข่ายอินเตอร์เน็ตในปัจจุบัน
2. เพื่อเพิ่มความปลอดภัยในการติดต่อแบบรีโมตล็อคอิน
3. ทำการสร้างโปรแกรมในส่วนของไคลเอนต์ที่ติดต่อกับโปรแกรม SSH ที่ใช้บนเซิร์ฟเวอร์ภายใต้ SSH โปรโตคอล
4. สามารถเขียนโปรแกรมประยุกต์ที่ทำงานบนระบบปฏิบัติการวินโดว์ ให้ใช้งานบนระบบเครือข่ายได้

1.3 ขอบเขตของโครงงาน

1. สร้างโปรแกรมที่ทำงานบนฝั่งไคลเอนต์ที่สามารล็อคอินเข้าสู่เซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ได้
2. โปรแกรมนี้สามารถทำงานภายใต้ระบบปฏิบัติการวินโดว์
3. ข้อมูลที่ใช้ในการติดต่อมีการรักษาความปลอดภัย

1.4 ประโยชน์ที่คาดว่าจะได้รับจากโครงงาน

1. โปรแกรมที่ได้เขียนขึ้นมีความปลอดภัยในการสื่อสารและสามารถทำงานได้จริง
2. เป็นแนวทางพัฒนาโปรแกรมประยุกต์บนระบบเครือข่าย
3. เป็นแนวทางพัฒนาระบบความปลอดภัยของโปรแกรมประยุกต์บนระบบเครือข่าย

บทที่ 2

การเข้ารหัสและการคำนวณ

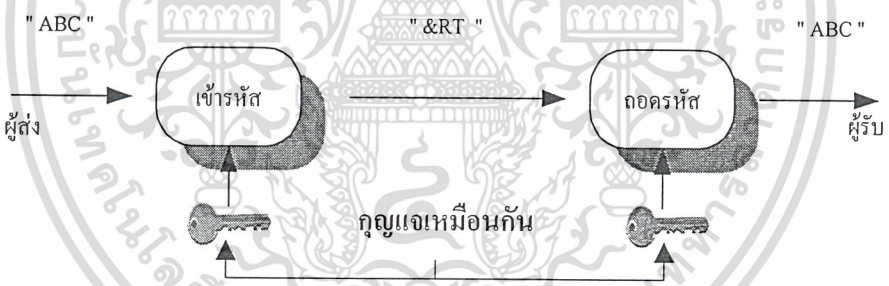
2.1 ระบบของเข้ารหัสข้อมูล (Cryptography System)

การเข้ารหัสข้อมูลเป็นการทำให้ข้อมูลที่ต้องการเป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูลปัจจุบัน โดยอาศัยหลักการของการเข้ารหัส (Encryption) และการการถอดรหัส (Decryption)

- การเข้ารหัสเป็นการเปลี่ยนรูปข้อมูล โดยผ่านรูปแบบและกระบวนการแปรรูปข้อมูล ทำให้ข้อมูลที่ส่งมีรูปแบบที่ไม่เหมือนเดิม เพื่อทำให้ข้อมูลเป็นความลับ
- การถอดรหัสเป็นการแปลงข้อมูลที่ได้เข้ารหัสให้กลับมาเป็นข้อมูลเดิม ซึ่งการเข้ารหัสและการถอดรหัสจะถูกควบคุมโดยกุญแจหรือที่เรียกว่า " Key "

2.1.1 ระบบการเข้ารหัสโดยใช้กุญแจเดียว (symmetric key)

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



รูป 2.1 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว

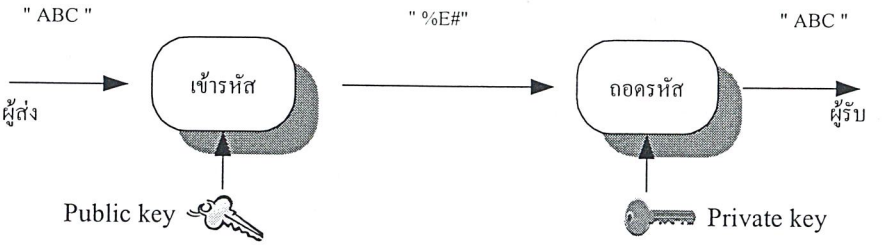
2.1.2 ระบบการเข้ารหัสแบบกุญแจสาธารณะ

การเข้ารหัสข้อมูลระบบนี้จะประกอบด้วยกุญแจ 2 อัน คือ

1 . กุญแจส่วนตัว (private key) เป็นกุญแจที่จะต้องเก็บเป็นความลับ

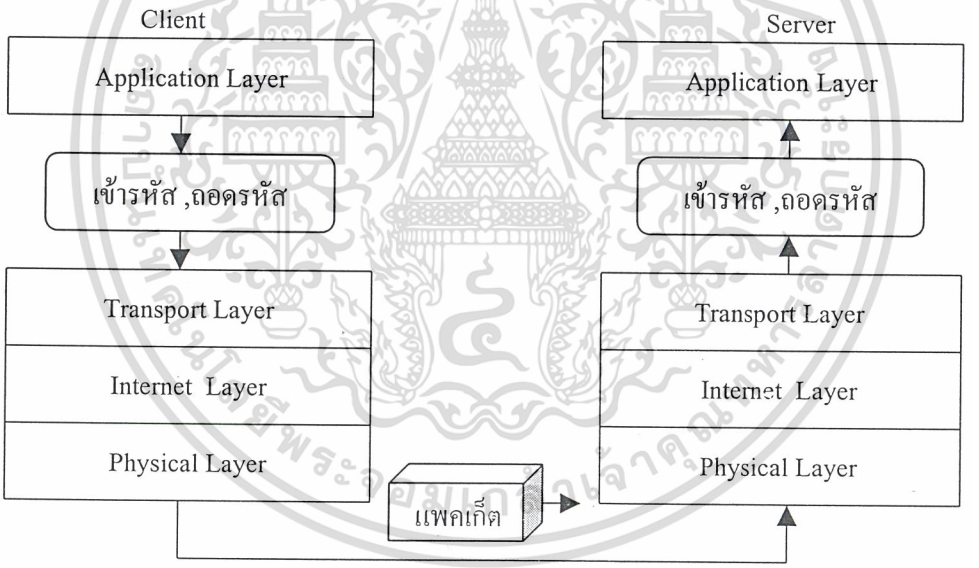
2. กุญแจสาธารณะ (public key) เป็นกุญแจที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งกุญแจทั้งสองนี้จะเป็นกุญแจที่ต่างกัน การมีวิธีในการเข้าและถอดรหัส ดังรูป



รูป 2.2 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ

สำหรับการเข้ารหัสข้อมูลที่ใช้ในโครงงานนี้ การเข้ารหัสข้อมูลจะทำการเข้ารหัสข้อมูลในช่วงระหว่างชั้นแอปพลิเคชั่นและชั้นทรานสปอร์ตในระบบ OSI โมเดลของโปรโตคอล TCP/IP ซึ่งวิธีการเข้ารหัสข้อมูลที่ใช้ คือ DES , 3DES (ระบบกุญแจเดียว) และ RSA (ระบบกุญแจสาธารณะ) ซึ่งรายละเอียดของการเข้ารหัสข้อมูลดังกล่าวจะมีดังต่อไปนี้



รูป 2.3 แสดงการเข้ารหัสข้อมูลบนระบบ OSI โมเดลของ TCP/IP

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

2.2 การเข้ารหัสแบบ DES (Data Encryption Standard)

2.2.1 ประวัติและที่มาของ DES

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

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

2.2.2 รายละเอียดของ DES

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

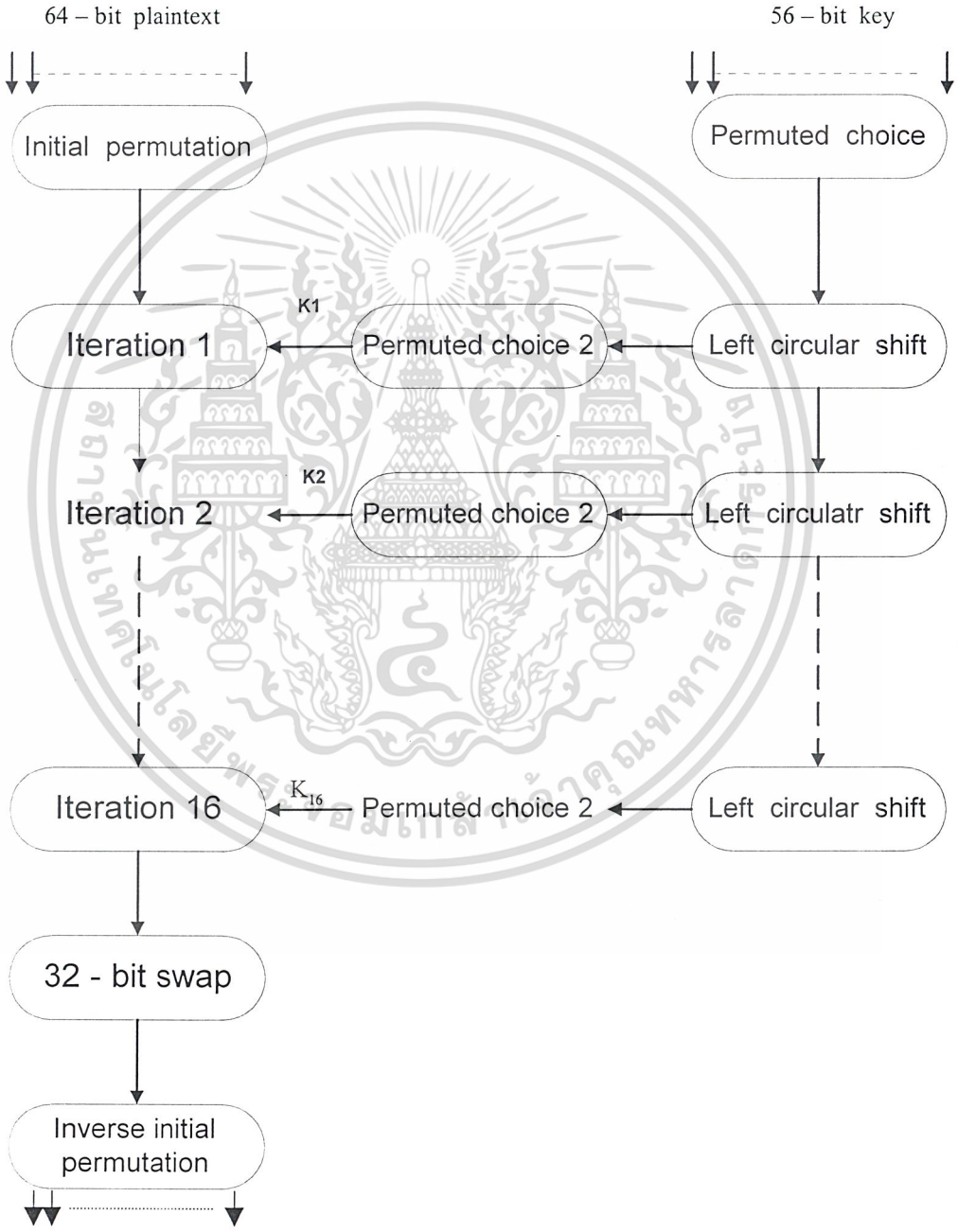
แม้ว่า DES ถูกนำมาใช้ตั้งแต่ช่วงทศวรรษที่ 70 (ค.ศ. 1960 – 1970) และได้รับการตอบรับอย่างดีจาก เหล่านักวิเคราะห์รหัส (Cryptanalysis)อย่างแพร่หลาย แต่ก็เป็นข้อถกเถียงกันเป็นอย่างมากถึงเรื่อง DES นั้นจะปลอดภัยได้หรือไม่และมีความปลอดภัยมากน้อยแค่ไหน แต่จนถึงปัจจุบันเราก็ยังไม่พบช่องโหว่ของ DES ตามเอกสารที่ตีพิมพ์เป็นสาธารณะ 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 สำหรับบทนี้จะไม่กล่าวถึง IDEA (เพราะมีความใกล้เคียงกับ DES)

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 จะทำการฟังก์ชั่นเดียวกัน 16 ครั้งซึ่งฟังก์ชั่นนี้ร่วมทั้งการทำ permutation และ substitution ซึ่งผลลัพธ์ที่ได้จากการทำทั้งหมด 16 ครั้งนี้จะได้ข้อมูลขนาด 64 บิตโดยใช้ทั้ง

ข้อมูลที่ไม่ได้ผ่านการเข้ารหัสและคีย์ในการทำ ซึ่งข้อมูลที่มีขนาด 64 บิตที่ได้นี้แบ่งเป็น 2 ด้านคือซ้ายและขวา ทั้งหมดจะถูก swap 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)



รูป 2.4 แสดงการขั้นตอนการทำงานของ DES

2.2.3 การทำ Initial Permutation

การทำ initial permutation และการทำ inverse initial permutation จะถูกอธิบายโดยใช้ตารางด้านล่างตามลำดับ ซึ่งจะเห็นได้ว่าฟังก์ชั่น permution ทั้ง 2 เป็นส่วนกลับซึ่งกันและกัน พิจารณาจาก 64 บิต:Mที่เข้ามา

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

2.2.4 รายละเอียดของการทำฟังก์ชันในแต่ละรอบ

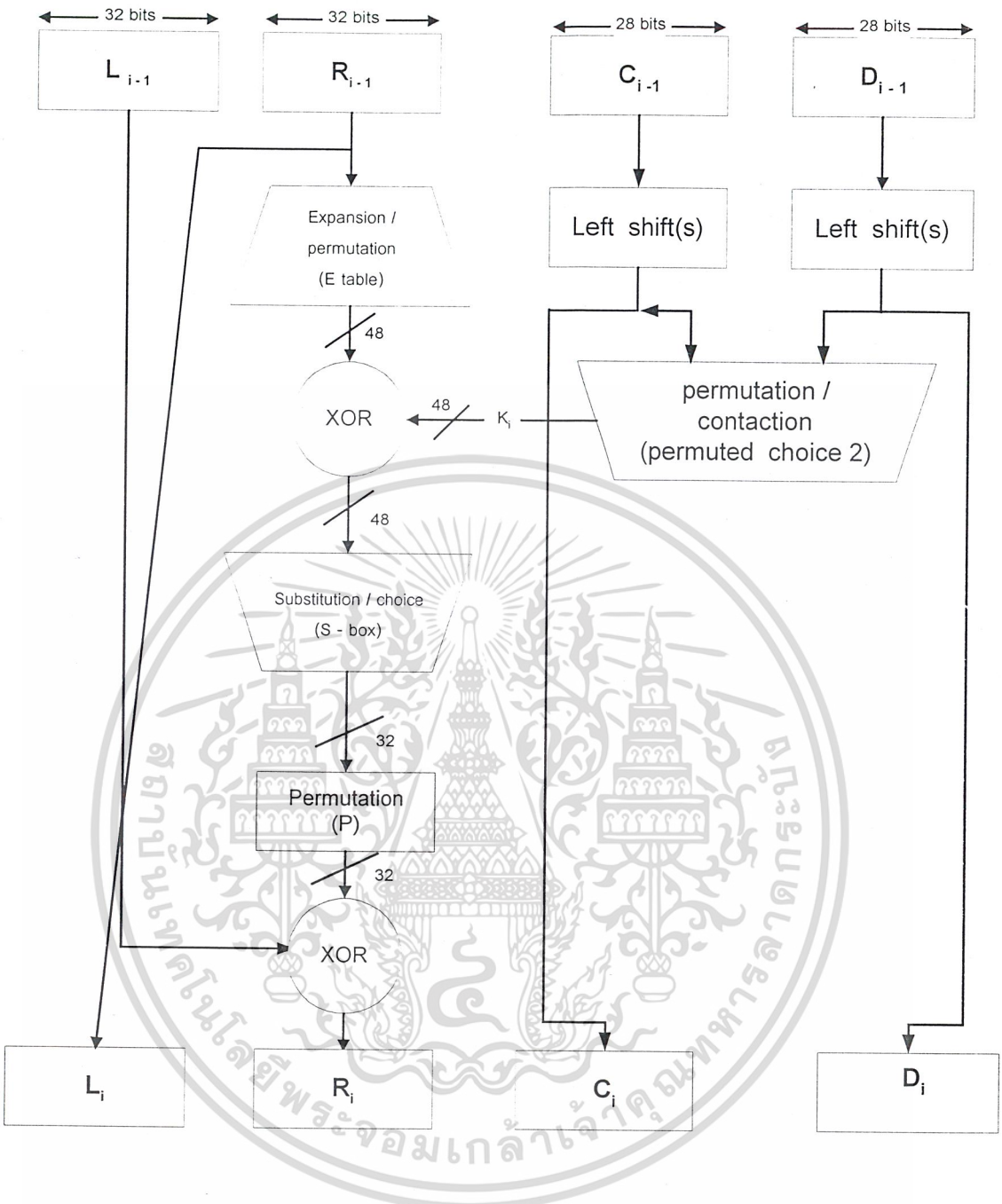
ข้อมูลที่เข้ามาที่มีขนาด 64 บิต โดยจะทำการแบ่งข้อมูลเป็น 2 ส่วนด้วยกันขนาดละ 32 บิตเท่ากัน (แบ่งเป็นซ้ายกับขวา) ซึ่งกระบวนการทำในแต่ละครั้งสามารถสรุปเป็นสูตรได้ดังนี้

$$L_I = R_{I-1}$$

$$R_I = L_{I-1} \oplus f(R_{I-1}, K_I)$$

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 $f(R_{I-1}, K_I)$ ซึ่งฟังก์ชั่น f จะแสดงดังรูป 2.5



รูป 2.5 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)

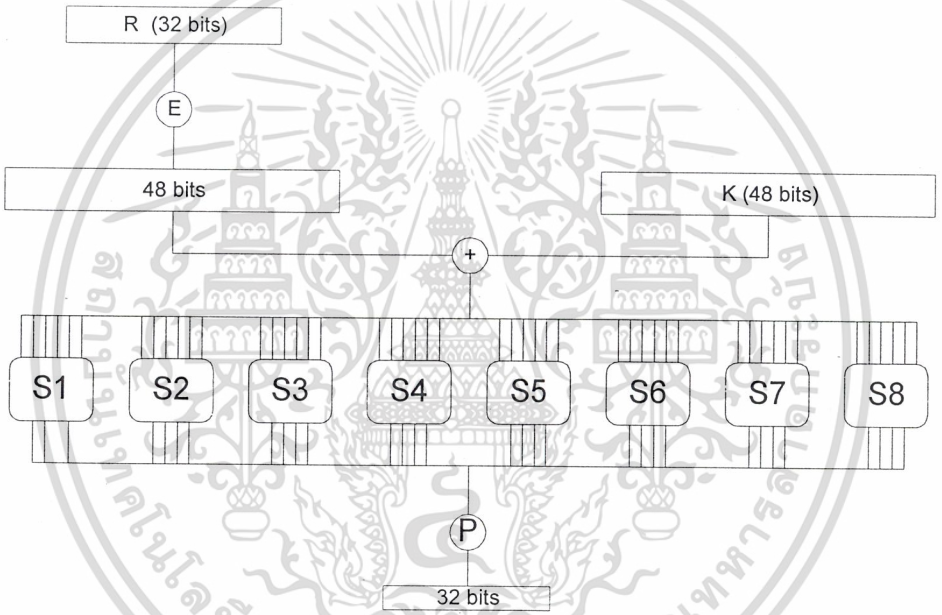
Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Form input bit	16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Form input bit	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

ตาราง 2.1 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 โดยจะนำผลที่ได้ผ่านไปยังฟังก์ชั่นที่เรียกว่า Substitution และ Permutation ที่สามารถผลิตผลลัพธ์ที่มีขนาด 32 บิต

โดย Substitution จะประกอบด้วยเซทของ 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 บิตตรงกลางมาเป็นตำแหน่งของคอลัมภ์ เช่น S_1 มีค่าเท่ากับ 011011 (ขนาด 6 บิต) เราจะนำบิตแรกและบิตสุดท้ายซึ่งก็คือ 0 และ 1 มาเป็นตำแหน่งของแถวจะได้แถวที่ 01 หรือถือแถว 1และ 4 บิตตรงกลางที่เหลือคือ 1101 จะได้ตำแหน่งของคอลัมภ์คือ คอลัมภ์ที่ 13 ดังนั้นค่าที่ตำแหน่งแถวที่ 1 และคอลัมภ์ที่ 13 ในตารางคือ 0101

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



รูป 2.6 แสดงการคำนวณ f(R,K)

Column Number

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

Column Number

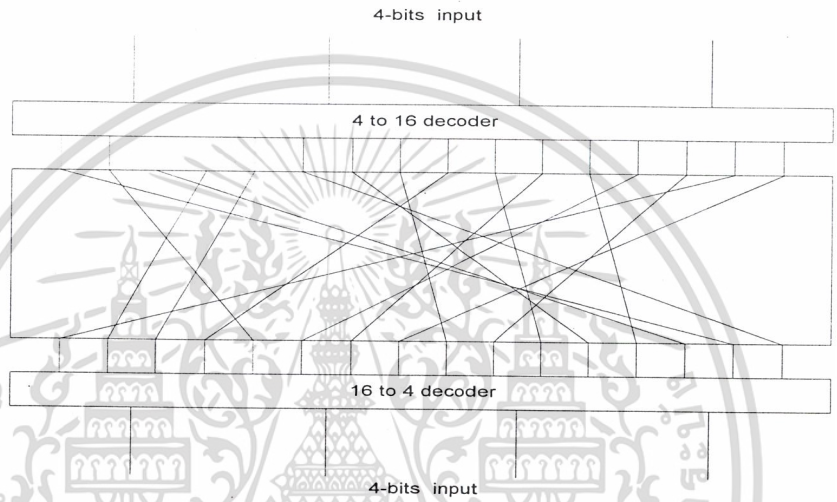
Box

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

ตาราง 2.2 แสดง S- box

2.2.5 การสร้างคีย์ (Key Generation)

ในรูปที่แสดงถึงการทำงานในแต่ละรอบของ DES เราจะเห็นว่าคีย์ที่ใช้มีขนาด 56 บิตซึ่งเป็นอินพุตในการทำ permutation โดยตาราง Permuted Choice One ดังรูป โดยเริ่มแรกจะทำการแบ่ง 56 บิตเป็น 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 $f(R_{I-1},K_I)$



รูป 2.7 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

ตาราง 2.3 แสดงการสร้างคีย์

2.2.6 การถอดรหัสข้อมูล DES

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

เราจะแสดงถึงผลลัพธ์ของขั้นตอนแรกในการกระบวนการถอดรหัส ซึ่งจะเท่ากับ 32 บิตที่ถูก Swap จากอินพุตของการทำทั้งหมด 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

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

$$D \oplus D = 0$$

$$E \oplus 0 = E$$

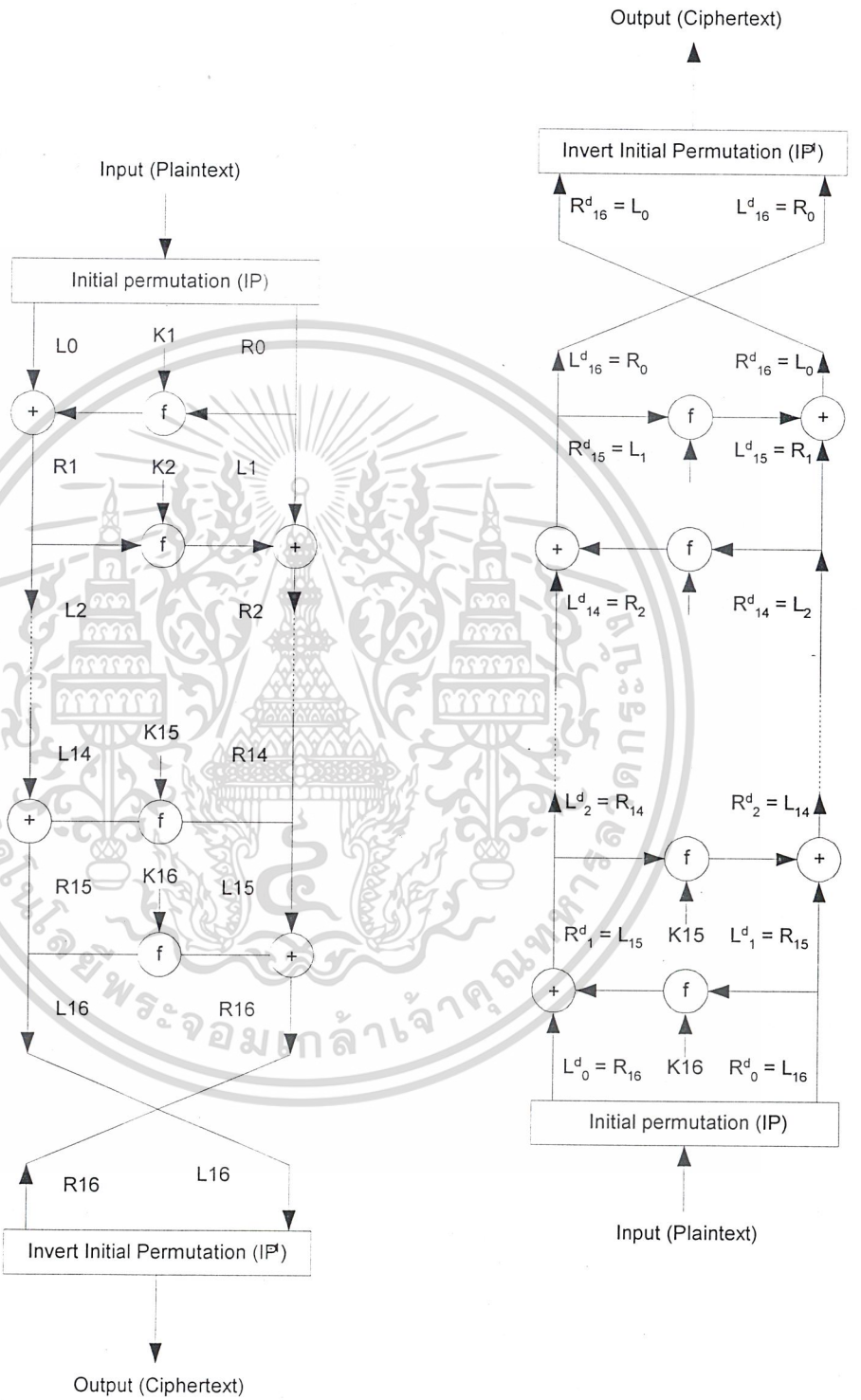
ดังนั้นเรามี Ld1 = R15 และ Rd1 = L15 จะได้เอาท์พุทของขั้นตอนแรกในการถอดรหัสคือ L15||R15 ซึ่งเราสามารถเขียนเป็นสมการในการถอดรหัสได้ดังนี้คือ

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

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

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

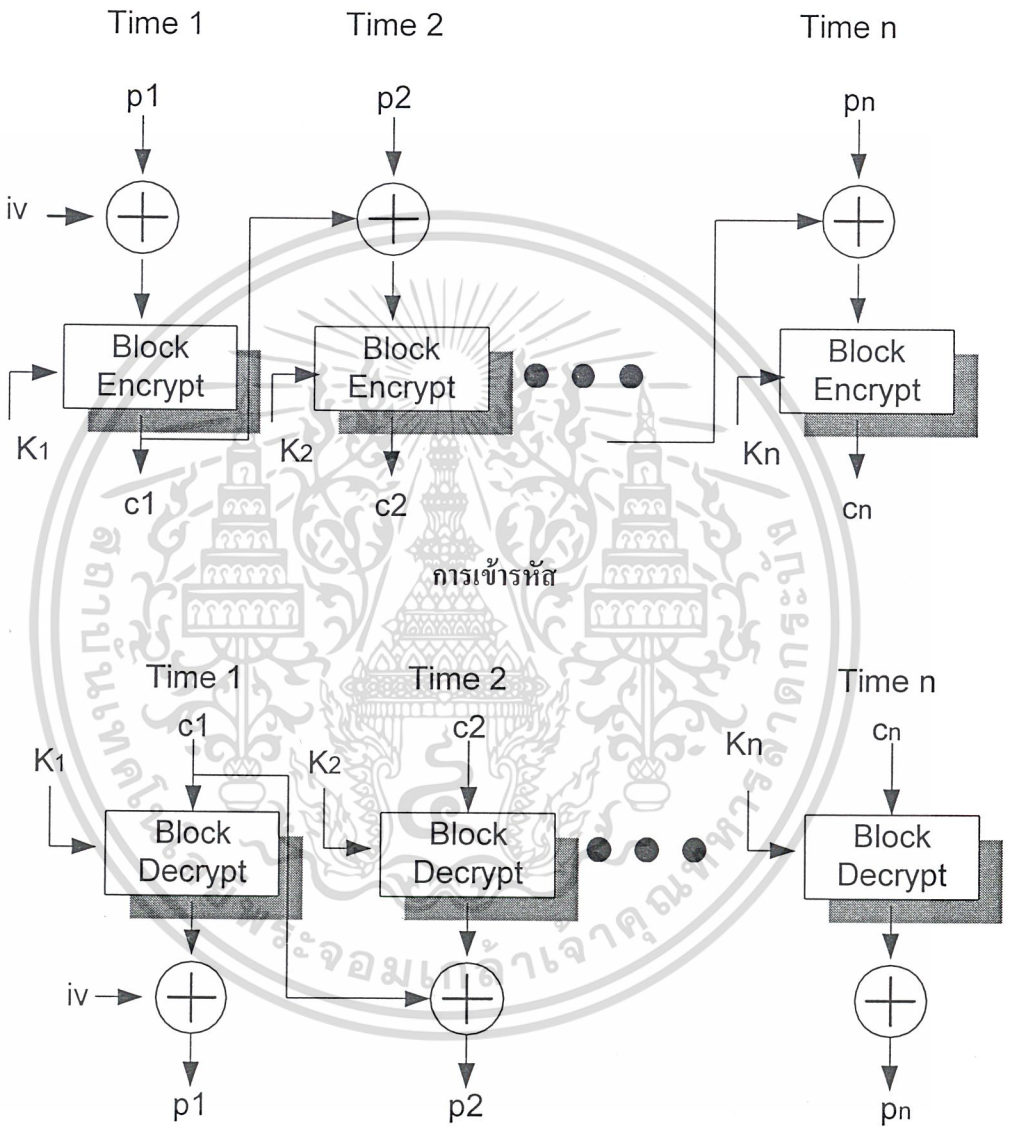
$$IP^{-1}(L0||R0) = IP^{-1}(IP(plaintext)) = plaintext$$



รูป 2.8 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES

2.2.7 โหมด CBC (Cipher Block Chaining)

เป็นวิธีการเข้ารหัสที่พัฒนามาจาก DES ช่วยทำให้ข้อมูลที่ส่งยิ่งมีความปลอดภัยมากยิ่งขึ้น โดยมีวิธีการคือ จะนำข้อมูลที่ผ่านการเข้ารหัสของข้อมูลตัวก่อนมา XOR กับข้อมูลที่ยังไม่ได้เข้ารหัสของตัวถัดไปก่อนจะทำการเข้ารหัสแบบ DES ตามปกติดังรูป



รูป 2.9 แสดงการเข้ารหัสและถอดรหัสของ DES CBC

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

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

สมการในการถอดรหัส

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

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

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

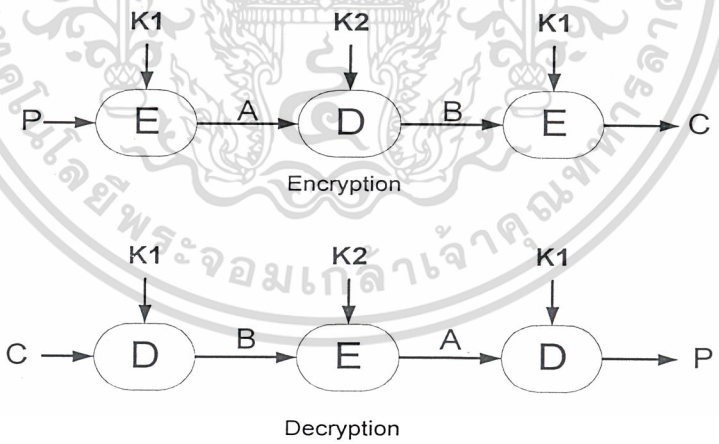
จะสังเกตเห็นได้ว่าบล็อคแรกของข้อมูลที่ผ่านการเข้ารหัส(Ciphertext) จะมีการนำ iv (Initialization Vector) มาทำการ XOR กับข้อมูลที่ไม่ได้เข้ารหัส(plaintext) ก่อนจะมีการเข้ารหัส DES ตามปกติ และในส่วนของการถอดรหัสข้อมูลก็จะใช้ iv ในการถอดรหัสข้อมูลเช่นเดียวกัน ดังนั้นจึงจำเป็นต้องมีข้อตกลงกันระหว่างผู้ส่งกับผู้รับก่อนว่าจะใช้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

2.3 การเข้ารหัสแบบ 3DES (Triple DES)

เป็นที่ทราบกันว่ายิ่งคีย์มีความยาวมากขึ้น การถอดรหัสยิ่งทำได้ยากยิ่งขึ้น นอกจากนี้เราก็สามารถเพิ่มสมรรถนะของการเข้ารหัสให้มากขึ้นได้โดย การเข้ารหัสหลายๆ ครั้ง

อย่างไรก็ตามการเข้ารหัสครั้งที่สองโดยใช้คีย์ที่ต่างจากครั้งแรกนั้น มิได้หมายความว่าข้อมูลจะมีความปลอดภัยเพิ่มขึ้นเป็น 2 เท่าถ้าการเข้ารหัสดังกล่าวเป็นการเข้ารหัสโดยเทคนิคทางคณิตศาสตร์

DES มิได้อยู่ในกลุ่มดังกล่าวแต่การที่จะทำให้ DES มีความปลอดภัยมาขึ้นเป็น 2 เท่านั้นจำเป็นที่เราจะต้องเข้ารหัสถึง 3 ครั้ง



รูป 2.10 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES

ขั้นตอนของการเข้ารหัสแบบ 3 ครั้งนั้นมีดังนี้

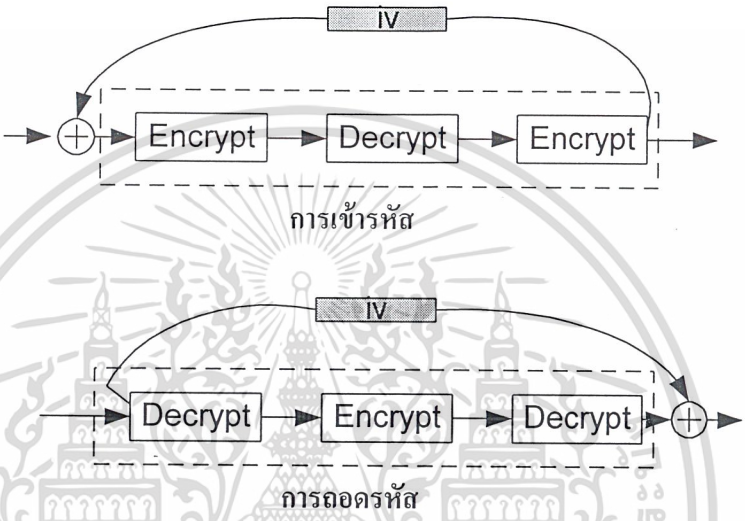
1. เข้ารหัสโดยใช้คีย์ตัวแรก
2. ถอดรหัสโดยคีย์ตัวที่สอง
3. เข้ารหัสโดยใช้คีย์ตัวที่สาม

(อาจใช้คีย์ตัวที่ 1 และ 3 เป็นตัวเดียวกันก็ได้ แต่ประสิทธิภาพของการเข้ารหัสก็จะลดลง)

2.3.1 Triple DES โหมด CBC (3DES_CBC Mode)

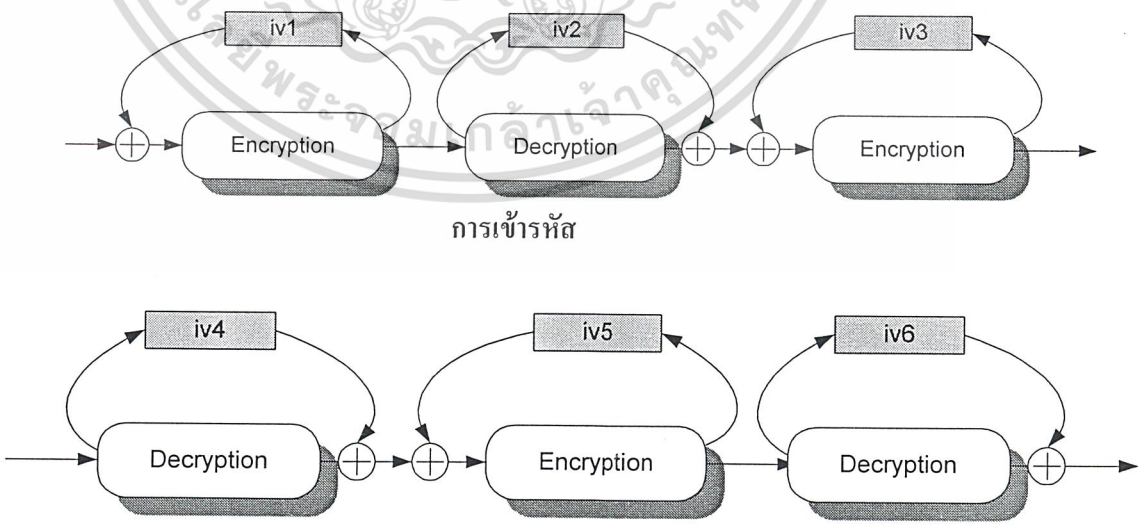
เนื่องจากการเข้ารหัสแบบ DES นั้นมีโหมดในการเข้ารหัสแบบ CBC ซึ่งการเข้ารหัส แบบ 3DES นี้ได้มีการประยุกต์มาจาก DES จึงได้มีการพัฒนา 3DES นี้ให้มีโหมด CBC ด้วย ซึ่งในการเข้ารหัสแบบ 3DES นี้ได้พัฒนาโหมดนี้ออกเป็น 2 แบบ

- **แบบ inner CBC** การเข้ารหัสแบบนี้จะมี iv เพียงตัวเดียวในการเข้ารหัสหรือถอดรหัสแบบ 3DES ในแต่ละครั้ง ดังรูป 2.11



รูป 2.11 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด inner CBC

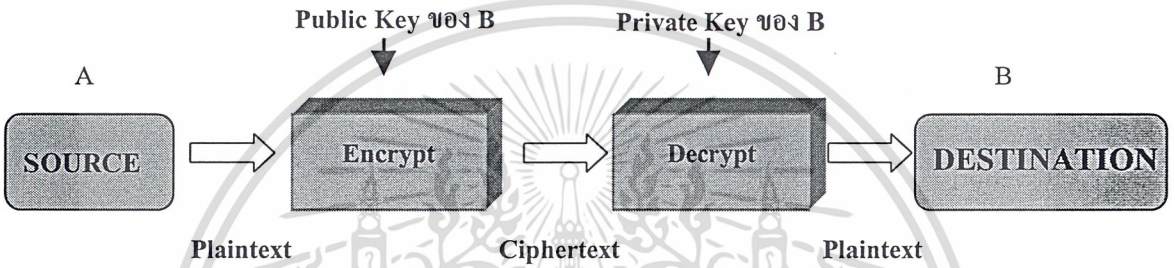
- **แบบ outer CBC** การเข้ารหัสแบบนี้แต่ละโมดูลในการเข้าหรือถอดรหัสจะมี iv เฉพาะของแต่ละโมดูล ดังรูป 2.12



รูป 2.12 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด outer CBC

2.4 การเข้ารหัสแบบ RSA

รหัส RSA ถูกคิดค้นขึ้นในปี 1977 โดยที่ชื่อ RSA ได้มาจากอักษรตัวแรกของนามสกุลของผู้ร่วมกันคิดค้นคือ Ron River, Adi Shamir และ Leonard Adleman เป็นวิธีการเข้ารหัสแบบกุญแจสาธารณะที่เรียกว่าพับบลิคคีย์ (Public–Key Crytosystem) เพื่อแก้ไขปัญหาการทำให้กุญแจเป็นความลับ(Secret–Key Cryptosystem) โดยสมาชิกแต่ละคนจะต้องมีกุญแจ 2 ชนิดคือ กุญแจส่วนตัวหรือไพรเวทคีย์ (Private Key) และกุญแจสาธารณะหรือพับบลิคคีย์(Public Key) โดยกุญแจส่วนตัวจะถูกเก็บไว้เป็นความลับ ส่วนกุญแจสาธารณะนั้นจะเปิดเผยให้ใครก็ได้ที่ต้องการส่งเอกสารให้แก่ตน การทำงานของรหัสลับมีหลักการว่าข้อมูลที่ถูกเข้ารหัสลับด้วยกุญแจสาธารณะของผู้ใด จะถูกถอดรหัสได้ด้วยกุญแจส่วนตัวของผู้นั้นเท่านั้น การทำงานของระบบพับบลิคคีย์สามารถอธิบายได้ดังต่อไปนี้



รูป 2.13แสดงขั้นตอนการทำ Public – Key Cryptosystem

2.4.1 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 ในคณิตศาสตร์ระบบโมดูโลฐาน (p-1)(q-1) นั่นคือ

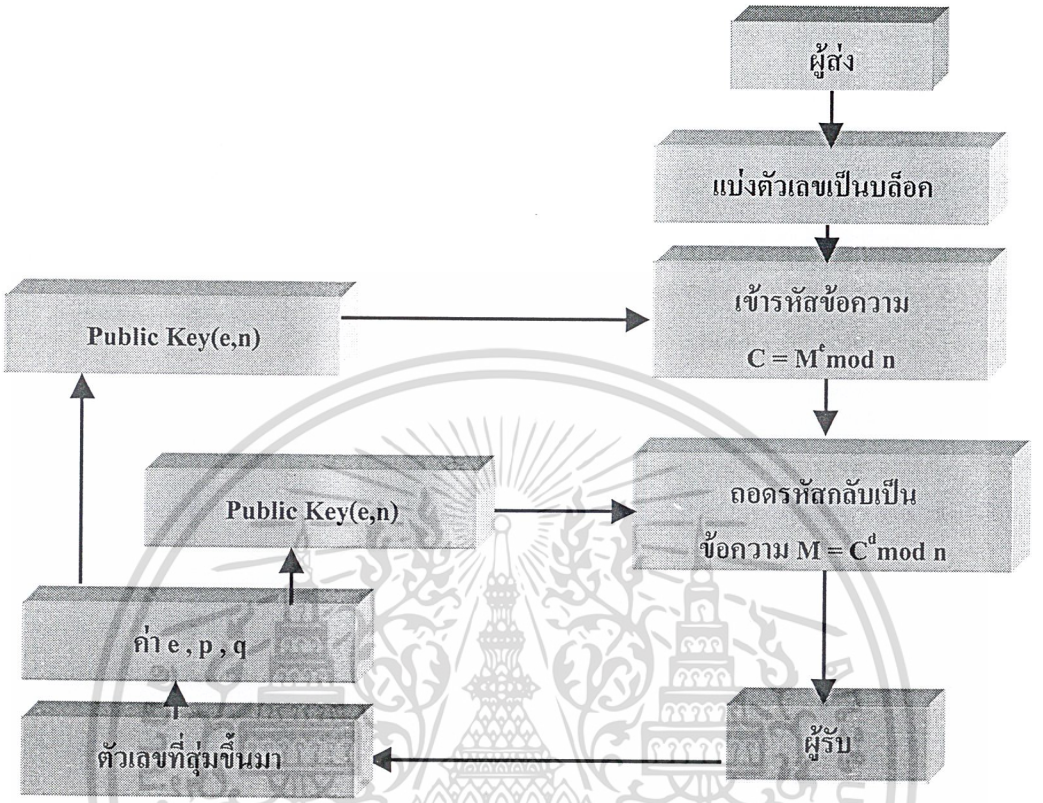
$$e*d \bmod (p-1)(q-1) = 1 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

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

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

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

และเพราะมีแต่บุญมีเท่านั้นรู้ค่า d บุญมีเท่านั้นจะถอดรหัสได้ คุณสมบัติสำคัญประการหนึ่งของ RSA คือจากสมการ (2) ในทางกลับกันถ้าเราเข้ารหัสด้วยไพรเวทคีย์ เราก็สามารถถอดรหัสด้วยพับบลิคคีย์ได้ด้วยเช่นกัน คุณสมบัติข้อนี้เองที่ทำให้ RSA มีประโยชน์มากเพราะในการใช้การเซ็นระบบดิจิตอลได้อีกด้วย



รูป 2.14 แสดงการเข้ารหัสแบบ RSA

ตัวอย่างขั้นตอนการเข้ารหัสแบบ RSA ซึ่งมีขั้นตอนในการหาKey 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 ก็จำเป็นต้องใช้การยกกำลังในระบบโมดูโล ซึ่งการยกกำลังนั้นสามารถทำได้โดยการใช้วงจรคูณระบบโมดูโลมาต่ออนุกรมกัน ดังนั้นความเร็วของทั้งการเข้าและถอดรหัสจึงขึ้นกับความเร็วของวงจรคูณระบบโมดูโลเป็นอย่างมาก นี่เองเป็นจุดอ่อนข้อหนึ่งของ RSA เมื่อเปรียบเทียบกับซีเคทคีย์ เช่น DES เพราะปัจจุบันการคูณในระบบโมดูโลยังค่อนข้างยุ่งยากเมื่อเปรียบเทียบกับการแทนค่าหรือสลับตำแหน่งใน 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 จะถูกใช้ในการเข้ารหัสข้อมูลที่มีขนาดใหญ่เพราะรวดเร็วกว่าในขณะที่RSA จะถูกใช้ในระบบสื่อสารที่ไม่ยาวนักแต่ต้องการความปลอดภัยสูงในบางครั้ง RSA ยังถูกใช้ร่วมกับ DES เพื่อเสริมจุดเด่นซึ่งกันและกัน เช่นตัวเอกสารจริงจะถูกเข้ารหัสด้วย DES โดยที่คีย์รหัส DES จะถูกเข้ารหัสด้วย RSA แล้วส่งไปด้วยกันหรือส่งไปก่อนแต่ในบางครั้ง DES อย่างเดียวก็พอแล้วหากการแลกเปลี่ยนคีย์สามารถทำได้อย่างปลอดภัยเพียงพอ หรือในกรณีที่ผู้ส่งและผู้รับเป็นบุคคลเดียวกัน เช่น ฮาร์ดดิสก์ในคอมพิวเตอร์ส่วนตัวหรือข้อมูลส่วนตัวในสมาร์ทการ์ด

2.5 การคำนวณแบบ MD5

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

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

ขั้นที่ 2 : เติมความยาว ใส่ขนาดของข้อความโดยจะมีค่าไม่เกิน 2^{64}

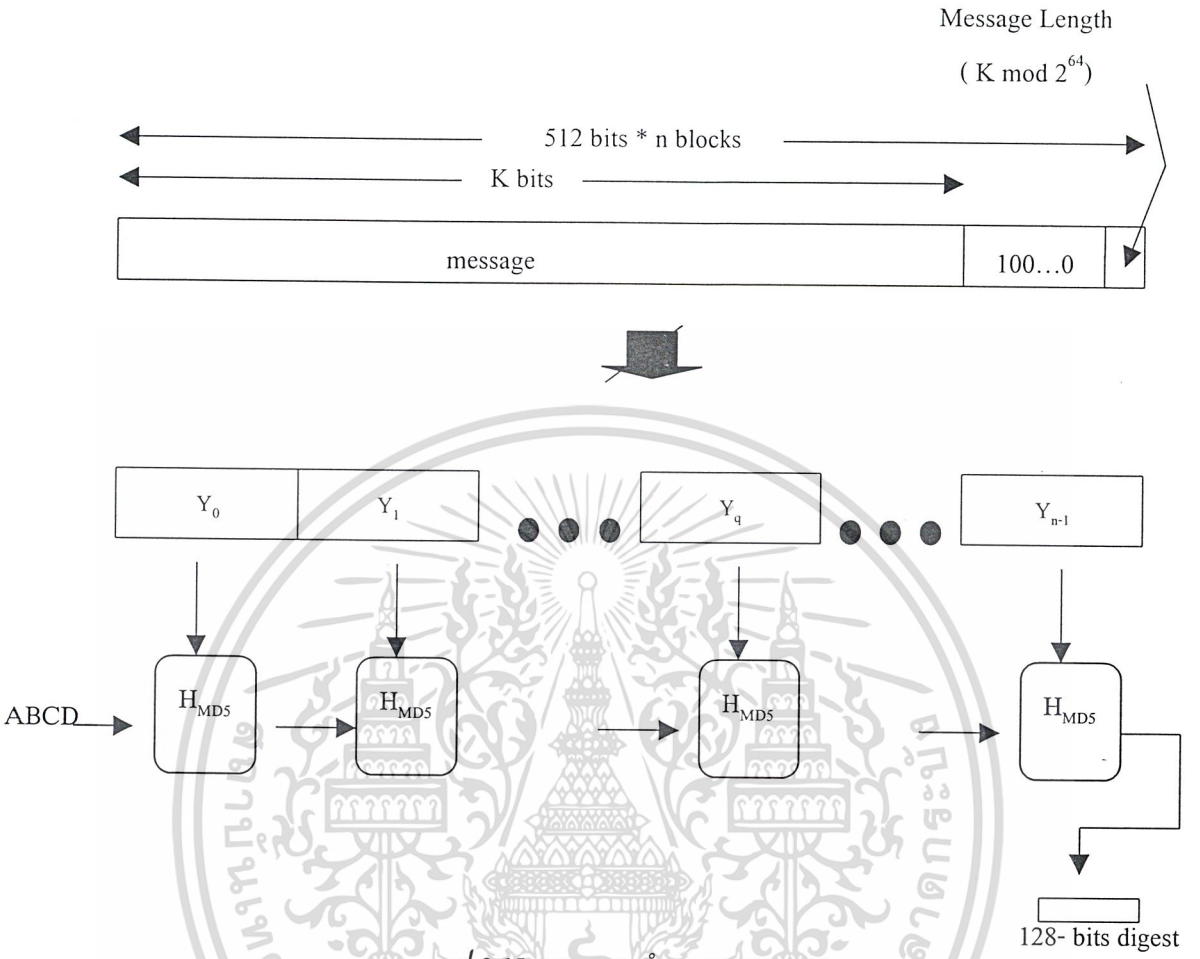
ขั้นที่ 3 : กำหนดค่าเริ่มของ MD บัฟเฟอร์ บัฟเฟอร์จะมีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการ hash โดยบัฟเฟอร์เริ่มต้นจะประกอบด้วย 32 บิตไบต์ 4 ตัว (A,B,C,D) ซึ่งมีค่าต่อไปนี้

A = 01234567

B = 89ABCDEF

C = FEDCBA98

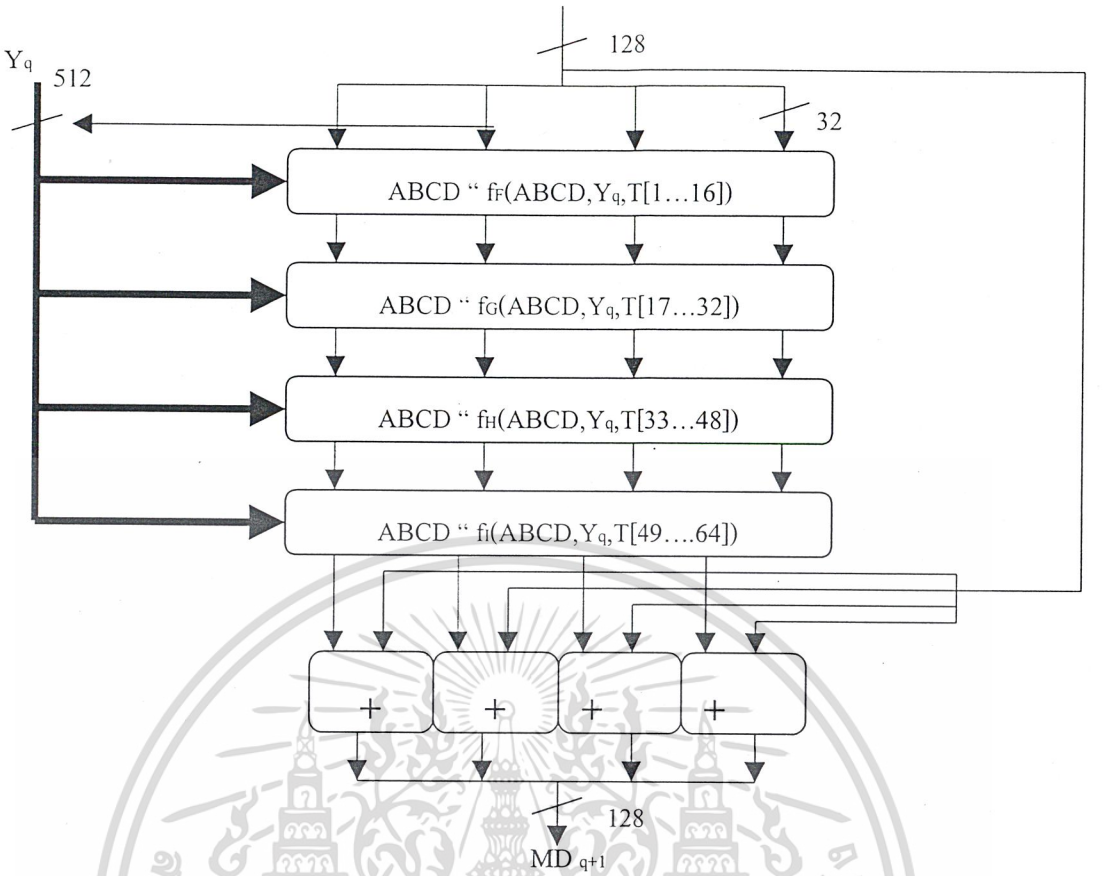
D = 76543210



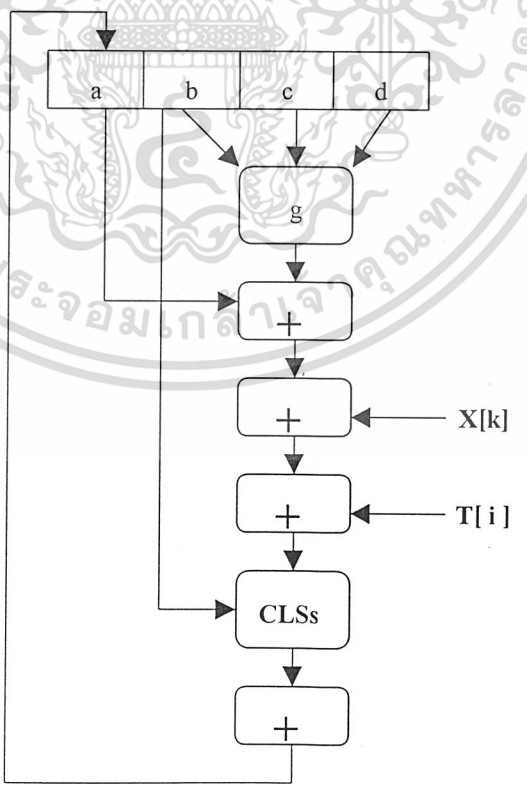
รูป 2.15 แสดงการคำนวณแบบ MD5

ขั้นที่ 4 : กระบวนการจัดการข้อความ 512 บิตบล็อก เป็นหัวใจของกระบวนการทั้งหมดประกอบด้วย 4รอบของกระบวนการที่คล้ายกันโดยแต่ละรอบจะใช้ฟังก์ชันเฉพาะของมัน 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



รูป 2.16 กระบวนการทำในหนึ่งบล็อกของ MD5



รูป 2.17 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 ($\overline{b}$. d)
FG	G(b, c ,d)	(b . d) v (c . $\overline{d}$)
FH	H(b, c ,d)	$b \oplus c \oplus d$
FI	I(b, c ,d)	$c \oplus (b . \overline{d})$

ขั้นที่ 5: Output

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

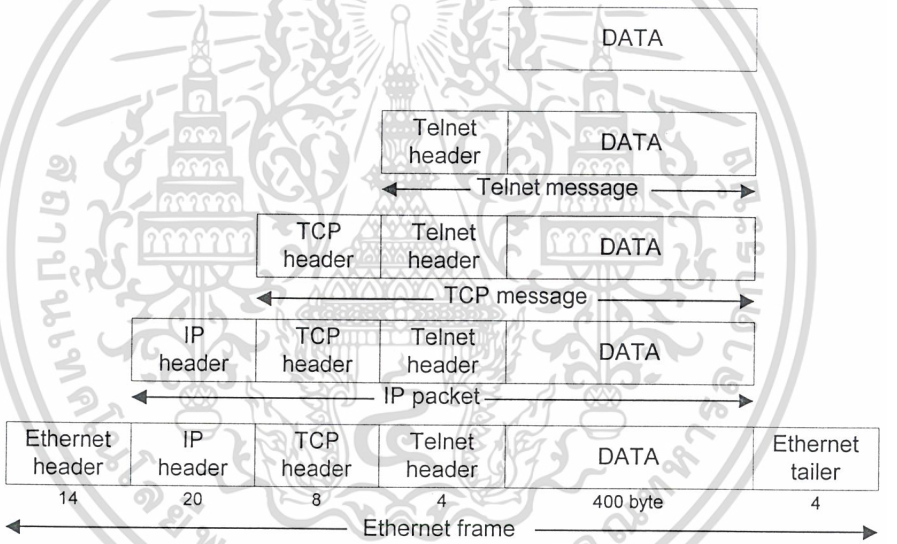
บทที่ 3

ภาพรวมของโปรแกรมเทลเนท

เทลเนท คือ โปรแกรมประยุกต์ที่ทำให้ผู้ใช้สามารถ ล็อคอินผ่านเทอร์มินอลใดๆ เพื่อขอใช้บริการจากโฮสต์ใด ๆ ที่เป็นเครื่องให้บริการที่อยู่ห่างไกลและใช้โปรโตคอล TCP / IP ในการติดต่อระหว่างเทอร์มินอลกับเครื่องให้บริการนั้นๆ นั่นเอง

3.1 เทลเนททำงานอย่างไร

เทลเนทเป็นโปรแกรมประยุกต์ที่ทำงานอยู่บนชั้นแอปพลิเคชั่นของโปรโตคอล TCP / IP ดังนั้นข้อมูลจากผู้ใช้ก่อนที่จะส่งออกไปในระบบเครือข่ายจะต้องมีการเพิ่มส่วนหัวในชั้น ต่าง ๆ ดังรูป



รูป 3.1 แสดงลักษณะของเฟรมบน TCP/IP

ส่วนรายละเอียดของ TCP message มีดังนี้

16-bit Source Port number			16-bit Destination Port number
32-bit Sequence number			
32-bit Acknowledgement number			
4-bit Header Lenght	6-bit Reserved	6-bit Flags	16-bit Window Size
16-bit TCP check sum			16-bit Urgent pointer
Option (if any) & Padding			
Data (Variable length, if any) (Telnet message)			

รูป 3.2 แสดงส่วนรายละเอียดของ TCP message

ส่วนรายละเอียดของ IP Packet มีดังนี้

4-bit Version	4-bit Header Length	8-bit Type of Service	16-bit Total Length	
16-bit Identification			3-bit Flags	13-bit Fragment Offset
8-bit Time to Live	8-bit Protocol		16-bit Header Checksum	
32-bit Source Address				
32-bit Destination Address				
Option (if any) & Padding				
Data (variable length) (TCP message)				

รูป 3.3 แสดงส่วนรายละเอียดของแพคเก็ต IP

ส่วนพอร์ตบริการของเทลเนทนั้น โดยทั่วไปแล้วจะอยู่ที่พอร์ตบริการเบอร์ 23 ดังนั้น การทำการติดต่อจะต้องกระทำกับพอร์ต 23 ที่เครื่องให้บริการเสมอที่มีการเรียกใช้โปรแกรมเทลเนท

3.2 เทลเนทมีหลักการทำงานเบื้องต้นอย่างไรบ้าง

หลังจากที่ผู้ใช้พิมพ์คำสั่ง "telnet < host id หรือ host name >" จะทำงานดังนี้

1. ที่เครื่องที่ผู้ใช้ (Client) จะทำการเริ่มโปรแกรมเทลเนท

2. เทลเนทจากเครื่องผู้ใช้จะพยายามติดต่อกับเครื่องให้บริการ ถ้าติดต่อได้เครื่องให้บริการจะตอบ (ACK) กลับ ถ้าไม่ได้ ก็จะแจ้งผิดพลาดแล้วออกจาก (ABORT) โปรแกรมไป
3. ถ้าติดต่อได้แล้ว เครื่องผู้ใช้ก็จะขอทำการติดต่อกับเครื่องให้บริการ โดยถ้าเครื่องให้บริการพร้อมให้บริการแล้วก็จะส่งสัญญาณ (ACK) ตอบกลับมา พร้อมทั้งส่ง Escape Character ที่จะใช้ในการทำเทอร์มินอลเสมือน (Virtual Terminal)
4. เครื่องผู้ใช้เมื่อได้รับสัญญาณ (ACK) ตอบกลับแล้วจะรอข้อมูลจากผู้ใช้ที่จะป้อนชื่อและรหัสผ่านแล้วทำการส่งข้อมูลเหล่านั้นไปให้เครื่องให้บริการ
5. เครื่องให้บริการจะตรวจสอบชื่อและรหัสผ่านและทำการตอบกลับว่าล็อคอิน (Login) สำเร็จหรือไม่โดยถ้าสำเร็จก็จะเข้าสู่กระบวนการของเชลล์ (Shell process) ต่อไป

3.3 Network Virtual Terminal (NVT)

เนื่องจากเทลเนทในทางปฏิบัติ เมื่อผู้ใช้ทำการล็อคอินกับเครื่องให้บริการใดๆก็ตาม เทลเนทจะทำการล็อคอินผ่าน NVT เพื่อให้ ผู้ใช้สามารถใช้เทอร์มินอลของตนในการล็อคอินเข้าไปใช้บริการและอุปกรณ์อื่นๆ ของเครื่องให้บริการได้

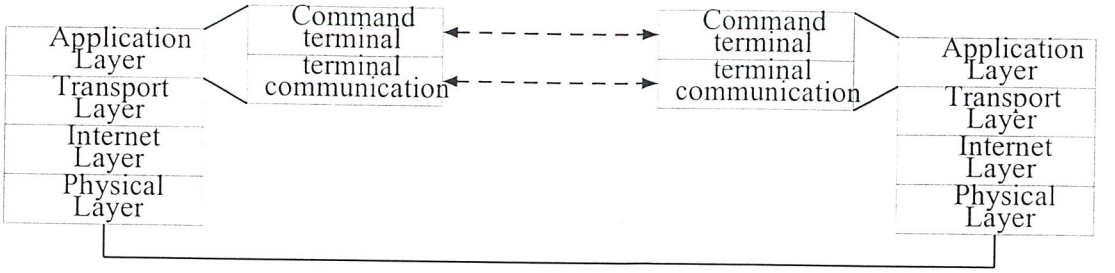
3.3.1 โครงสร้างและหลักการของ NVT

NVT เป็นหลักการที่ใช้ส่งข้อมูลระหว่างแอปพลิเคชั่นของเทอร์มินอล (terminal application) ที่อยู่ต่างเครื่องกันและต่างสิ่งแวดล้อมกัน ซึ่งได้แก่ ความแตกต่างในเรื่องของระบบปฏิบัติการ(OS) ตลอดจนถึงความแตกต่างในเรื่องโปรโตคอลที่ใช้ในระดับบน (Upper-Level) ของแอปพลิเคชั่นที่ทำงานอยู่ต่างเครื่องกันนั้น ซึ่งหลักการของ NVT นี้ จะทำให้แอปพลิเคชั่นที่รันอยู่บนเครื่องใช้บริการและแอปพลิเคชั่นที่รันอยู่บนเครื่องให้บริการซึ่งมีสิ่งแวดล้อมที่ต่างกันสามารถติดต่อกันได้ ซึ่งก็เปรียบเสมือนโปรแกรมเทลเนทที่รันอยู่บนเครื่องใช้บริการที่ต้องติดต่อกับ Telnetd ที่ทำงานอยู่บนเครื่องให้บริการนั่นเอง

หลักการของการทำ NVT มีอยู่ว่า เราจะมองเทอร์มินอลที่รันแอปพลิเคชั่น ทั้ง 2 ที่ต้องการติดต่อกันนั้นเป็น 2 เทอร์มินอล คือ physical Terminal กับ logical Terminal

Physical Terminal คือ เทอร์มินอลที่รันแอปพลิเคชั่นนั้นจริง ๆ ซึ่ง Physical Terminal ที่รันแอปพลิเคชั่น ทั้ง 2 อาจแตกต่างกันในหลาย ๆ เรื่องได้ดังที่ได้กล่าวมาแล้ว

Logical Terminal คือเทอร์มินอลในความคิดโดยเราจะกำหนดมาตรฐานต่าง ๆ ในการติดต่อให้เหมือนกันทั้ง 2 เครื่อง และเมื่อ Logical Terminal บนเครื่องทั้ง 2 เหมือนกันแล้วจึงทำให้ Logical Terminal ทั้ง 2 เครื่องคุยกันได้แล้ว logical Terminal ของแต่ละเครื่องก็จะแมปข้อมูล (MAP Data) ที่ติดต่อกันนั้น ให้อยู่ในรูปแบบที่ Physical Terminal ของเครื่องตนเองเข้าใจ ดังนั้นแอปพลิเคชั่น ที่รันอยู่บน Physical Terminal ที่ต่างกัน จึงสามารถติดต่อและแลกเปลี่ยนข้อมูลกันได้ อย่างไม่มีปัญหาเรื่องสภาพแวดล้อมของแต่ละเครื่องเข้ามาเกี่ยวข้องเลย และด้วยหลักการที่กล่าวทั้งหมดนี้ในทางปฏิบัติแล้วเราจะแบ่งแอปพลิเคชั่นเลเยอร์ (Application Layer) ออกเป็น 2 เลเยอร์ย่อยๆ (Sublayer) ดังรูป



รูป 3.4 แสดงโครงสร้างการทำ NVT

Command Terminal เป็นเลเยอร์ย่อย (sublayer) ที่รันแอปพลิเคชั่นนั้นๆ อยู่จริง โดยที่เลเยอร์ย่อยนี้ จะมีสภาพแวดล้อมต่าง ๆ ตามเครื่องที่รันอยู่ ซึ่งเลเยอร์ย่อยนี้ ก็เปรียบเสมือน Physical Terminal นั่นเอง

Terminal Communication เป็นเลเยอร์ย่อยที่ทำหน้าที่แปลงสิ่งแวดล้อมที่ไม่เหมือนกันทั้ง 2 เครื่องให้เป็นมาตรฐานเดียวกัน เพื่อให้แอปพลิเคชั่น ทั้ง 2 สามารถคุยกันได้และแปลงข้อมูลที่ติดต่อกันให้อยู่ในรูปแบบที่ physical Terminal ของตนเองเข้าใจ ซึ่งเลเยอร์ย่อยนี้ก็เปรียบเสมือน Logical Terminal นั่นเอง

3.3.2 การทำงานและออปชั่นของ NVT ในเทลเนท

การทำงานจะเริ่มจากเมื่อผู้ใช้รันเทลเนทที่เทอร์มินอลของตนเอง ที่สามารถติดต่อเข้าไปใน NVT ของเครื่องให้บริการที่ต้องการจะติดต่อด้วยได้อย่างที่ได้กล่าวมาแล้ว NVT ที่ติดต่อด้วยนั้น จะทำให้ผู้ใช้เหมือนได้ใช้เทอร์มินอลบนเครื่องเครื่องให้บริการนั้นจริง ๆ แต่อุปกรณ์ที่ผู้ใช้เห็นบนเครื่องให้บริการนั้นจริง ๆ แล้วเป็นเพียงการจำลองมาเท่านั้น

เมื่อให้เห็นภาพได้ชัดเจนยิ่งขึ้น คุณลองคิดถึงอุปกรณ์ในส่วนของเทอร์มินอล (Terminal Device) จริงๆ ดูว่าอะไรเป็นอุปกรณ์ที่เป็นอินพุต (เช่นแป้นพิมพ์) และอะไรเป็นอุปกรณ์ที่เป็นเอาท์พุท (เช่น จอภาพ, เครื่องพิมพ์) ดังนั้น NVT ก็จะจำลองอุปกรณ์ในส่วนของเทอร์มินอลจริงๆ เหล่านั้น ซึ่งก็มีแป้นพิมพ์เป็นอินพุตและเครื่องพิมพ์เป็นเอาท์พุต เครื่องพิมพ์จะพิมพ์ข้อความที่ได้รับมาจากเครื่องให้บริการบางข้อความที่เครื่องให้บริการต้องแสดงและแป้นพิมพ์ก็จะสร้างข้อความที่จะต้องส่งออกไปยังเครื่องให้บริการเสมือนกับว่าเทอร์มินอลที่ใช้อยู่นั้น เป็นเทอร์มินอลจริง ๆ บนเครื่องให้บริการนั่นเอง

และเนื่องจากในแต่ละเครื่องให้บริการมักจะมีผู้ใช้จำนวนมาก ดังนั้นการจะทำให้ NVT ตอบสนองกับสภาพแวดล้อมของเทอร์มินอลของแต่ละผู้ใช้จึงทำได้ยาก ซึ่งวิธีแก้ก็คือ ใช้ออปชั่นต่างๆตามข้อตกลงที่แต่ละเทอร์มินอลและผู้ใช้แต่ละคนต้องการ โดยจะทำการตกลงกันในตอนที่ทำการติดต่อกัน ซึ่งออปชั่นที่ NVT เตรียมไว้มีตัวอย่างดังนี้

- ออปชั่นเปลี่ยนลักษณะตัวอักษรของ NVT
- ออปชั่นที่เลือกว่าจะให้ NVT ทำการส่งตัวอักษรกลับมาให้ผู้ใช้เห็นบนจอภาพหรือไม่
- ออปชั่นเลือกโหมดว่าจะส่งข้อมูลเป็นบรรทัดทีเดียวหรือจะเลือกโหมดส่งข้อมูลที่ละตัวอักษร

และยังมีออปชั่นอื่น ๆ อีก การขอทำออปชั่นต่างๆ เหล่านี้ สามารถเลือกว่าจะใส่เพิ่มหรือถอนออกก็ได้ ซึ่งการตกลงเจรจาที่จะทำออปชั่นนั้น ทำได้โดยใช้ลำดับของข้อความ WILL, WON'T, DO และ DON'T นั่นเอง

"WILL X" เป็นข้อความที่ส่งจากด้านใดด้านหนึ่งโดยด้านที่ส่งนั้นได้บอกอีกด้านหนึ่งที่ติดต่อด้วยว่า ตนเองจะทำออปชั่น X ซึ่งถ้าอีกด้านหนึ่งยอมรับการทำออปชั่นนั้น ก็จะตอบกลับมาด้วย "DO X" แต่ถ้าอีกด้านหนึ่งไม่ยอมรับการทำออปชั่นนั้น ก็จะตอบกลับมาว่า "DON'T X"

"DO X" เป็นข้อความที่ส่งโดยด้านใดด้านหนึ่ง โดยที่มันต้องการให้อีกด้านหนึ่งที่ทำการติดต่อด้วย ทำออปชั่น X ให้มัน โดยถ้าอีกด้านหนึ่งนั้นตอบรับการทำออปชั่น X ก็จะตอบกลับมาด้วย "WILL X" แต่ถ้าอีกด้านหนึ่งไม่ยอมทำออปชั่น X ก็จะทำการตอบกลับไปว่า "WON'T X" นั่นเอง ซึ่งตัวอย่างของเทลเนท ออปชั่น (Telnet option) ได้แสดงเอาไว้ในตารางแล้ว

3.4 โหมดการส่งข้อมูลของเทลเนท

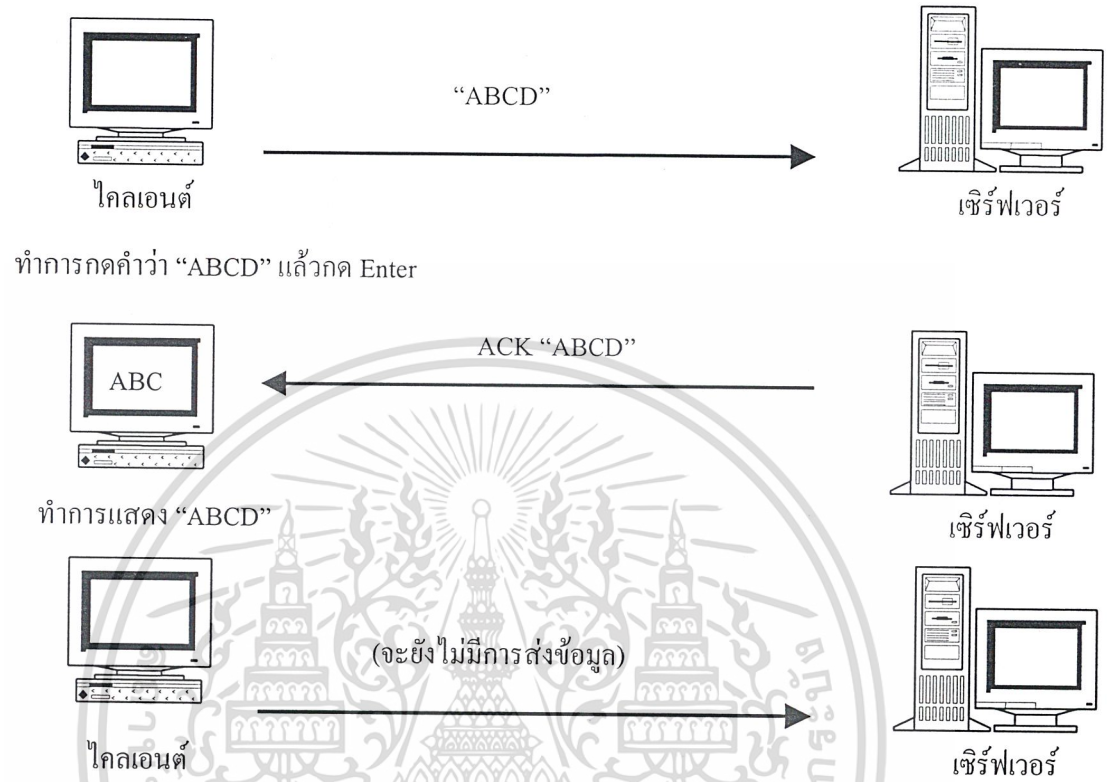
หลังจากที่ทำติดต่อระหว่างคำสั่งของเทลเนท (Telnet Command) บนเครื่องใช้บริการกับกระบวนการของ Telnetd บนเครื่องให้บริการแล้ว ก็จะมีการตกลงออปชั่นกัน หลังจากนั้นก็จะทำการส่งข้อมูล ซึ่งโหมดการส่งข้อมูลที่เป็นคำสั่งของเทลเนทไปยังกระบวนการของ Telnetd จะมีอยู่ 2 โหมดคือ โหมดการส่งข้อมูลทีละบรรทัด(โหมด line-at-a-time) และโหมดการส่งข้อมูลทีละตัวอักษร (โหมด character-at-a-time)

3.4.1 โหมดการส่งข้อมูลแบบทีละบรรทัด

การส่งข้อมูลในโหมดนี้ จะส่งข้อมูลทีละบรรทัด คือ ส่งเมื่อผู้ใช้มีการกดคีย์ขึ้นบรรทัดใหม่ "New line" หลังจากส่งข้อมูลจากเครื่องใช้บริการไปให้เครื่องให้บริการแล้ว เครื่องให้บริการก็จะตอบรับ (ACK) ข้อมูลที่ส่งมานั้นเท่านั้น จะไม่มีการส่งข้อความกลับมา (Echo) เพื่อแสดงผลบนหน้าจอเครื่องใช้บริการแต่อย่างใด เพราะการแสดงผลบนเครื่องใช้บริการจะถูกควบคุมด้วยตัวของเครื่องใช้บริการเอง เนื่องจากไม่สามารถรอการส่งข้อความกลับมาจากเครื่องให้บริการได้ เพราะเครื่องใช้บริการเองต้องรอการกดคีย์เพื่อขึ้นบรรทัดใหม่ "New line" จึงจะส่งข้อมูล ดังนั้นถ้ารอข้อความที่ส่งกลับมาจากเครื่องให้บริการแล้วผู้ใช้จะไม่เห็นตัวอักษรที่ตนพิมพ์เข้าไปจนกว่าผู้ใช้จะกดคีย์เพื่อขึ้นบรรทัดใหม่ "New line" ซึ่งถ้าเป็นอย่างนี้แล้วจะเป็นการไม่สะดวกที่ผู้ใช้จะแก้ข้อความที่พิมพ์ผิด เพราะผู้ใช้ยังไม่เห็นข้อความที่พิมพ์เข้าไป

ดังนั้นเพื่อไม่ให้เกิดเหตุการณ์ดังกล่าว การส่งข้อมูลในโหมดนี้จึงให้เครื่องใช้บริการเป็นเครื่องที่ควบคุมการแสดงผลเอง ดังนั้นการส่งข้อมูลในโหมดนี้จึงไม่มีปัญหาในการใช้คีย์พิเศษเช่น Backspace หรือ Delete เพราะอย่างที่ได้กล่าวไปแล้ว การจัดการแสดงผลจะทำโดยเทอร์มินอลของผู้ใช้ และผู้ใช้ยังเป็นผู้กำหนดว่าจะส่งข้อมูลไปให้เครื่องให้บริการเมื่อไหร่เองอีกด้วย

แต่การส่งข้อมูลแบบทีละบรรทัดก็มีปัญหาคือ การส่งข้อมูลในโหมดนี้ เมื่อส่งข้อมูลไปแล้วเราจะไม่สามารถกลับไปแก้ไขข้อมูลเดิมได้อีก จึงทำให้ไม่สามารถใช้โปรแกรมประเภท Word หรือ Editor ได้ในเทลเนทโหมดนี้ เพราะการส่งข้อมูลแบบนี้ไม่สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้นั่นเอง



รูป 3.5 แสดงโหมดการส่งข้อมูลแบบ line-at-a-time

3.4.2 โหมดการส่งข้อมูลแบบทีละตัวอักษร

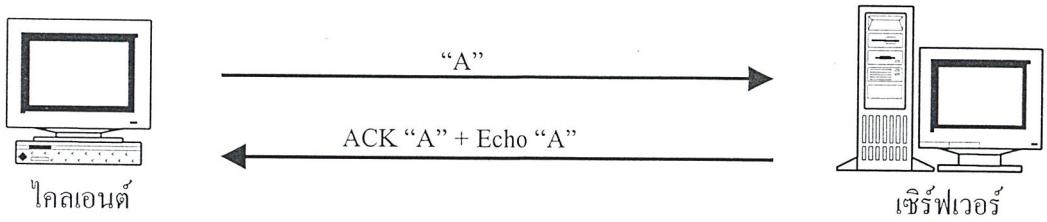
การส่งข้อมูลในโหมดนี้จะส่งข้อมูลที่ละตัวอักษร คือจะส่งกันที่ที่ผู้ใช้มีการกดคีย์ใดๆ และจะส่งข้อมูลที่ละตัวอักษร ซึ่งการส่งข้อมูลในโหมดนี้คำสั่งของเทลเนทที่รันอยู่บนเครื่องใช้บริการจะยังไม่แสดงผลข้อมูลที่ส่งไปยังเครื่องให้บริการบนจอภาพทันทีแต่จะรอให้เครื่องให้บริการตอบรับและส่งข้อความกลับมาให้จึงจะแสดงผลบนจอภาพ

ข้อดีของการส่งข้อมูลในโหมดนี้ ก็คือผู้ใช้สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้ง่ายกว่าการส่งข้อมูลใน โหมดการส่งแบบทีละบรรทัดเพราะข้อมูลที่ส่งไปแล้วนั้น มีขนาดเพียง 1 ตัวอักษร ดังนั้นเราจึงส่งข้อมูลไปแก้ไขข้อมูลที่ส่งไปแล้วเหล่านั้นได้ แล้วด้วยเหตุนี้จึงทำให้การส่งข้อมูลในโหมดนี้สามารถใช้งานโปรแกรมจำพวก Word หรือ Editor ได้ จึงทำให้การส่งข้อมูลในโหมดนี้ เป็นที่นิยมใช้กันในปัจจุบัน

แต่การส่งข้อมูลใน โหมดการส่งแบบทีละตัวอักษรนี้ก็มีปัญหาคือ การแสดงผลบนจอภาพของการส่งข้อมูลในโหมดนี้ นั้นจะต้องแสดงจากข้อความที่ถูกส่งกลับมาจากเครื่องให้บริการ ดังนั้นการแสดงผลบางอย่างจึงไม่สามารถทำได้ เช่น เมื่อผู้ใช้กด Key backspace แล้ว Host ส่ง ASCII ของ backspace กลับมาให้เครื่องใช้บริการของผู้ใช้ แล้วเครื่องใช้บริการของผู้ใช้ จะแสดงผลของ Key backspace ออกมา

เป็น character ของ ASCII ของ backspace ซึ่งไม่ได้เป็นไปตามที่ผู้ใช้ ต้องการซึ่งสิ่งที่ผู้ใช้ ต้องการคือ ให้ Move Cursor กลับไปหนึ่งตัวอักษรและลบ Character ตัวนั้นด้วย

ดังนั้น จึงต้องมีการทำเทอร์มินอลเสมือนที่จะกล่าวถึงในหัวข้อต่อไป



รูป 3.6 แสดงโหมดการส่งข้อมูลแบบ character-at-a-time

3.5 เทอร์มินอลเสมือน (Virtual Terminal)

เหตุผลในการทำเทอร์มินอลเสมือน ก็เพราะการส่งข้อมูลในโหมดการส่งข้อมูลทีละตัวอักษร การแสดงผลบนจอภาพของเครื่องใช้บริการของผู้ใช้จะถูกควบคุมโดยเครื่องให้บริการ แต่เนื่องจากเครื่องใช้บริการและเครื่องให้บริการอยู่ห่างไกลกัน การติดต่อก็ทำได้โดยผ่านเครือข่ายที่เชื่อมโยงเครื่องทั้งสอง ซึ่งการส่งข้อมูลผ่านเครือข่ายนี้เอง ทำให้เครื่องให้บริการไม่สามารถควบคุมการแสดงผลของเครื่องใช้บริการด้วยการส่งข้อความกลับมาเพียงอย่างเดียวได้ ดังนั่นจึงต้องมีการทำเทอร์มินอลเสมือน เช่น เมื่อผู้ใช้กดคีย์backspace แทนที่เครื่องให้บริการจะส่งตัวอักษรที่เป็น backspace กลับมาก็ส่งคำสั่งให้เครื่องใช้บริการทำการย้ายตำแหน่งเคอร์เซอร์ถอยหลังกลับ 1 ตัวอักษรแทน เป็นต้น ซึ่งคำสั่งที่เครื่องให้บริการใช้ทำเทอร์มินอลเสมือนเหล่านี้นั้น จะมีลักษณะเป็นชุดของคำสั่งต่างๆ โดยจะเริ่มต้นด้วย "escape" character ซึ่งจะตกลงกันเอาไว้ระหว่างเครื่องให้บริการและเครื่องใช้บริการตั้งแต่ตอนที่ติดต่อกันตอนแรก (ตอนที่ตกลงออปชั่นต่างๆ นั่นเอง) ซึ่งชุดของคำสั่งที่ใช้ในการทำเทอร์มินอลเสมือนเหล่านี้ ได้มีการกำหนดไว้เป็นมาตรฐานต่างๆ ไว้มากมายหลายมาตรฐาน แต่ที่นิยมใช้กันมากที่สุดได้แก่ มาตรฐานตระกูล VT ซึ่งในภาคผนวกได้มีตัวอย่าง Code ของ VT 100 ไว้ให้แล้ว

บทที่ 4

SSH(Secure Shell)

SSH เป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่ผู้ใช้สามารถล็อคอินเข้าใช้บริการจากเครื่องให้บริการที่อยู่ห่างไกลบนระบบเครือข่าย และสามารถทำการประมวลผล คำสั่งในเครื่องให้บริการที่อยู่ห่างไกลได้ (remote machine) พร้อมทั้งสามารถแลกเปลี่ยนข้อมูลบนระบบเครือข่ายอินเตอร์เน็ตได้อย่างปลอดภัย เนื่องจากมีวิธีการพิสูจน์ความเป็นเจ้าของที่มีประสิทธิภาพ (Strong Authentication) และระบบการติดต่อที่ปลอดภัย (Secure Communication) บนระบบเครือข่าย

4.1 ภาพรวมของโปรโตคอล SSH

โปรโตคอล SSH ประกอบด้วยโปรแกรมในส่วนของเครื่องที่ให้บริการ(เซิร์ฟเวอร์) และอีกส่วนหนึ่งคือส่วนของเครื่องใช้บริการ(ไคลเอนต์) โดยการติดต่อระหว่างเครื่องให้บริการและเครื่องใช้บริการจะติดต่อกันโดยใช้หมายเลข IP (IP Address) และ TCP/IP socket ซึ่งเป็นการติดต่อสองทิศทาง (bi-directional communication)

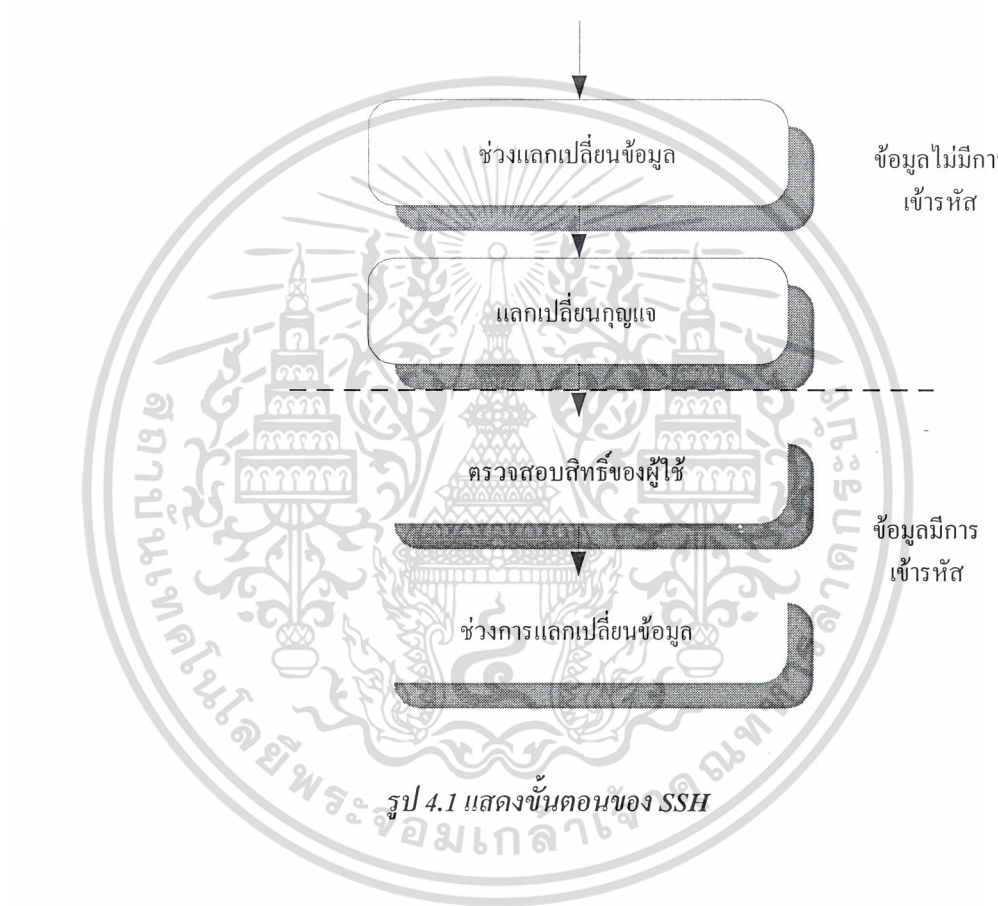
การติดต่อกันจะเริ่มการติดต่อจากเครื่องใช้บริการ โดยเครื่องให้บริการจะรอฟังจากพอร์ตที่กำหนดไว้เฉพาะ ว่ามีเครื่องที่จะใช้บริการติดต่อเข้ามายังพอร์ตนั้นหรือไม่ ซึ่งเครื่องให้บริการสามารถติดต่อกับเครื่องใช้บริการได้หลายเครื่องพร้อมๆ กัน

เมื่อเครื่องผู้ใช้บริการทำการติดต่อกับเครื่องผู้ให้บริการ หากเครื่องผู้ให้บริการยอมรับการติดต่อก็จะส่งข้อความเพื่อตรวจสอบเวอร์ชันของกันและกัน ที่เรียกว่า **Version Identification String** ของตนกลับไป เมื่อเครื่องใช้บริการได้รับก็จะทำการแปลข้อความของผู้ให้บริการและส่งข้อความในการตรวจสอบเวอร์ชันของตนเองกลับไปเช่นกัน ซึ่งหน้าที่ของ **Version Identification String** คือ เป็นข้อความที่ใช้ในการตรวจสอบการติดต่อว่าเวอร์ชันของโปรแกรมและโปรโตคอลนั้นรองรับกันหรือไม่

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

ต่อมาเครื่องผู้ใช้บริการจะทำการตรวจสอบสิทธิ์ของตน โดยทำการส่งหมายเลขเพื่อบ่งบอกวิธีในการพิสูจน์สิทธิ์ และทำการพิสูจน์สิทธิ์ตามแบบที่ได้ส่งไปเซิร์ฟเวอร์

เมื่อทำการตรวจสอบและพิสูจน์สิทธิ์แล้วว่าถูกต้อง เครื่องผู้ใช้บริการจะทำการส่งตัวเลขเพื่อทำการเตรียมตัวสำหรับการเรียกใช้เชลล์(Shell) และทำการเรียกใช้เชลล์เพื่อที่จะเข้าไปทำการติดต่อในส่วนอินเตอร์แอคชั่นโหมด(Interactive Session Mode) ซึ่งการทำงานในโหมดนี้เป็นโหมดในการแลกเปลี่ยนข้อมูลกันและกันระหว่างผู้ที่ใช้บริการกับเครื่องผู้ให้บริการ และจะสิ้นสุดการติดต่อเมื่อเครื่องผู้ให้บริการส่งคำสั่งเลิกบริการ(Exit Status) ของโปรแกรมไปยังเครื่องผู้ใช้บริการ



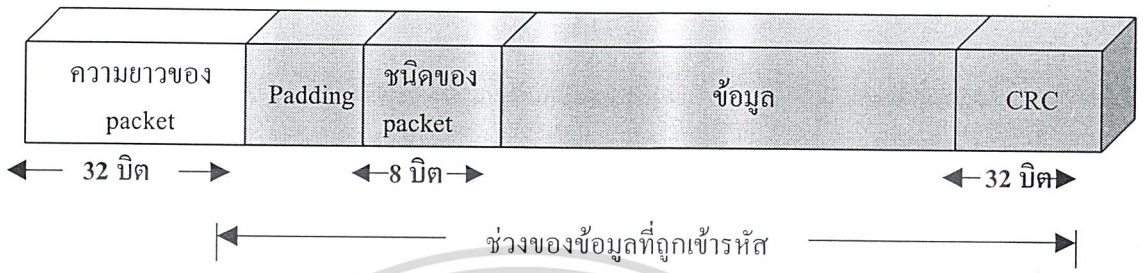
รูป 4.1 แสดงขั้นตอนของ SSH

4.2 ลักษณะและโปรโตคอลของข้อมูล

จากภาพรวมของโปรโตคอลที่ได้กล่าวมาข้างต้น ทำให้เราทราบวิธีการทำงาน ขั้นตอนการติดต่อและรูปแบบข้อความที่ใช้ติดต่อกัน แต่ก่อนที่เราจะศึกษาขั้นตอนการทำงานอย่างละเอียด เราจะมาทำความรู้จักกับคำที่ได้กล่าวมาข้างต้นก่อนว่าแต่ละตัวคืออะไร ทำหน้าที่อย่างไร และข้อความที่ใช้มีอะไรบ้าง

4.2.1 Binary Packet Protocol

ภายหลังจากการทำการตรวจสอบเวอร์ชันของกันและกัน ทั้งเครื่องให้บริการและเครื่องผู้ใช้บริการจะทำการส่งแพคเก็ตที่มีรูปแบบเฉพาะ ซึ่งเรียกว่า ***Binary Packet Protocol*** รูปแบบของแพคเก็ตเป็นดังนี้



รูป 4.2 แสดงฟิลด์ต่างๆ บน แพคเก็ตของระบบ Binary Packet Protocol

ความยาวของแพคเก็ต : มีขนาด 32 บิตซึ่งเป็นจำนวนเต็มบวก แบ่งออกเป็น 8 บิต/ไบต์ บิตแรกเป็นบิตที่มีค่าสูงสุด ความยาวของแพคเก็ต จะไม่นับรวมฟิลด์ของความยาวของแพคเก็ตและ padding ความยาวสูงสุดของแพคเก็ต คือ 26244 ไบต์

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

ชนิดของแพคเก็ต : จะมีขนาด 8 บิต ค่า 255 กันไว้สำหรับใช้ในอนาคต

ข้อมูลที่จะส่ง : ข้อมูล (มีลักษณะเป็นแบบเลขฐานสอง) ที่จะส่ง โดยขึ้นกับชนิดของแพคเก็ต จำนวนของข้อมูลมีขนาดเท่ากับ ความยาวของแพคเก็ตลบ 5

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

แพคเก็ตนี้ ยกเว้นฟิลด์ของความยาว จะถูกเข้ารหัสโดยอัลกอริทึมที่เลือกใช้ โดยความยาวของส่วนที่ถูกเข้ารหัส (padding + ชนิดของแพคเก็ต + ข้อมูล + Check) จะเป็นจำนวนเท่าของ 8 ไบต์

4.2.2 การเข้ารหัสแพ็คเก็ต (Packet Encryption)

แพคเก็ตของโปรโตคอลนี้รองรับการเข้ารหัสได้หลายชนิด โดยตั้งแต่เริ่มต้นการติดต่อ ตัวเซิร์ฟเวอร์จะส่งบิทมาร์ค(bitmask) ของวิธีการเข้ารหัสแบบต่างๆ ที่เครื่องให้บริการรองรับ ตัวเครื่องผู้ใช้บริการจะเลือกวิธีการในการเข้ารหัสข้อมูล และสร้างเซสชั่นคีย์ ขนาด 256 บิต และทำการส่งไปให้เครื่องผู้ให้บริการ

วิธีการเข้ารหัสที่รองรับและโค้ดของวิธีการเข้ารหัสแต่ละแบบคือ

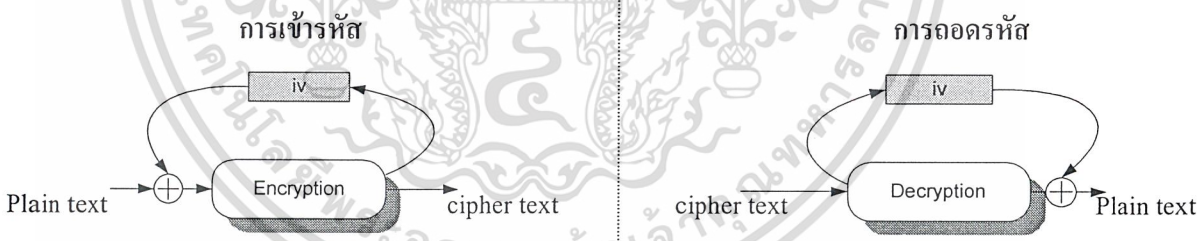
ชนิดแพคเก็ต	หมายเลขแทนชนิดของการเข้ารหัส	ชนิดของการเข้ารหัส
SSH_CIPHER_NONE	0	No encryption
SSH_CIPHER_IDEA	1	IDEA in CFB mode
SSH_ CIPHER_DES	2	DES in CBC mode
SSH_ CIPHER_3DES	3	Triple – DES in CBC mode
SSH_ CIPHER_TSS	4	An experimental stream cipher
SSH_ CIPHER_RC4	5	RC4

ตาราง 2.1 แสดงวิธีการเข้ารหัสที่ SSH รองรับ

สำหรับในบทนี้จะอธิบายเฉพาะการเข้ารหัสที่ใช้ในโครงงานนี้เท่านั้น

SSH_CIPHER_DES

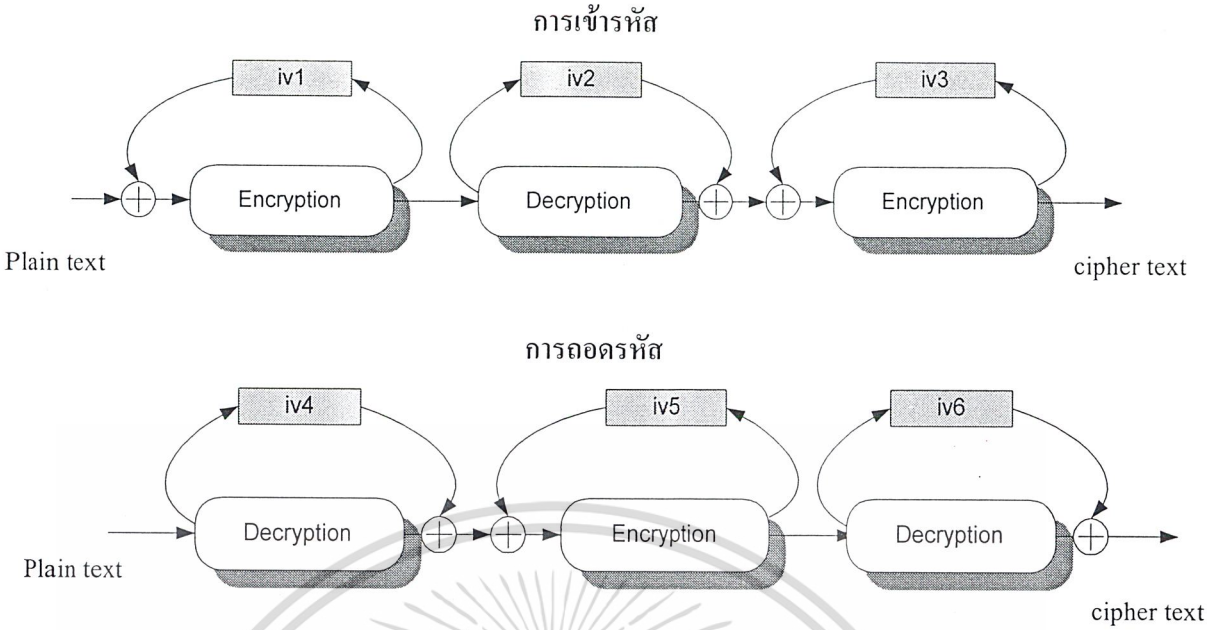
กุญแจในการเข้ารหัสนำมาจาก 8 บิตแรกของเซสชั่นคีย์ และ บิตที่มีค่าน้อยสุดของแต่ละไบต์จะถูกตัดทิ้งไป กุญแจที่ได้จะมีขนาด 56 บิต การเข้ารหัสแบบ DES ที่ใช้นี้จะเป็นโหมด CBC ซึ่ง iv (initialization vector) ในการเริ่มต้นถูกกำหนดให้เป็น 0 ทั้งหมด (ซึ่งได้อธิบายละเอียดในบทการเข้ารหัสข้อมูล)



รูป 4.3 แสดงการเข้ารหัสและถอดรหัสแบบ DES โหมด CBC

SSH_CIPHER_3DES

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



รูป 4.4 แสดงวิธีการเข้ารหัสและถอดรหัสแบบ 3DES โหมด CBC

สำหรับการเข้ารหัส ข้อมูลจะถูกเข้ารหัสด้วย cipher แรกและถอดรหัสด้วย cipher ที่ 2 และครั้งสุดท้ายจะถูกเข้ารหัสอีครั้งด้วย cipher 3 ส่วนการถอดรหัส ข้อมูลจะถูกถอดรหัสด้วย cipher แรกและเข้ารหัสด้วย cipher ที่ 2 และครั้งสุดท้ายจะถูกถอดรหัสอีครั้งด้วย cipher 3 โดยกุญแจที่ใช้สำหรับ cipher แรกจะถูกนำมาจาก 8 ไบต์แรกของเซสชั่นคีย์ และกุญแจสำหรับ cipher ที่ 2 มาจาก 8 ไบต์ถัดไปของเซสชั่นคีย์ และกุญแจสำหรับ cipher ที่ 3 จะเป็น 8 ไบต์ถัดไปของเซสชั่นคีย์เช่นกัน (ค่า iv จะไม่เกี่ยวข้องกัน)

4.2.3 ชนิดของข้อมูลที่ถูกเข้ารหัส (Data Type Encoding)

ฟิลด์ข้อมูลของแต่ละแพคเก็ตจะเก็บข้อมูลที่เข้ารหัสตามวิธีที่ได้อธิบายไป ซึ่งจะประกอบรายการของข้อมูลหลายๆ ตัว แต่ละรายการจะมีรหัสในแต่ละแบบและถูกนำมาใช้โดยการนำแต่ละรายการมาต่อกัน

ข้อมูลแต่ละชนิดจะถูกเก็บตามนี้คือ

- **8 bit byte** เป็นข้อมูลที่บรรจุในไบต์ๆ เดี่ยว (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

4.2.4 หมายเลข TCP/IP พอร์ต (TCP/IP Port Number & Other option)

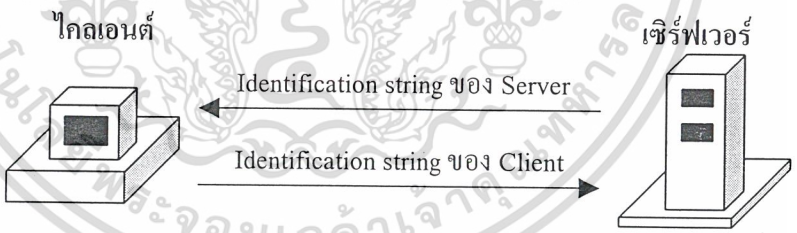
ตัวผู้ให้บริการจะรอติดต่อบนโปรโตคอล TCP/IP พอร์ตที่ 22 ตัวผู้ใช้บริการจะใช้พอร์ตใดของตนก็ได้ในการติดต่อเข้ากับเครื่องผู้ให้บริการ แต่ถ้าตัวเครื่องใช้บริการต้องการจะใช้การติดต่อแบบ .rhosts หรือ /etc/hosts.equiv จะต้องทำการติดต่อจากพอร์ตที่ถูกสงวนไว้ (เบอร์พอร์ตที่ต่ำกว่าเบอร์ 1024)

4.3 รายละเอียดขั้นตอนการติดต่อ

4.3.1 ช่วงตรวจสอบเวอร์ชัน (Protocol Version Identification)

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

ส่วนผู้ใช้บริการเมื่อได้รับข้อความมาก็จะทำการแปล ถ้าไม่รองรับเวอร์ชันของโปรโตคอลก็จะทำการปิดการติดต่อ แต่ถ้ารองรับเครื่องผู้ใช้บริการก็จะส่งข้อความที่เป็นข้อมูลของตัวเองลักษณะเดียวกับที่เครื่องผู้ให้บริการส่งมากลับไป เมื่อเครื่องผู้ให้บริการได้รับก็จะตีความหมาย ถ้าเป็นเวอร์ชันของโปรโตคอลที่รองรับ ถ้าตรงกันไหมก็จะส่งข้อความแรกกลับไป ซึ่งข้อความจะอยู่ในรูปของ binary packet protocol



รูป 4.5 แสดงการตรวจสอบเวอร์ชัน Identification

4.3.2 ช่วยการแลกเปลี่ยนกุญแจ (Key Exchange)

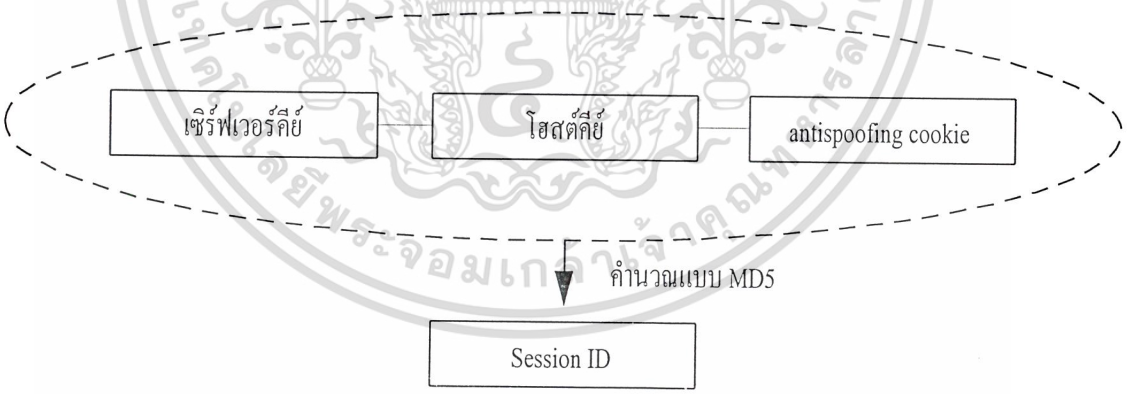
หลังจากที่มีการตรวจสอบเวอร์ชันว่ารองรับกันได้แล้ว ก็จะเข้าสู่การแลกเปลี่ยนกุญแจ เนื่องจากการเข้ารหัสของข้อมูลต่างๆ ที่ใช้ติดต่อกันทั้งสองฝ่ายจะต้องมีกุญแจเดียวกันในการเข้ารหัสและถอดรหัสของข้อมูลนั้นๆ ซึ่งจะต้องมีวิธีการทำให้ทั้งคู่มีกุญแจเดียวกัน โดยที่ผู้อื่นไม่ทราบว่ากุญแจนี้ คืออะไร ซึ่งมีขั้นตอน คือ

หลังจากที่เซิร์ฟเวอร์ได้รับ Identification String จากไคลเอนต์และตรวจสอบว่ารองรับแล้ว เซิร์ฟเวอร์จะส่งข้อความในรูปแบบ Binary Packet Protocol ที่เป็นชนิดของ SSH_SMSG_PUBLIC_KEY ซึ่งจะประกอบด้วย

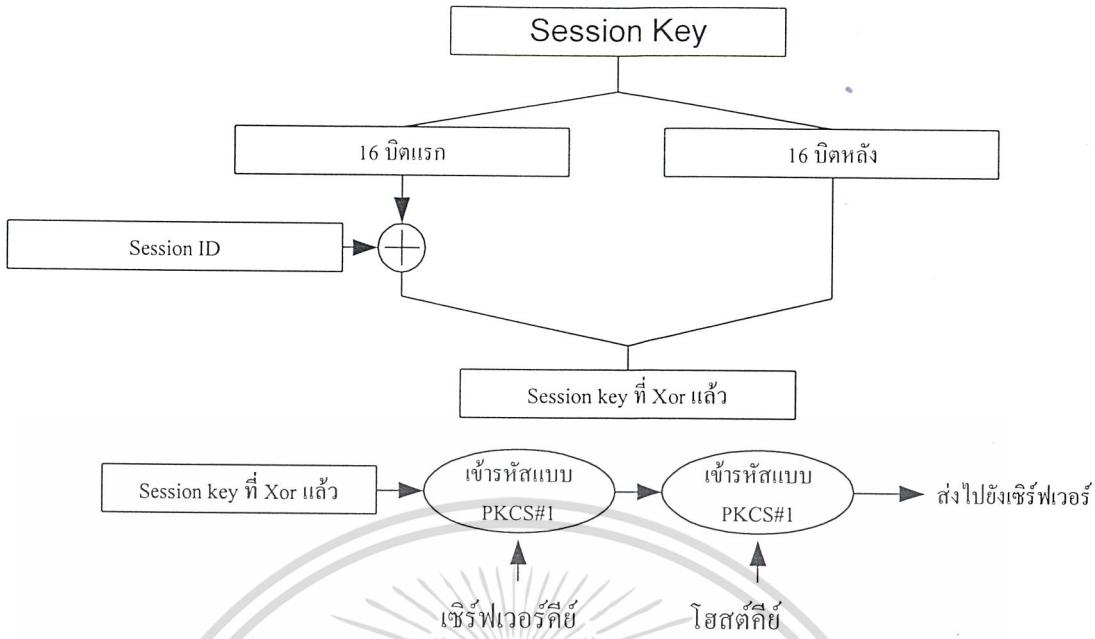
- **ค่าที่สุ่มขึ้นมาขนาด 64 บิต (cookie)** เพื่อให้ไคลเอนต์ส่งข้อมูลนี้กลับมา ทำให้ ยากต่อการทำการปลอมแปลง IP
- **โฮสต์คีย์ (host key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ มีขนาด 1024 บิต
- **เซิร์ฟเวอร์คีย์ (server key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ที่มีการสร้างใหม่ทุกๆ ชั่วโมง มีขนาด 768 บิต
- **การเข้ารหัสที่รองรับ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

เมื่อไคลเอนต์ได้รับข้อความ SSH_SMSG_PUBLIC_KEY ไคลเอนต์จะสร้างกุญแจ (เรียกว่า session key) ที่ใช้สำหรับเข้ารหัสแพคเก็ตต่างๆ ขึ้นมาโดยการสุ่มค่าขึ้น (กุญแจจะมีขนาด 32 ไบต์) เพื่อจะทำการส่งไปให้เซิร์ฟเวอร์ โดยก่อนที่จะทำการส่งจะนำเอาเซสชั่นคีย์ 16 บิตแรก มาเอ็กซ์คลูซีฟออร์ กับ session ID (มีขนาด 16 ไบต์) แล้วนำมาต่อรวมกับ 16 บิตหลังของเซสชั่นคีย์ จากนั้นจึงทำการเข้ารหัสแบบ PKCS#1 (ซึ่งเป็นการเข้ารหัสแบบ RSA) 2 ครั้งด้วยโฮสต์คีย์ และ เซิร์ฟเวอร์คีย์ โดยใช้กุญแจที่สั้นกว่าก่อน (ซึ่งคือ เซิร์ฟเวอร์คีย์)

สำหรับของ session ID เป็นค่าที่ทั้งสองฝั่งคำนวณหาได้ โดยการนำเอาโฮสต์คีย์ ต่อด้วย เซิร์ฟเวอร์คีย์ ต่อด้วย cookie แล้วนำมาคำนวณด้วย อัลกอริทึมแบบ MD5 ซึ่งทั้งสองฝั่งจะได้ค่านี้เช่นเดียวกัน

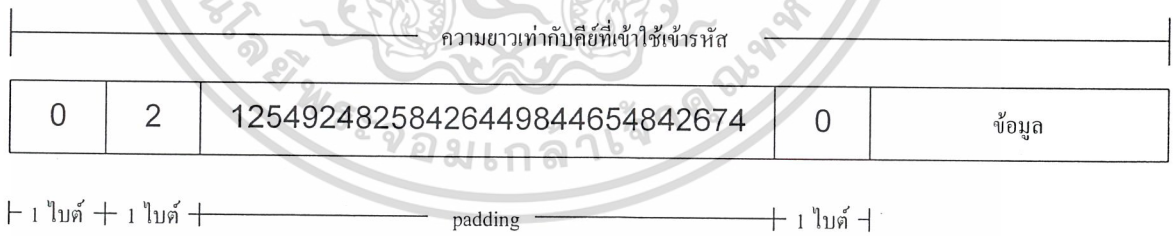


รูป 4.6 แสดงการคำนวณหา Session ID



รูป 4.7 แสดงการเข้ารหัสเซสชั่นคีย์ก่อนส่งให้เซิร์ฟเวอร์

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



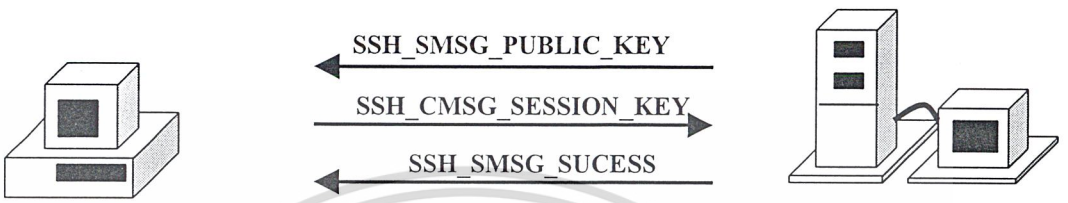
รูป 4.8 แสดงตัวอย่างของข้อมูลที่จะทำการเข้ารหัส PKCS#1

หลังจากที่ได้ข้อมูลที่เป็นคีย์ (ที่ทำการเข้ารหัสข้างต้น) ก็จะส่งแพคเก็ตที่มีชนิดเป็น SSH_CMSG_SESSION_KEY โดยมีข้อมูล ต่างๆ ดังต่อไปนี้ ตามลำดับ

- ชนิดของการเข้ารหัส
- cookie ที่ได้จากเซิร์ฟเวอร์
- Session Key ที่ถูกเข้ารหัสข้างต้น

- Protocol เวอร์ชันที่ใช้

เมื่อเซิร์ฟเวอร์ได้รับแพคเก็ต SSH_CMSG_SESSION_KEY ก็จะถอดรหัสและคำนวณหา session key ออกมา หากลักษณะของข้อมูลถูกต้องก็จะส่ง แพคเก็ตกลับไปว่าเรียบร้อยแล้ว โดยมีชนิดว่า SSH_SMSG_SUCCESS ซึ่งข้อมูลนี้จะถูกทำการเข้ารหัสด้วยวิธีการตามที่ไคลเอนต์เลือกมา



รูป 4.9 ขั้นตอนการส่งของแพคเก็ตต่างๆ ในช่วงการแลกเปลี่ยนคีย์

4.3.3 ช่วงตรวจสอบชื่อและพิสูจน์สิทธิ์ (Declare User Name & Authentication Phase)

ไคลเอนต์จะส่งแพคเก็ตชนิด SSH_CMSG_USER ไปยังเซิร์ฟเวอร์ ซึ่งจะมีชื่อของผู้ใช้ในแพคเก็ตนี้ เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบว่ามีผู้ใช้คนนั้นในรายการหรือไม่ และตรวจสอบการตรวจสอบสิทธิ์ที่ต้องใช้ แล้วจะตอบกลับด้วยแพคเก็ตชนิด SSH_SMSG_SUCCESS ถ้าผู้ใช้คนนี้ถูกกำหนดไว้ว่าไม่ต้องการให้มีการพิสูจน์สิทธิ์ (ไม่ต้องใช้รหัสผ่าน) หรือ SSH_SMSG_FAILURE ถ้าผู้ใช้จำเป็นต้องมีการพิสูจน์สิทธิ์

สำหรับในกรณีที่ไม่มีชื่อของผู้ใช้คนนี้ เซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_FAILURE มาเช่นกัน ทำให้ผู้ใช้ไม่ทราบว่าผู้ใช้คนนี้มีอยู่หรือเปล่า

เมื่อไคลเอนต์ได้รับแพคเก็ต SSH_SMSG_FAILURE ไคลเอนต์จะทำการส่งวิธีการในการพิสูจน์สิทธิ์ให้กับเซิร์ฟเวอร์ ซึ่งวิธีการพิสูจน์สิทธิ์ที่รองรับ มีวิธีต่างๆ ดังนี้

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

ถ้าเซิร์ฟเวอร์ได้รับวิธีการพิสูจน์สิทธิ์จากไคลเอนต์ หากรองรับวิธีการก็จะส่งแพคเก็ตชนิด SSH_SMSG_SUCCESSS แต่หากไม่รองรับก็จะส่งแพคเก็ตชนิด SSH_SMSG_FAILURE เวลาสำหรับการพิสูจน์สิทธิ์จะถูกกำหนดไว้ไม่ให้เกิน 5 นาที (ถ้าหากเกิน เซิร์ฟเวอร์ก็จะทำการยกเลิกการติดต่อ) สำหรับวิธีการตรวจสอบสิทธิ์วิธีต่างๆ มีรายละเอียดดังนี้

ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv

โดยไคลเอนต์จะส่ง SSH_CMSG_AUTH_RHOTS ตามด้วยชื่อของผู้ใช้ฝั่งไคลเอนต์ เมื่อเซิร์ฟเวอร์ที่บนระบบ UNIX ได้รับ ก็จะทำการตรวจสอบจากไฟล์ /etc/host.equiv และจากไฟล์ .rhosts ในไดเรกเตอรี่ของผู้ใช้เอง การติดต่อแบบนี้จะต้องใช้พอร์ตเฉพาะพิเศษ

การวิธีการพิสูจน์สิทธิ์ด้วยระบบที่เชื่อถือฝั่งรีโมต (trusts the remote host) สามารถที่จะถูกอ่านของข้อมูลได้โดยวิธีการปลอมแปลงเลข IP แต่จะถูกป้องกันไว้จากโปรโตคอล (เพราะทุกๆ แพคเก็ตจะถูกเข้ารหัสอยู่)

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

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

โดยไคลเอนต์จะส่ง SSH_CMSG_AUTH_RHOSTS_RSA ตามด้วยชื่อของผู้ใช้และกุญแจสาธารณะของไคลเอนต์

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

ถ้าเซิร์ฟเวอร์รู้โฮสต์คีย์ของเครื่องไคลเอนต์ มันจะทำการตรวจสอบจากโฮสต์คีย์ที่ได้มาว่าตรงกันหรือไม่ ถ้าไม่ ก็จะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อเช่นกัน

ถ้าหากถูกต้องเซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_AUTH_RSA_CHALLENGE ที่บรรจุด้วย challenge ที่ถูกเข้ารหัสสำหรับไคลเอนต์ โดย challenge จะมีขนาดเป็น 32 บิตที่ถูกสุ่มขึ้นมา แล้วทำการจัดข้อความและเข้ารหัสด้วย กุญแจสาธารณะของไคลเอนต์ (วิธีการเช่นเดียวกับที่เข้ารหัส session key)

ไคลเอนต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับ session ID ที่มีอยู่ มาทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพคเก็ตของ SSH_AUTH_RSA_RESPONSE

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

ตรวจสอบสิทธิ์ด้วยวิธี RSA

โดยไคลเอนต์จะส่ง SSH_CMSG_AUTH_RSA ตามด้วยพลับบลิคคีย์โมดูลัสของไคลเอนต์ (ต่างจากวิธี .rhosts + RSA ที่ส่งค่ากุญแจสาธารณะทั้งหมดไป แต่วิธีนี้ส่งเฉพาะค่า n ที่เป็นโมดูลัส ของพลับบลิคคีย์)

เซิร์ฟเวอร์จะตอบกลับทันทีด้วย SSH_SMSG_FAILURE ถ้ามันไม่พบคีย์ของการพิสูจน์สิทธิ์ของไคลเอนต์นี้ในตัวมัน (ค่ากุญแจสาธารณะทั้งหมดของ ไคลเอนต์) ตรงกันข้ามมันจะสร้าง challenge มาทำการเข้ารหัสด้วยกุญแจสาธารณะของไคลเอนต์

ไคลเอนต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับ session ID ที่มีอยู่ มาทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพคเก็ตของ SSH_AUTH_RSA_RESPONSE

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

ตรวจสอบสิทธิ์จากรหัสผ่าน password

โดยไคลเอนต์จะส่ง SSH_CMSG_AUTH_PASSWORD ตามด้วยข้อความ password (ข้อความนี้จะยังไม่มีการทำอะไร แต่จะถูกทำการเข้ารหัสด้วยกลไกของแพคเก็ตโดยอัตโนมัติ)

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบ password และส่ง SSH_SMSG_SUCCESS ถ้าการพิสูจน์สิทธิ์ถูกต้อง และในทางตรงกันข้ามจะส่ง SSH_SMSG_FAILURE

4.3.4 ช่วงการเตรียมการ (Preparatory Operation)

หลังจากที่ทำการพิสูจน์สิทธิ์แล้ว เซิร์ฟเวอร์จะรอคำร้องขอจากไคลเอนต์ และทำการจัดการกับคำสั่งที่เข้ามา และเซิร์ฟเวอร์จะตอบกลับด้วย SSH_SMSG_SUCCESS เมื่อคำร้องขอที่เข้าถูกจัดการเสร็จสิ้น แต่ในกรณีตรงกันข้ามเมื่อคำร้องขอที่เข้ามาไม่สามารถจัดการได้หรือจัดการไม่สำเร็จก็จะส่ง SSH_SMSG_FAILURE ซึ่งข้อความช่วงนี้เป็นการเตรียมการสำหรับในช่วงต่อไป

4.3.5 Interactive Session และ Exchange of Data

ในช่วง interactive session ข้อมูลทุกตัวจะถูกเขียนโดย Shell หรือคำสั่งที่ทำงานอยู่บนเครื่องเซิร์ฟเวอร์เพื่อส่งออกไปยังส่วนแสดงผลหรือส่วนแสดงข้อผิดพลาดบนเครื่องไคลเอนต์ ส่วนอินพุตจะมาจากส่วนรับข้อมูลบนเครื่องไคลเอนต์ ซึ่งจะถูกส่งไปยังโปรแกรมบนเครื่องเซิร์ฟเวอร์

การแลกเปลี่ยนข้อมูลทั้งหมดจะเป็น อะซิงโครนัส (asynchronous) โดยการส่งสามารถเกิดขึ้นได้ทุกเวลา และไม่ต้องมีการตอบสนอง (TCP/IP ปกติจัดการการสร้างความได้น่าเชื่อถืออยู่แล้ว และ packet protocol จะทำการป้องกันการเข้ามายุ่งหรือทำการปลอมแปลง IP)

เมื่อไคลเอนต์ได้รับ EOF จากส่วนรับข้อมูล มันจะส่ง SSH_CMSG_EOF การแลกเปลี่ยนข้อมูลและ interactive mode จะสิ้นสุดลงเมื่อเซิร์ฟเวอร์ส่ง SSH_SMSG_EXITSTATUS เพื่อที่จะแสดงว่าการติดต่อกับตัวไคลเอนต์ได้สิ้นสุดลง ส่วนไคลเอนต์หรือเซิร์ฟเวอร์จะสามารถยกเลิกการติดต่อ เมื่อไรก็ได้โดยการส่งข้อความ SSH_MSG_DISCONNECT หรือทำการปิดการเชื่อมต่อ(close the connection)

4.3.6 การยกเลิกการติดต่อ(Termination of the Connection)

โดยปกติการยกเลิกการติดต่อจะเริ่มโดยเซิร์ฟเวอร์ซึ่งจะทำการส่ง SSH_SMSG_EXITSTATUS หลังจากที่โปรแกรมจบลง ตัวไคลเอนต์จะทำการตอบสนองกับข้อความดังกล่าว โดยการส่ง SSH_CMSG_EXIT_CONFIRMATION และทำการปิดซ็อกเก็ตของตัวเองลง ส่วนเซิร์ฟเวอร์เมื่อได้รับข้อความจึงค่อยทำการปิดซ็อกเก็ตลง เป้าหมายสำคัญสำหรับสำหรับการยืนยันในการยกเลิกคือ ในบางระบบอาจจะสูญเสียข้อมูลที่ส่งไปก่อนหน้านี้เมื่อซ็อกเก็ตถูกปิด และการยกเลิกในฝั่งไคลเอนต์ก่อนทำให้เกิด

TCP/IP TIME_WAIT[RFC 0793] ซึ่งทำให้เซิร์ฟเวอร์รอรับคำตอบจากฝั่งไคลเอนต์ที่ไม่ได้ใช้ ทำให้เซิร์ฟเวอร์ต้องสูญเสียรีซอร์ส

ถ้าในระหว่างโปรแกรมมีสัญญาณที่จะทำให้ยกเลิก ฝั่งตัวเซิร์ฟเวอร์จะส่งแพคเก็ตชนิด SSH_MSG_DISCONNECT พร้อมกับข้อความที่เกี่ยวข้อง ถ้าการติดต่อถูกปิด file descriptor ที่ชี้ไปยังโปรแกรมจะถูกปิดลงและเซิร์ฟเวอร์จะ exit แต่ถ้าโปรแกรมรันบน tty ตัว kernel จะส่งสัญญาณ SIGHUP เมื่อฝั่ง pty master ถูกปิด

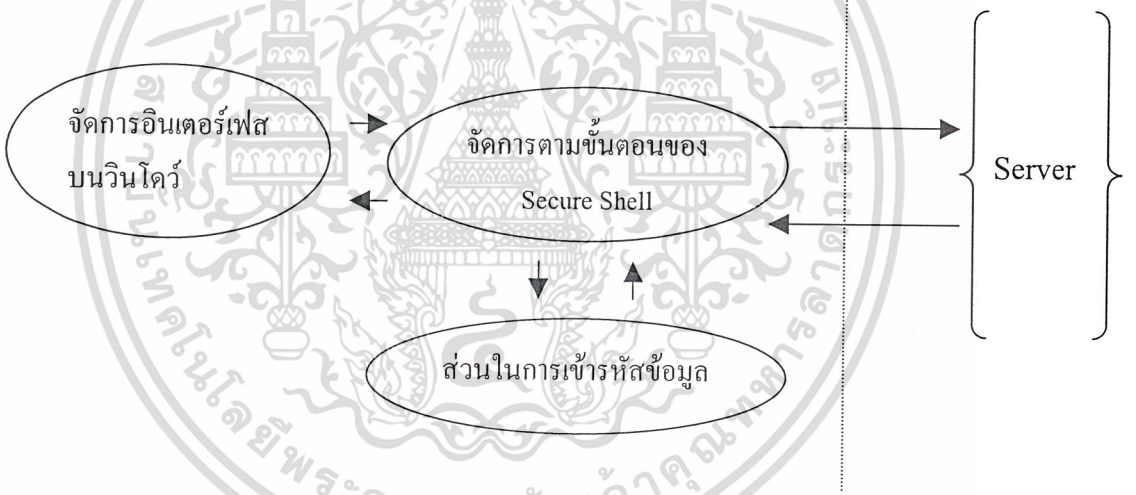
บทที่ 5

การออกแบบ

เนื่องจากการติดต่อบนระบบที่มี Secure Shell แตกต่างจากการติดต่อแบบปกติ คือ ข้อมูลต่างๆ ที่ติดต่อกันจะต้องมีการรักษาความปลอดภัย โดยมีการเข้ารหัสข้อมูลที่ใช้ในการติดต่อกัน แม้ว่าจะมีผู้ดักจับเพื่ออ่านข้อมูลก็ไม่สามารถทราบได้ว่าเป็นข้อมูลอะไร

ดังนั้น ในการออกแบบโปรแกรมเพื่อที่จะทำการติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ซึ่งสามารถทำงานได้บนระบบปฏิบัติการวินโดว์และต้องมีการรักษาความปลอดภัยบนข้อมูลที่ติดต่อกันจะต้องออกแบบให้ทำงานได้ตามมาตรฐาน ซึ่งใช้ทฤษฎีพื้นฐานจากบทต่างๆ ที่ได้กล่าวมา ซึ่งได้แบ่งการออกแบบเป็น 3 ส่วน

1. ส่วนที่การจัดการอินเตอร์เฟสบนวินโดว์ และทำตัวเองเป็นตัวจำลองเทอร์มินัล
2. ส่วนในการจัดการข้อมูล ตามขั้นตอนของ Secure Shell
3. ส่วนในการทำการเข้ารหัสและคำนวณข้อมูล



รูป 5.1 แสดงความสัมพันธ์ของส่วนต่างๆ ที่ออกแบบ

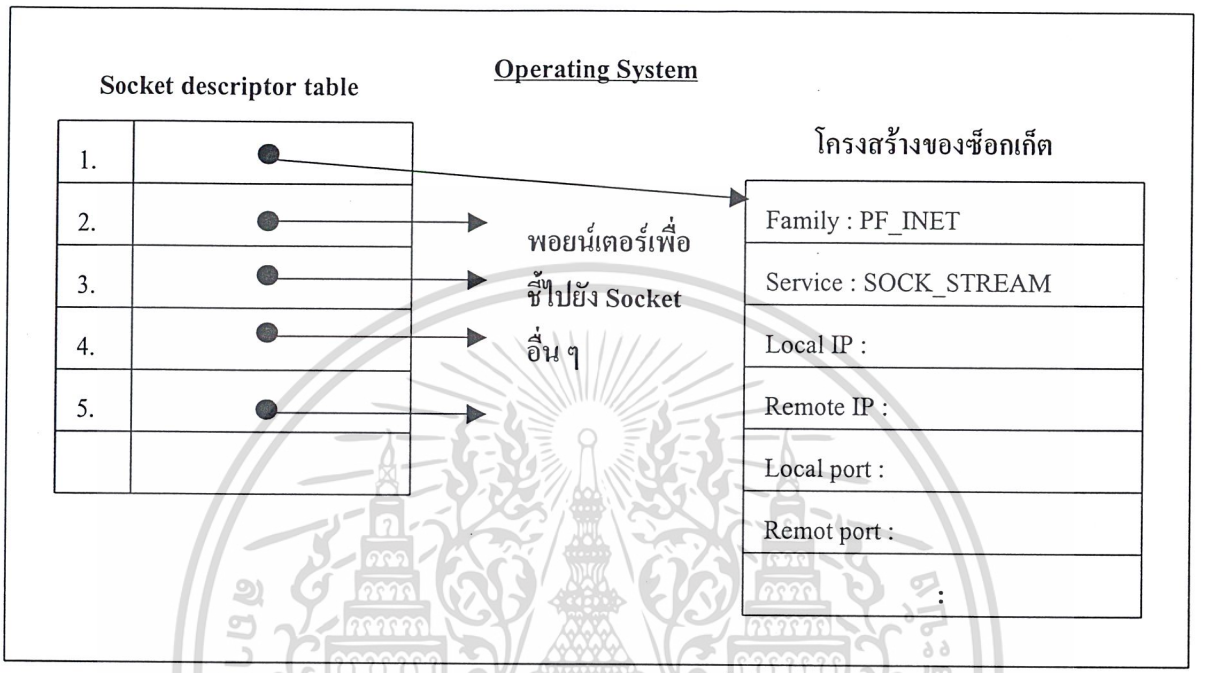
5.1 The Socket API

ก่อนที่จะทำการออกแบบเขียนโปรแกรมในระบบนี้ได้ จำเป็นจะต้องทราบถึงอุปกรณ์ที่ใช้ในการติด1ต่อบนระบบ TCP/IP ซึ่งอุปกรณ์นี้เป็นอุปกรณ์ที่จะใช้เพื่อติดต่อกับระบบ (System Call) ซึ่งอุปกรณ์นี้เรียกว่า Application Program Interface (API) ซึ่งกลุ่มของ API ที่ใช้ในการติดต่อผ่านเครือข่ายคือ ซ็อกเก็ต (Socket)

5.1.1 โครงสร้างของซ็อกเก็ต

ในแต่ละแอปพลิเคชั่นใช้ในการการติดต่อกับเครือข่ายจะต้องมีการเก็บข้อมูลที่ใช้ในการติดต่อ โดยจะมีตารางที่เรียกว่า Socket descriptor table ซึ่งเป็นอาร์เรย์เก็บพอยน์เตอร์เพื่อชี้ไปยังซ็อกเก็ตที่เก็บข้อมูลของแต่ละการติดต่อ

เมื่อแอปพลิเคชั่นจะทำการติดต่อกับเครือข่าย จะต้องทำการสร้าง Socket ขึ้นมา โดยเรียกใช้ฟังก์ชันเพื่อสร้างซ็อกเก็ต ซึ่งระบบปฏิบัติการ (OS) จะทำการจองโครงสร้างข้อมูลมาใหม่ เพื่อเก็บข้อมูลที่ใช้ในการติดต่อแล้วใส่ค่าพอยน์เตอร์ซึ่งชี้ตำแหน่งของซ็อกเก็ตนี้ลงใน Socket descriptor table และเมื่อแอปพลิเคชั่นต้องการใช้ซ็อกเก็ตก็จะถูกเรียกใช้จาก Socket descriptor นี้



รูป 5.2 แสดงการจัดการของ OS หลังจากที่มีการเปิดซ็อกเก็ต

5.1.2 ฟังก์ชันในซ็อกเก็ต

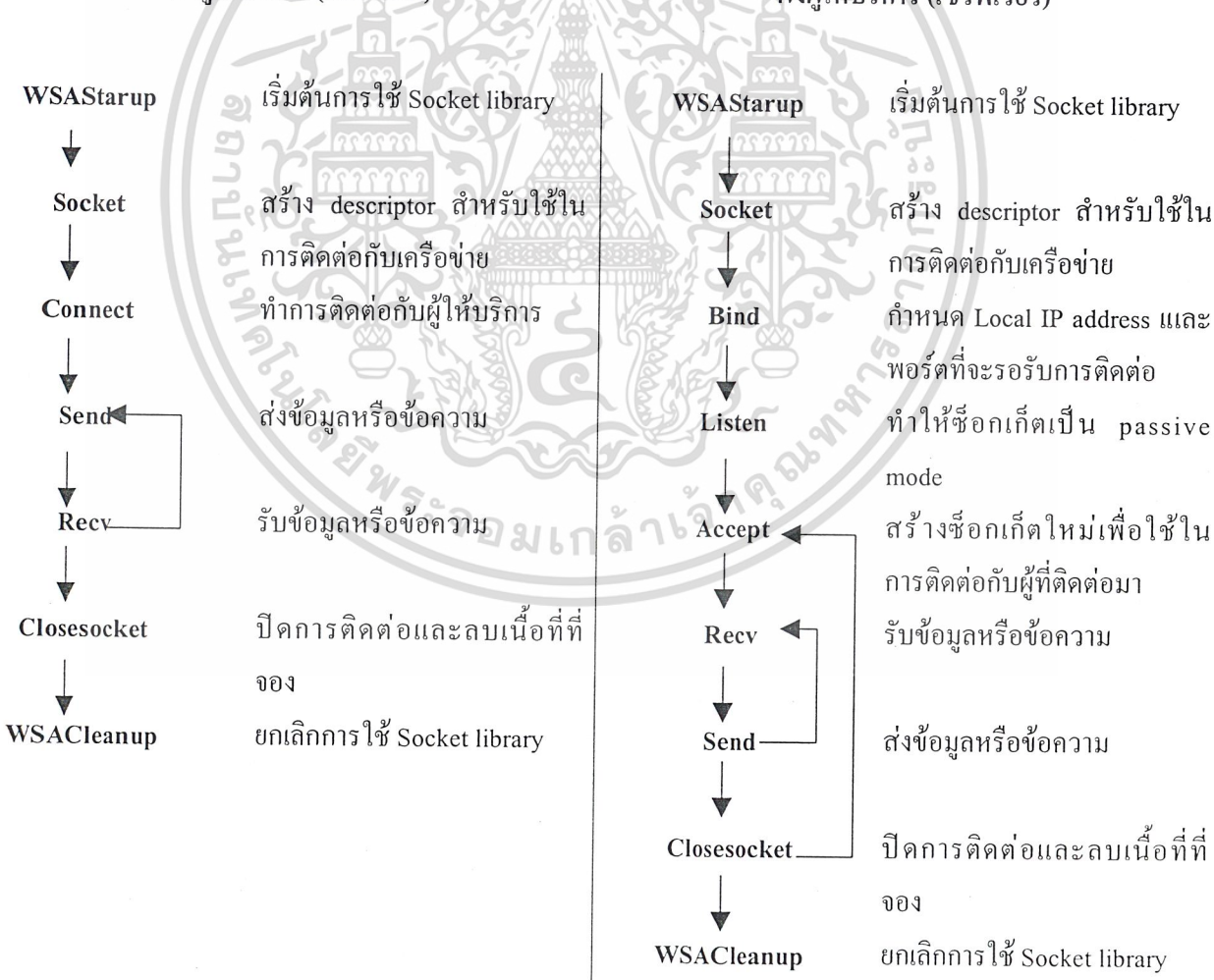
เมื่อซ็อกเก็ตถูกสร้างขึ้น สามารถที่จะเป็นได้ทั้งผู้รับการติดต่อหรือผู้สร้างการติดต่อ ซ็อกเก็ตที่ผู้ให้บริการใช้ (เซิร์ฟเวอร์) จะคอยรับการติดต่อเข้ามา เรียกว่า **Passive Socket** ขณะที่ socket ที่ถูกใช้โดยผู้ใช้บริการ (ไคลเอนต์) เรียกว่า **Active Socket** ซึ่งทั้งสองแบบนี้มีการสร้างเหมือนกัน แต่แตกต่างกันที่การใช้งาน

ในการใช้ซ็อกเก็ตในการติดต่อจะต้องมีการเรียกใช้ System Call โดยมีการส่งผ่านค่าพารามิเตอร์ที่จำเป็นลงไป ซึ่งจะใช้ฟังก์ชันของซ็อกเก็ต API ในการเรียก System Call ต่างๆ ดังต่อไปนี้

1. **ฟังก์ชัน WSAStartup** โปรแกรมที่ใช้วินโดว์ซ็อกเก็ตจะต้องเรียก WSAStartup ก่อนที่จะมีการใช้ซ็อกเก็ต เมื่อเรียกฟังก์ชันนี้ ระบบจะทำการค้นหาไลบารี่ในการติดต่อ
2. **ฟังก์ชัน WSACleanup** เมื่อแอปพลิเคชั่นจบการใช้งานซ็อกเก็ตแล้ว แอปพลิเคชั่นจะเรียกใช้ WSACleanup เพื่อลบการจองซ็อกเก็ตที่จองไว้ทุกๆ อัน
3. **ฟังก์ชัน Socket** เมื่อแอปพลิเคชั่นจะทำการสร้างซ็อกเก็ตขึ้นมาใหม่ จะทำการเรียกฟังก์ชันนี้ พร้อมกับส่งค่าพารามิเตอร์เพื่อบอกชนิดของการติดต่อ ได้แก่ protocal family (เช่น PF_INET สำหรับ TCP/IP) , โปรโตคอลหรือชนิดของการบริการที่ต้องการ (เช่น data

stream หรือ datagram) และชนิดของการบริการที่ใช้จากชนิดของ protocal family (เช่น TCP หรือ UDP)

4. **ฟังก์ชัน Connect** หลังจากที่สร้างซ็อกเก็ตแล้ว ผู้ใช้บริการจะเรียกคำสั่งนี้ เพื่อทำการติดต่อไปยังเซิร์ฟเวอร์ที่ต้องการ
5. **ฟังก์ชัน Send** ทั้งผู้ใช้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการส่งข้อมูล โดยจะทำการก็อบปี้ข้อมูลที่จะส่งลงในบัฟเฟอร์เพื่อที่จะส่งออกไป
6. **ฟังก์ชัน Recv** ทั้งผู้ใช้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการรับการข้อมูล
7. **ฟังก์ชัน Closesocket** ฟังก์ชันนี้จะใช้สำหรับจบการใช้ซ็อกเก็ตในแต่ละอัน
8. **ฟังก์ชัน Bind** เป็นฟังก์ชันที่ผู้ให้บริการเป็นผู้ใช้ในการเปิดพอร์ตเพื่อรอรับการติดต่อ
9. **ฟังก์ชัน Listen** เป็นฟังก์ชันเพื่อที่จะรอการติดต่อจากผู้ใช้บริการ
10. **ฟังก์ชัน Accept** เป็นฟังก์ชันที่ผู้ใช้บริการใช้ เพื่อทำการเปิดซ็อกเก็ตใหม่เพื่อรับสร้างเป็นคู่ในการติดต่อของผู้ใช้บริการแต่ละคน



รูป 5.3 แสดงลำดับการจัดการซ็อกเก็ตบน TCP/IP

5.2 ส่วนต่างๆ จากการออกแบบ

สำหรับแอปพลิเคชั่นที่ถูกเขียนขึ้นจะเขียนจากโปรแกรม Visual C++ ดังนั้น โปรแกรมที่ออกแบบขึ้น จึงเป็นลักษณะโครงสร้างแบบเชิงวัตถุ (Object Oriented)โดยแต่ละส่วนจะมีส่วนประกอบ และรายละเอียด ดังต่อไปนี้

5.2.1 ส่วนจัดการอินเตอร์เฟสกับวินโดว์และส่วนจำลองเทอร์มินอล

ในส่วนอินเตอร์เฟสกับผู้ใช้ จะต้องนำเอาข้อมูลที่ได้รับมาแสดงผลบนหน้าจอของผู้ใช้ ซึ่งบนโปรแกรม Visual C++ นั้นจะมีเครื่องมือที่ใช้ในการสร้างส่วน อินเตอร์เฟส ซึ่งเป็น Tools ของ MFC (Microsoft Foundation Class) ซึ่งจะทำการสร้างโค้ดในส่วนของการทำอินเตอร์เฟสหลักๆ ขึ้นมาโดยอัตโนมัติ

สำหรับโหมดที่เลือกในการสร้างแอปพลิเคชั่นนี้ โดยเลือกเป็นแบบ SDI โดยเป็นหน้าต่างวินโดว์หน้าต่างเดียว (ไม่มีลูกหน้าต่าง) ซึ่งจะได้ลักษณะคล้ายโปรแกรมโน๊ตแพ็ต (Notepad)

ส่วนในตัวจำลองเทอร์มินอลจะเป็นออบเจ็กที่คอยจัดการกับข้อมูลที่ได้รับเข้ามาจากส่วนที่จัดการข้อมูลตามขั้นตอนของ Secure Shell ซึ่งจะทำการแปลความหมายของข้อมูลที่เข้ามาว่าข้อมูลที่จะต้องนำมาแสดงผลนี้อยู่บรรทัดใด สีอะไร แล้วทำการส่งไปแสดงผลยังส่วนอินเตอร์เฟส สำหรับเทอร์มินัลที่ใช้ได้ทำการออกแบบนี้เป็นเทอร์มินอลชนิด VT100

สำหรับการรับข้อมูลจากอินพุตที่ผู้ใช้ป้อนเข้ามา ข้อมูลต่างๆ ที่รับมาไม่จำเป็นต้องผ่านตัวจำลองเทอร์มินอล ซึ่งจะส่งข้อมูลไปให้ส่วนที่จัดการตามขั้นตอนของ Secure Shell อัตโนมัติ
ในส่วนนี้จะมีออบเจ็กต์ที่ใช้ในการจัดการจำลองเทอร์มินอล คือ SSH_Terminal

รูป 5.4 แสดงการติดต่อของข้อมูลในส่วนจัดการอินเตอร์เฟสและจำลองเทอร์มินอล

5.2.2 ส่วนจัดการข้อมูลตามขั้นตอนของโปรโตคอล Secure Shell

ในส่วนนี้จะต้องเขียนโปรแกรมขึ้นเพื่อจัดการกับข้อมูลตามมาตรฐาน Secure Shell ซึ่งได้ออกแบบเป็นลักษณะเชิงวัตถุได้ ประกอบด้วยออบเจ็กต์ต่างๆ ดังนี้

1. **ออบเจ็กต์ *SSH_Socket*** ใช้รับและส่งแพคเก็ตผ่านที่ผ่านทางซ็อกเก็ตบนระบบ TCP/IP ประกอบด้วยวิธีการที่สำคัญ คือ

- **Connect** เป็นฟังก์ชันเพื่อเปิดการติดต่อเข้าหาเซิร์ฟเวอร์

- **Send** เพื่อส่งข้อมูลเข้าหาเซิร์ฟเวอร์
- **Recv** เพื่อรับข้อมูลจากเซิร์ฟเวอร์และนำข้อมูลมาจัดการ
- **Close** เพื่อปิดซ็อกเก็ตที่ทำการติดต่อ

2. ออบเจ็กต์ ***SSH_Packet*** ใช้แยกข้อมูลในแพคเก็ตที่รับมาเป็นข้อมูลชนิดต่างๆ แล้วนำมาประมวลผลหรือจัดข้อมูลต่างๆ ลงเป็นแพคเก็ตเพื่อส่งกลับไปหาเซิร์ฟเวอร์ รวมทั้งทำการตรวจสอบความถูกต้องของแพคเก็ตที่รับมา (เช่นตรวจสอบ CRC) ประกอบด้วยวิธีการที่สำคัญ คือ
   - **Packet** เพื่อจัดการนำข้อมูลที่จะส่งมาทำเป็นแพ็คเก็ต
   - **GetPacketfromBytes** เพื่อตีความหมายจากข้อมูลที่เข้ามาจากเซิร์ฟเวอร์ที่เป็นไบต์ๆ เข้ามาเพื่อแยกเป็นส่วนต่างๆ ข้อมูล
   - **CheckCRC** เพื่อตรวจสอบ CRC จากข้อมูลที่เข้ามา
3. ออบเจ็กต์ ***SSH_Crypto*** ใช้ในการเข้ารหัสข้อมูล โดยในออบเจ็กต์นี้จะเก็บอัลกอริทึมในการเข้าและถอดรหัสข้อมูลแบบต่างๆ ไว้ เช่น 3DES , DES , IDEA (ในแอปพลิเคชั่นที่สร้างมีเพียง 3DES เท่านั้น) ประกอบด้วย method ที่สำคัญ คือ
   - **SetKey** เพื่อจัดการกำหนดคีย์ในการเข้ารหัส
   - **Encryption** เพื่อเข้ารหัสข้อมูลที่จะทำการส่ง
   - **Decryption** เพื่อถอดรหัสข้อมูลที่รับเข้ามา
   - **EncryptRSA** เพื่อเข้ารหัสแบบ RSA (ตามมาตรฐาน PKCS#1)
4. ออบเจ็กต์ ***SSH_IO*** ใช้ในการตีความข้อมูลที่เข้ามาว่าเป็นข้อมูลชนิดใด และทำอะไรต่อไป รวมทั้งจัดการกับข้อมูลที่จะส่งออกไปว่าจะมีลักษณะใด (เช่น ทำการส่งชื่อของผู้ใช้หลังจากเซิร์ฟเวอร์ ตอบรับหลังจากที่เราส่ง Session key ไป) ประกอบด้วยวิธีการที่สำคัญ คือ
   - **HadlePacket** เพื่อแยกว่าเป็นแพคเก็คชนิดใด และกระทำตามขั้นตอนของ SSH
   - **Send** เพื่อทำการส่งข้อมูลไปหาเซิร์ฟเวอร์

5.2.3 ส่วนในการเข้ารหัสข้อมูล

ในแอปพลิเคชั่นนี้จะประกอบด้วยออบเจ็กต์ของการเข้ารหัสและการคำนวณ

- การเข้ารหัสแบบ DES
- การเข้ารหัสแบบ 3DES ในโหมด CBC
- การเข้าแบบ RSA
- การคำนวณแบบ MD5

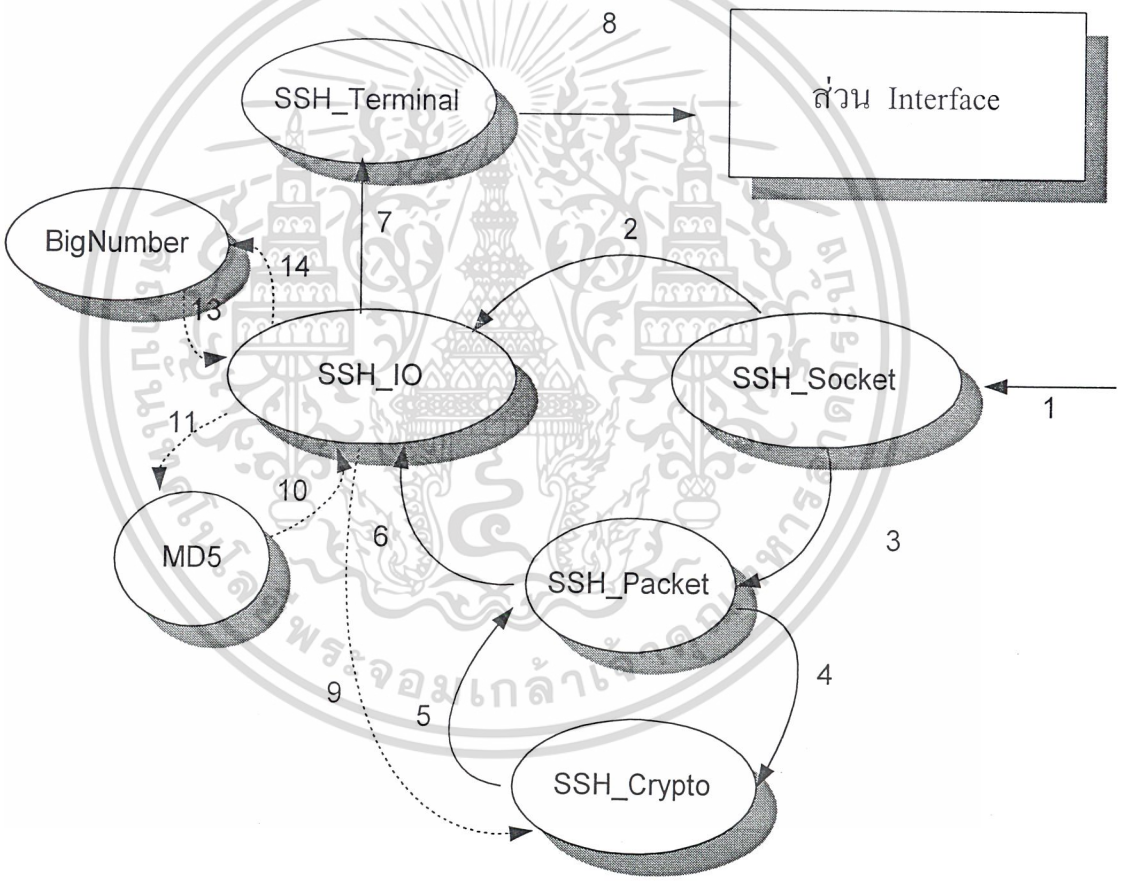
ซึ่งออบเจ็กต์ที่ได้ออกแบบมีดังนี้

1. ออบเจ็กต์ ***MD5*** ใช้ทำการคำนวณแบบ MD5 ซึ่งจะทำให้ข้อมูลต่างๆ ที่เข้ามามีขนาดเหลือเป็น 16 ไบต์ (256 บิต)

2. *ออบเจ็กต์ BigNumber* ใช้สำหรับการคำนวณกับข้อมูลที่มีขนาดใหญ่ ซึ่งใช้ประกอบในการเข้ารหัสแบบ RSA (เนื่องจากการเข้ารหัสแบบ RSA จะนำข้อมูลปกติมาทำการยกกำลังและมอดูโลกับข้อมูลขนาดหลายร้อยบิต)
3. **ออบเจ็กต์ *SSH_Terminal*** ใช้เพื่อการจำลองตัวเองเป็นเทอร์มินัล ๆ หนึ่งให้ทำงานได้และแสดงผลตามคำสั่งของเซิร์ฟเวอร์

5.3 ไดอะแกรมของการออกแบบ

จากที่เราได้ออกแบบออบเจ็กต์ต่างๆ ขึ้นมา มีขั้นในการติดต่อซึ่งไดอะแกรมข้างล่างนี้เป็นลักษณะการทำงานของออบเจ็กต์ต่างๆ เมื่อได้รับข้อมูลจากเซิร์ฟเวอร์



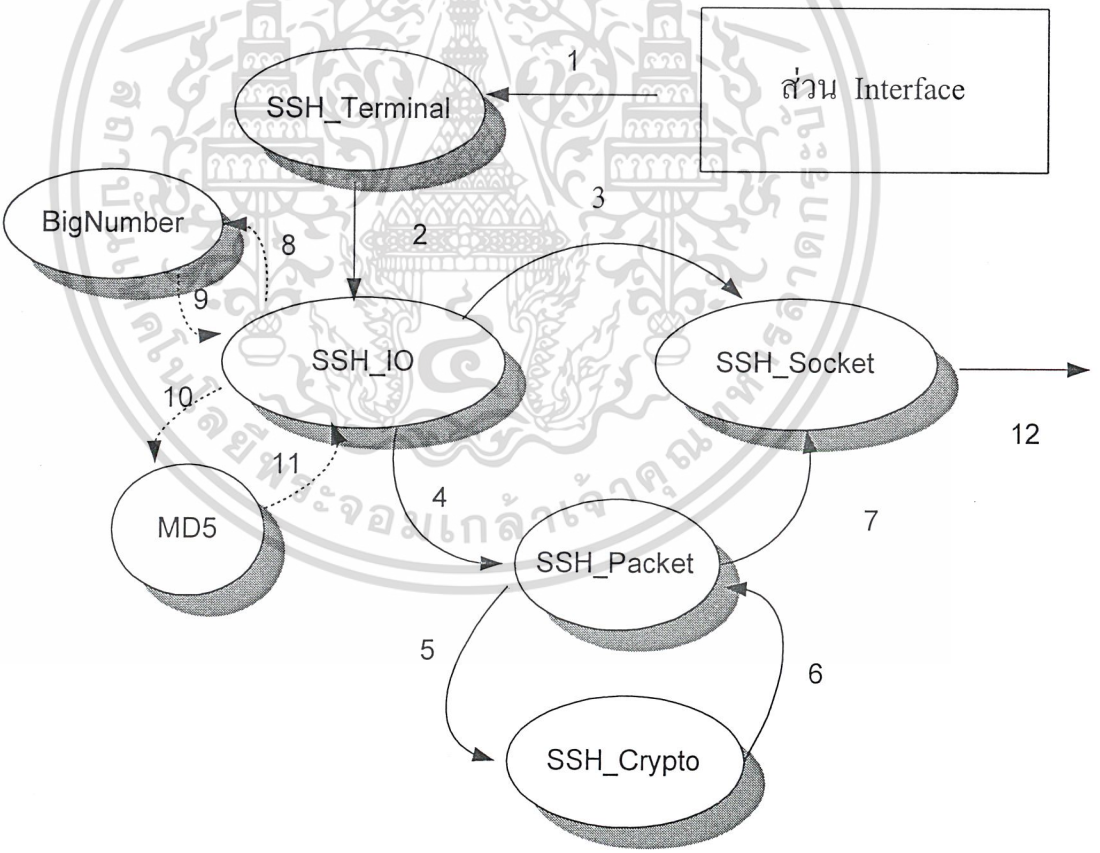
รูป 5.5 แสดงออบเจ็กต์และการทำงานเมื่อได้รับข้อมูลจากเซิร์ฟเวอร์

จากไดอะแกรมข้างต้น แต่ละเส้นแทนการทำงานดังต่อไปนี้

1. มีแพคเก็ตเข้ามาผ่านทางซ็อกเก็ต ซึ่งทำให้ ออบเจ็กต์ **SSH_Socket** ทำงาน
2. ซ็อกเก็ตส่งข้อมูลเข้าสู่ออบเจ็กต์ **SSH_IO** โดยตรง (เมื่อยังไม่เข้าสู่ระบบ Binary Packet Protocol)

3. ซ็อกเก็ตส่งข้อมูลที่เป็นแพคเก็ตมาเพื่อให้ออบเจ็กต์ **SSH_Packet** ทำการจำแนกข้อมูล
4. ส่งข้อมูลไปที่ออบเจ็กต์ **SSH_Crypto** เพื่อทำการถอดรหัสข้อมูล
5. ส่งข้อมูลที่ถอดรหัสกลับเข้าสู่ **SSH_Packet** เพื่อจำแนกข้อมูลอีกทีหนึ่ง
6. ส่งข้อมูลที่ถูกแยกออกเป็นฟิลด์ชนิดต่างๆ แล้ว เพื่อทำการทำตามขั้นตอนของโปรโตคอล Secure Shell
7. ข้อมูลที่ใช้สำหรับควบคุมหรือจัดการตัวเทอร์มินอล รวมทั้งแปลข้อมูลตามรูปแบบของเทอร์มินอล
8. นำข้อมูลออกสู่ส่วนแสดงผล
9. สัญญาณบอก **SSH_Crypto** เพื่อบอกว่าจะให้มีการเข้ารหัสข้อมูลหรือไม่ แบบใด
10. ส่งข้อมูลเพื่อทำการคำนวณแบบ MD5 โดยข้อมูลที่ส่งไปมีขนาดได้ไม่จำกัด
11. ข้อมูลที่ได้มาจากการทำ MD5 โดยจะมีขนาด 128 บิต (16 ไบต์)
12. ข้อมูลที่ใช้สำหรับคำนวณทางคณิตศาสตร์ขนาดใหญ่ เช่น เพื่อเข้าหรือถอดรหัสแบบ RSA
13. ข้อมูลที่ได้รับจากการคำนวณทางคณิตศาสตร์ขนาดใหญ่

ไดอะแกรมข้างล่างนี้เป็นลักณะการทำงานของออบเจ็กต์ต่างๆ เมื่อจะส่งข้อมูลไปหาเซิร์ฟเวอร์



รูป 5.6 แสดงออบเจ็กต์และการทำงานเมื่อจะส่งข้อมูลจากเซิร์ฟเวอร์

จากไดอะแกรมข้างต้น แต่ละเส้นแทนการทำงานดังต่อไปนี้

1. มีอินพุตเข้ามาจากส่วนอินเตอร์เฟส

2. ข้อมูลที่ถูกจะนำมาเข้า **SSH_IO** โดยตรง (เพื่อแปลเข้าสู่ระบบ Binary Packet Protocol)
3. ข้อมูลที่ส่งไปออกไปหาเซิร์ฟเวอร์โดยตรง (ไม่เป็นแพคเก็ต มีในช่วงแรกสุดเท่านั้น)
4. ข้อมูลจะถูกนำเข้าไปทำเป็นแพคเก็ต
5. ข้อมูลถูกทำการเข้ารหัส
6. ข้อมูลส่งกลับไปยัง **SSH_Packet**
7. ข้อมูลส่งไปให้ซ็อกเก็ต เพื่อทำการส่งไปยังเซิร์ฟเวอร์
8. ข้อมูลที่ใช้สำหรับคำนวณทางคณิตศาสตร์ขนาดใหญ่ เช่น เพื่อเข้าหรือถอดรหัสแบบ RSA
9. ข้อมูลที่ได้รับจากการคำนวณทางคณิตศาสตร์ขนาดใหญ่
10. ข้อมูลที่จะทำการคำนวณแบบ MD5 มีขนาดไม่จำกัด
11. ข้อมูลที่ได้มาจากการทำ MD5 มีขนาด 128 บิต (16 ไบต์)
12. ข้อมูลถูกส่งออกไปยังเซิร์ฟเวอร์

บทที่ 6

การทดลองและผลการทดลอง

ในการออกแบบโปรแกรมจำเป็นที่จะต้องมีการทดลองเพื่อทดสอบสิ่งที่สร้างขึ้นมา ซึ่งได้ทดลองติดตั้งโปรแกรมลงระบบเครือข่าย ที่มีเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell

6.1 จุดประสงค์ของการทดลอง

6.1.1 เพื่อทดสอบแอปพลิเคชั่นที่สร้างขึ้นตามความรู้และทฤษฎีของ Secure Shell โปรโตคอล และการเข้ารหัสข้อมูลแบบต่างๆ จากที่ได้ศึกษามาว่าสามารถนำมาใช้ได้จริงและทำให้การสื่อสารไม่ผิดพลาด

6.1.2 เพื่อเป็นแนวทางในการนำเอาความรู้เกี่ยวการรักษาความปลอดภัย นำไปประยุกต์ใช้ให้เหมาะสมกับงานใดๆ

6.2 การเตรียมอุปกรณ์และสภาวะการทำงานเพื่อทดลองโปรแกรม

อุปกรณ์ที่ใช้การรันโปรแกรมและอุปกรณ์ที่ต้องใช้ในระบบ

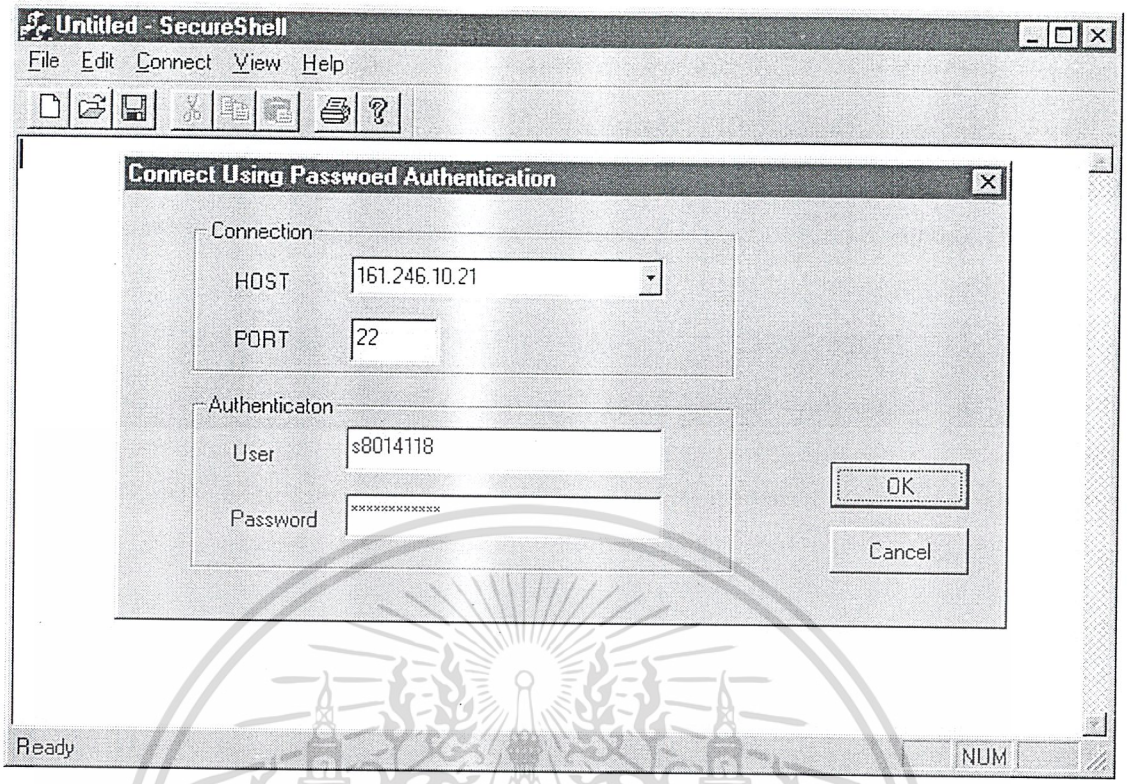
- เครื่องคอมพิวเตอร์ไคลเอนต์ ต้องมีวินโดว์ 95 , 98 หรือ NT เป็นระบบปฏิบัติการ
- เครื่องคอมพิวเตอร์เซิร์ฟเวอร์ ซึ่งจะต้องมีการลงโปรแกรม Secure Shell ไม่เกินเวอร์ชัน 1

6.3 ทำการทดลองโปรแกรมและผลการทดลอง

ในการทดลองนี้ได้แบ่งการทดลองออกเป็นช่วงๆ ซึ่งในบางช่วงให้มีการแสดงผลข้อมูลขึ้นเพื่อใช้ในการตรวจสอบ ซึ่งในการใช้งานจริงนั้นไม่จำเป็นต้องให้แสดงผลก็ได้

6.3.1 ช่วงการเตรียมตัวติดต่อกับเซิร์ฟเวอร์

เมื่อเปิดโปรแกรมขึ้นแล้ว เริ่มทำการติดต่อโดยเลือกไปที่ Connect -> Login... จะขึ้นไดอะล็อกการพิสูจน์สิทธิ์แบบใช้รหัสลับ (password Authentication) เพื่อให้ทำการกำหนดเครื่องเซิร์ฟเวอร์ที่ใช้ในการติดต่อ พอร์ตที่ใช้ รวมถึงชื่อของผู้ใช้และรหัสลับเพื่อทำการล็อคเข้าไป (รายละเอียดที่กรอกตรงนี้รวมทั้งรหัสลับจะเป็นอันเดียวกันกับการใช้รีโมตแอคเซสตัวอื่นๆ เช่น rlogin หรือ telnet เพียงแต่เปลี่ยนพอร์ตที่ใช้เป็นพอร์ต 22 เพราะข้อมูลของการติดต่อจะเป็นข้อมูลเดียวกันข้อมูลที่ติดต่อเมื่อใช้โดย rlogin หรือ telnet)

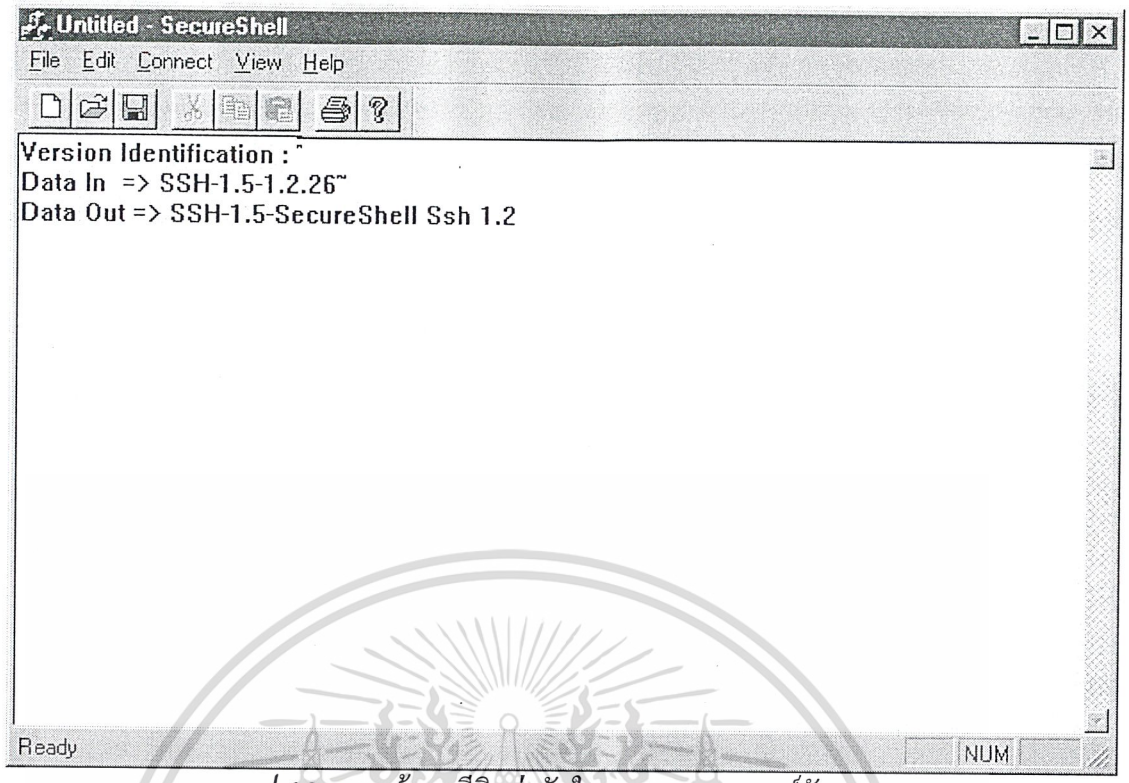


รูป 6.1 แสดงหน้าจอของการกำหนดค่าในการติดต่อ

6.3.2 ช่วงในการทำความปลอดภัยและตรวจสอบสิทธิ์

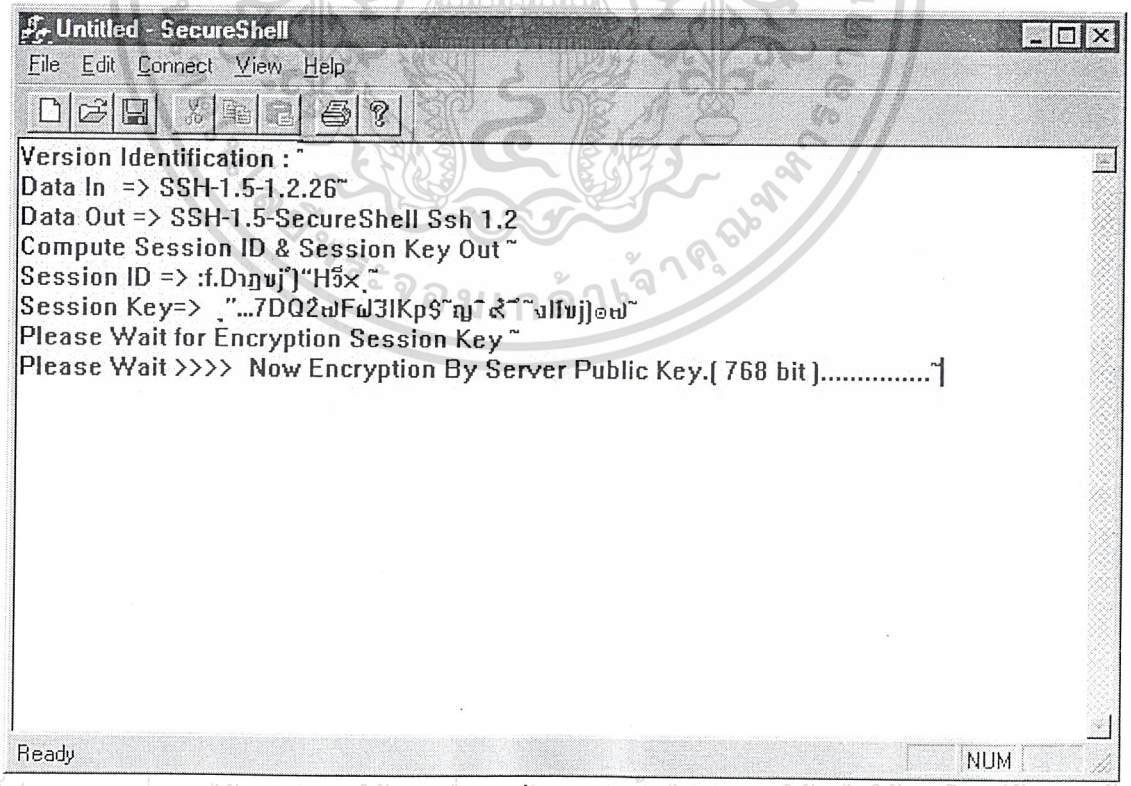
ในช่วงนี้ จากทฤษฎีที่ได้ศึกษามาจะเห็นว่าการติดต่อได้แบ่งเป็นช่วงๆ ดังนั้น เพื่อที่จะได้เห็นลักษณะของข้อมูลในการติดต่อว่ามีการทำงานเช่นไร และถูกต้องไหม จึงได้ให้มีการแสดงผลของข้อมูลที่แลกเปลี่ยนกัน ซึ่งข้อมูลนี้ในการใช้จริงไม่จำเป็นต้องแสดงให้เห็น

- **ช่วงการตรวจสอบเวอร์ชัน** เมื่อเริ่มทำการติดต่อ (connect) กับเซิร์ฟเวอร์ เซิร์ฟเวอร์จะส่งข้อมูลของเซิร์ฟเวอร์เพื่อบอกเวอร์ชันที่ใช้ได้ เมื่อไคลเอนต์ตรวจสอบแล้วก็ทำการส่งเวอร์ชันของตนกลับไปยังเซิร์ฟเวอร์



รูป 6.2 แสดงข้อมูลที่ติดต่อกันในการตรวจสอบเวอร์ชัน

- **ช่วงการแลกเปลี่ยนคีย์ที่ใช้** เมื่อตรวจสอบเวอร์ชันว่าใช้กันได้แล้ว จะได้รับข้อมูลที่เป็นพับบลิคคีย์และข้อมูลอื่นๆ จากเซิร์ฟเวอร์ เพื่อนำมาใช้ในการแลกเปลี่ยนคีย์ ซึ่งมีลักษณะของข้อมูลต่างๆ ดังรูป



รูป 6.3 แสดงข้อมูลที่ใช้สำหรับแลกเปลี่ยนคีย์

- **ช่วงการตรวจสอบผู้ใช้และพิสูจน์สิทธิ์** เมื่อทำการแลกเปลี่ยนคีย์แล้วเซิร์ฟเวอร์จะทำการตรวจสอบชื่อผู้ใช้และรหัสผ่าน ซึ่งข้อมูลที่ส่งไปตอนนี้จะถูกเข้ารหัสแล้ว ซึ่งเมื่อจับข้อมูลจากโปรแกรม NetX-ray แล้ว นำข้อมูลมาเปรียบเทียบกับโปรแกรม telnet พบว่าข้อมูลจะไม่สามารถอ่านได้ตามปกติเมื่อใช้ผ่าน Secure Shell

ข้อมูล password ปกติ	ข้อมูลที่ได้จาก Telnet	ข้อมูลที่ได้จากโปรแกรม Secure shell
Fxicdi,g,viN	**F , x , i , c , d , i , g , v , i ,N**	. . . . . ▌▌ÀR·▌P. ýûx. . . . . Ú. .á®L ³»*ùÎÈ▌:²^ßÜh\¬| >S

รูป 6.4 แสดงการเปรียบรหัสผ่านที่อ่านได้บน Net X-ray ระหว่างโปรแกรม telnet และ Secure Shell

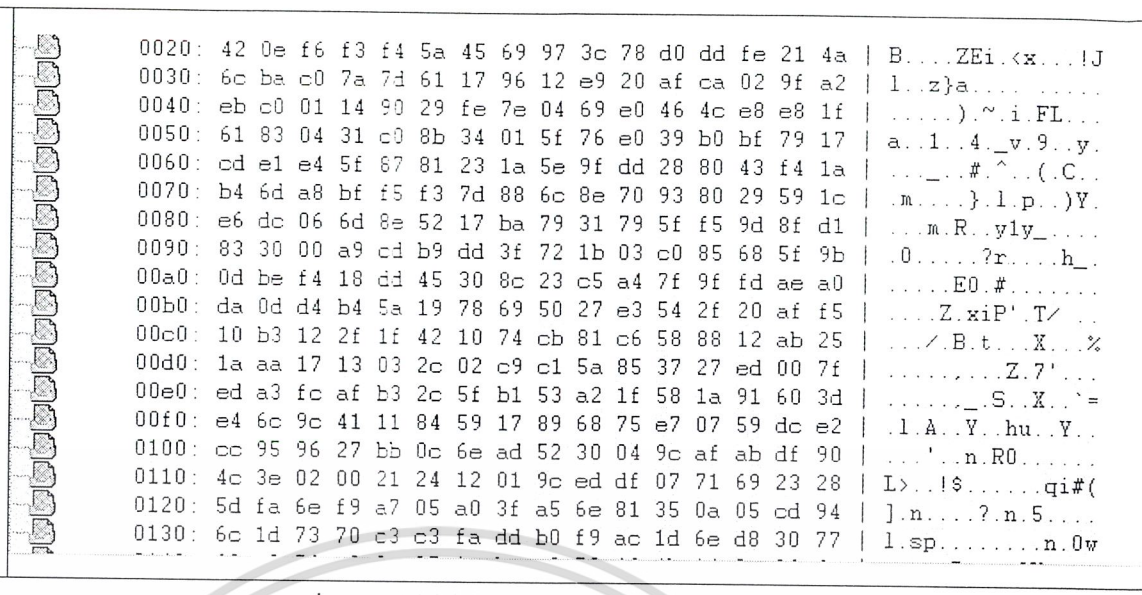
6.3.3 ช่วงการแลกเปลี่ยนข้อมูลแบบ Interactive

ในช่วงนี้จะทำการแลกเปลี่ยนข้อมูลในลักษณะที่เมื่อไคลเอนต์กดคีย์บอร์ดใดๆ ก็จะส่งข้อมูลนี้ไปให้เซิร์ฟเวอร์แปลความและส่งข้อมูลกลับมา ซึ่งเมื่อกับการติดต่อแบบรีโมตแอคเซสตามปกติ แต่การสื่อสารในช่วงนี้ข้อมูลจะต้องมีการเข้ารหัสข้อมูล ซึ่งเมื่อนำมาแสดงผลนั้นก็จะมีการถอดรหัสเพื่อแสดงผล ซึ่งเมื่อเปรียบเทียบกับ telnet จะเห็นว่ามีความปลอดภัยกว่าเพราะข้อมูลนี้ไม่สามารถนำไปอ่านได้ทันทีเมื่อเปรียบเทียบกับ telnet ซึ่งข้อมูลนี้สามารถอ่านเข้าใจได้ทันที

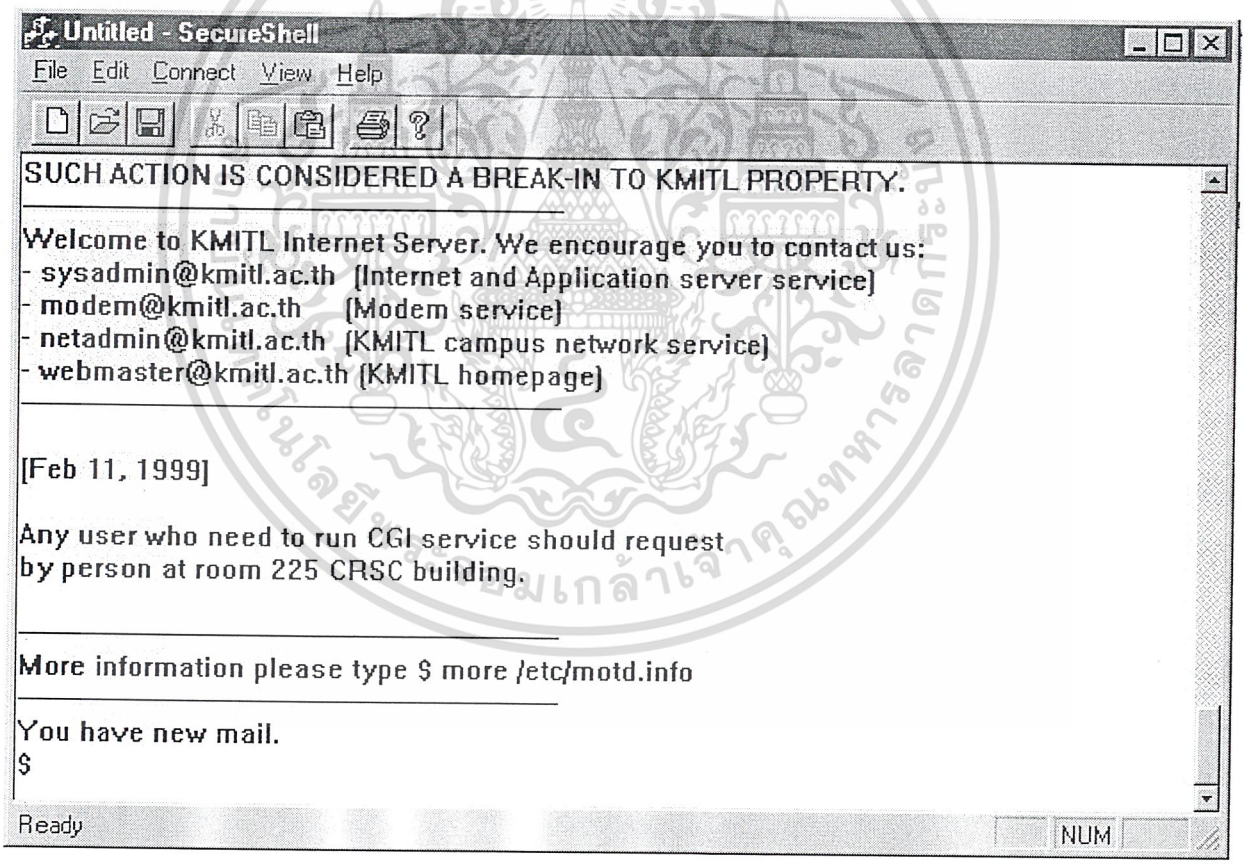
ข้อมูลจากโปรแกรม Telnet	

```
0020: 20 20 70 75 62 6c 69 63 5f 68 74 6d 6c 20 20 20 |   public_html
0030: 20 20 20 74 70 69 63 73 0d 0a 61 64 64 5f 74 61 |    tpics..add_ta
0040: 6c 6b 20 20 20 20 20 20 20 20 20 6c 65 78 20 20 | lk          lex
0050: 20 20 20 20 20 20 20 20 20 20 20 20 73 38 30 31 |              s801
0060: 34 31 31 38 2e 74 78 74 20 20 20 20 20 75 73 65 | 4118.txt     use
0070: 73 0d 0a 61 73 73 32 5f 34 64 34 31 31 38 2e 7a | s..ass2_4d4118.z
0080: 69 70 20 20 6d 61 69 6c 20 20 20 20 20 20 20 20 | ip  mail
0090: 20 20 20 20 20 73 38 30 31 34 31 31 38 2e 7a 69 |      s8014118.zi
00a0: 70 20 20 20 20 20 77 65 62 70 61 67 65 0d 0a 62 | p     webpage..b
00b0: 6f 6f 6b 6d 61 72 6b 20 20 20 20 20 20 20 20 20 | ookmark
00c0: 6d 65 6d 62 65 72 20 20 20 20 20 20 20 20 20 20 | member
00d0: 20 73 65 63 75 72 65 20 20 20 20 20 20 20 20 20 |  secure
00e0: 20 20 79 61 63 63 0d 0a 62 6f 72 72 6f 77 20 20 |   yacc..borrow
00f0: 20 20 20 20 20 20 20 20 20 70 6f 6d 2e 64 6f 63 |          pom.doc
0100: 20 20 20 20 20 20 20 20 20 20 20 73 6d 6f 62 61 |           smoba
0110: 6b 0d 0a 64 65 61 64 2e 6c 65 74 74 65 72 20 20 | k..dead.letter
0120: 20 20 20 20 70 72 6f 67 72 61 6d 20 20 20 20 20 |     program
```

ข้อมูลจากโปรแกรม Secure Shell



รูป 6.5 แสดงการเปรียบข้อมูลที่อ่านได้บน Net X-ray ระหว่างโปรแกรม telnet และ Secure Shell



รูป 6.6 แสดงการข้อมูลที่แสดงบนโปรแกรม Secure Shell

บทที่ 7

บทวิจารณ์และสรุป

7.1 บทสรุปและวิเคราะห์ของมาตรฐาน Secure Shell

ในการติดต่อสื่อสารผ่านระบบเครือข่ายในปัจจุบัน นอกจากความรวดเร็วและถูกต้องของข้อมูลแล้ว สิ่งที่ต้องคำนึงถึง คือ ความปลอดภัยของข้อมูลในที่ใช้ในการติดต่อสื่อสาร สำหรับโครงงานที่ได้จัดทำขึ้นนี้ เป็นโปรแกรมที่ใช้ติดต่อกับเซิร์ฟเวอร์ ตามมาตรฐานของโปรโตคอล Secure Shell ซึ่งมีความปลอดภัยมากกว่าโปรโตคอล Telnet และโปรโตคอลของ rlogin ดังนี้

- การปลอมแปลง IP จะถูกจำกัด เนื่องจากมีการเช็คถึงผู้ใช้ที่ถูกต้อง โดยการเข้ารหัส host keys และ cookie ที่สุ่มขึ้น
- ข้อมูลจะถูกเข้ารหัสซึ่งทำให้ยากในการลอบฟังและปลอมแปลงข้อมูล ซึ่งรวมทั้งมีการเข้ารหัสข้อมูลที่ใช้พิสูจน์สิทธิ์ เช่น รหัสลับ (password)
- Man-in-the-middle attacks ไม่สามารถทำได้ เนื่องจากมีการใช้ host key เพื่อเข้ารหัส session key

ซึ่งการเข้ารหัสข้อมูลของโปรแกรมนี้ใช้อัลกอริทึมของการเข้ารหัสแข็งแกร่ง คือ RSA อัลกอริทึม ซึ่งใช้สำหรับพิสูจน์สิทธิ์ในการแลกเปลี่ยนคีย์ โดยการเข้ารหัส session key และ ใช้ 3DES ,DES , IDEA ฯลฯ อันใดอันหนึ่ง ในการเข้ารหัสข้อมูลที่ติดต่อกัน สำหรับการเข้ารหัสที่กล่าวมานี้เป็นอัลกอริทึมที่มีความเชื่อถือในปัจจุบัน ซึ่งยากในการอ่านข้อมูลหากไม่มีคีย์ที่ถูกต้อง

7.2 บทสรุปและวิเคราะห์ของโปรแกรมที่สร้างขึ้น

สำหรับโปรแกรมที่สร้างขึ้นนี้ เป็นโปรแกรมในส่วนของไคลเอนต์เพื่อติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell (เวอร์ชันที่ต่ำกว่า 2.00) ซึ่งโปรแกรมที่สร้างขึ้นมีข้อจำกัดที่ควรจะพัฒนา ดังนี้

- โปรแกรมนี้ควรพัฒนาให้สามารถรองรับวิธีการพิสูจน์สิทธิ์และอัลกอริทึมที่ใช้ในการเข้ารหัสข้อมูลหลายแบบมากขึ้น ตามที่เซิร์ฟเวอร์สามารถรองรับได้
- อัลกอริทึมที่เขียนขึ้นสำหรับการเข้าและถอดรหัสข้อมูล ควรให้มีความรวดเร็วมากขึ้น

7.3 แนวทางพัฒนาต่อในอนาคต

สำหรับมาตรฐาน Secure Shell มีแนวทางที่พัฒนาและประยุกต์ ดังนี้

- มาตรฐาน Secure Shell ได้นำเอาอัลกอริทึมต่างๆ มาประยุกต์ใช้เพื่อรักษาข้อมูลให้มีความปลอดภัยในการติดต่อแบบรีโมตล็อคอิน ซึ่งในการใช้งานผ่านเครือข่ายนั้นมีการติดต่อหลายแบบ เช่น การส่งไฟล์ (ftp) ซึ่งควรมีการพัฒนาและประยุกต์ใช้ให้เหมาะสมต่อไป
- อัลกอริทึมในการเข้ารหัสข้อมูลนั้น หากใช้อยู่แต่แบบเดิมก็มีโอกาสที่จะถูกอ่านข้อมูลได้ จึงควรมีการพัฒนาโดยมีการใช้อัลกอริทึมแบบใหม่ๆ อยู่เสมอ
- ควรพัฒนาโปรแกรมให้ใช้กับภาษาไทยได้ เพื่อให้มีความสะดวกในการใช้งานของคนไทย

ภาคผนวก ก

รายละเอียดของแพคเก็ต Secure Shell

SSH Packet Description

ในการติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ในแต่ละแพคเก็ตที่ติดต่อกันจะมีชนิดที่แตกต่างกัน ขึ้นอยู่กับข้อมูลที่จะส่ง ซึ่งแพคเก็ตแต่ละชนิดจะมีเบอร์เฉพาะ เพื่อให้ทั้งเซิร์ฟเวอร์และไคลเอนต์เข้าใจตรงกัน

สำหรับข้อความที่บ่งบอกชนิดของแพคเก็ตในแต่ละเบอร์นี้ ข้อความที่มี _MSG_ ในชื่อเป็นข้อความที่ทั้งสองฝั่ง (ไคลเอนต์และเซิร์ฟเวอร์) สามารถส่งได้ ข้อความที่มี _CMSG_ หมายถึงไคลเอนต์เท่านั้นที่ส่งแพคเก็ตชนิดนี้ได้ และข้อความที่มี _SMSG_ หมายถึงเซิร์ฟเวอร์เท่านั้นที่ส่งแพคเก็ตชนิดนี้

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

CODE ของ VT100 Virtual Terminal

ANSI/VT100 Terminal Control

คอมพิวเตอร์เทอร์มินอลและโปรแกรมจำลองเทอร์มินอลหลายๆ ชนิดรองรับการควบคุมสีและเคอร์เซอร์ของระบบโดยการใช้ตัวอักษรที่เรียกว่า " escape character " ซึ่ง VT100 ก็เป็นมาตรฐานหนึ่งที่ใช้ตัวอักษรชนิดนี้ในการควบคุม

ในภาคผนวกนี้เป็นส่วนหนึ่งของกลุ่มควบคุมของ VT100 โดย <ESC> หมายถึง " escape character " ซึ่งมีค่าเป็น 0x1B ส่วนคำในปีกกา ({ }) คือ พารามิเตอร์ที่ใช้กำหนดตามความหมายของพารามิเตอร์ภายในปีกกานั้น เช่น ROW หมายถึง หมายเลขแถว

ในการตีความหมาย เซิร์ฟเวอร์จะส่ง <ESC> ตามด้วยคำสั่งควบคุม เพื่อควบคุมลักษณะการแสดงผลของข้อความที่ตามมา

Device Status

The following codes are used for reporting terminal/display setting , and vary depending on the implementation :

Query Device Code <ESC>[c

Requests a Report Device Code response from the device.

Report Device Code <ESC>[{code}0c

Generated by the device in response to Query Device Code request.

Query Device Status <ESC>[5n

Requests a Report Device Status response from the device.

Report Device OK <ESC>[0n

Generated by the device in response to a Query Device Status request; indicates that device is functioning correctly.

Report Device Failure <ESC>[3n

Generated by the device in response to a Query Device Status request;indicates that device is functioning improperly.

Query Cursor Position <ESC>[6n

Requests a Report Cursor Position response from the device.

Report Cursor Position <ESC>[{ROM} ;{COLUMN}R

Generated by the device in response to a Query Cursor Position request;

Reports current cursor position.

Terminal Setup

The h and l codes are used for setting terminal/display mode,and vary depending on the implementation. Line Wrap is one of the few setup codes that tend to be used consistenly:

Reset Device <ESC>c

Reset all terminal setting to default.

Enable Line Wrap <ESC>[7h

Text wraps to next line if longer than the length of the display area.

Enable Line Wrap <ESC>[7l

Disable line wrapping.

Fonts

Some terminals support multiple fonts: normal/bold, swiss/italic,etc. There are a variety of special codes for certain terminals; the following are fairly standards:

Font Set Go <ESC>(

Set default font.

Font Set G1 <ESC>)

Set alternate font.

Cursor Control

Cursor Home <ESC>[{ROW};{COLUMN}H

Sets the cursor positon where subsequent text will begin. If no row/column parameters are provided(ie.<ESC>[H), the cursor will move to the *home* position , at the upper left of the screen.

Cursor Up <ESC>[{COUNT}A

Moves the cursor up by *COUNT* rows; the default count is 1.

Cursor Down <ESC>[{COUNT}B

Moves the cursor down by *COUNT* rows; the default count is 1.

Cursor Forward <ESC>[{COUNT}C

Moves the cursor forward by *COUNT* columns; the default count is 1.

Cursor Backward <ESC>[{COUNT}D

Moves the cursor backward by *COUNT* columns; the default count is 1.

Force Cursor Position <ESC>[{Row};{COLUMN}f

Identical to Cursor Home.

Save Cursor <ESC>[s

Save current cursor position.

Unsave Cursor <ESC>[u

Restores cursor position after a Save Cursor.

Save Cursor & Attrs <ESC>7

Save current cursor position.

Restore Cursor & Attrs <ESC>8

Restores cursor position after a Save Cursor.

Scrolling

Scroll Screen <ESC>[r

Enable scrolling for entire display.

Scroll Screen <ESC>[{start};{end}r

Enable scrolling from row {start} to row {end}.

Scroll Downn <ESC>D

Scroll display down one line.

Scroll Up <ESC>M

Scroll display up one line.

Tab Control

Set Tab <ESC>H

Sets a tab at the current position.

Clear Tab <ESC>[g

Clears tab at the current position.

Clear All Tabs <ESC>[3g

Clears all tabs.

Erasing Text

Erase End of Line <ESC>[K

Erases from the current cursor position to the end of the current line.

Erase Start of Line <ESC>[1K

Erases from the current cursor position to the start of the current line.

Erase Line <ESC>[2K

Erase the entire current line.

Erase Down <ESC>[J

Erase the screen from the current line down to the bottom of the screen.

Erase Up <ESC>[1J

Erase the screen from the current line up to the top of the screen.

Erase Screen <ESC>[2J

Erase the screen with the background color and moves the cursor to *home*.

Printing

Some terminals support local printing:

Print Screen <ESC>[i

Print the current screen.

Print Line <ESC>[1i

Print the current line.

Stop Print Log <ESC>[4i

Disable log.

Start Print Log <ESC>[5I

Start log; all received text is echoed to a printer.

Define Key

Set Key Definition <ESC>[{key};"{string}"p

Associates a *string* of text to a keyboard key.{key} indicates the keys by its ASCII value in decimal.

Set Display Attributes

Set Attribute Mode <ESC>[{attr1};...;{attrn}m

Sets multiple display attribute settings. The following lists standard attributes:

0	Reset all attributes
1	Bright
2	Dim
3	Underscore
4	Blink
5	Reverse
6	Hidden

Foreground Colors

30	Black
31	Red
32	Green
33	Yellow
34	Blue
35	Magenta
36	Cyan
37	White

Background Colors

40	Black
41	Red
42	Green
43	Yellow
44	Blue
45	Magenta
46	Cyan
47	White

ภาคผนวก ค

NVT CODE และ คำสั่งของ Telnet

NVT CODE

Name	Code	Meaning
NULL	0	No operation
Line Feed(LF)	10	Moves theprinter to the next line keeping the jsame horizontal postion.
Carriage Return (CR)	13	Movesthe printer to the left margin of the current line.
Bell (BELL)	7	Produces a sound or a visible signal.
Back Space (BS)	8	Moves one character position back toward the left.
Horizontal Tab (HT)	9	Moves to the next vertical tab position. How vertical tab positions are determined is not specified.
Vertical Tab (VT)	11	Moves to the next vertical tab position. How vertical tab positions are determined is not specified.
Form Feed (FF)	12	Moves to the top of next page, keeping the same horizontal position.

TELNET Commands

Command	Code	Function Requested
Interrupt Process (IP)	244	Terminate the current process.
About Output (AO)	245	Stop sending output for a process but allow the process to run to completion.
Are You There (AYT)	246	Requests a response from the server that the server is still active.
Erase Character (EC)	247	Delete the last character of output.
Erase Line	248	Delete the current line of output.
Break	243	NVT character BRK.

บรรณานุกรม

1. Network Working Group , Request for comments, " The SSH (Secure Shell) Remote Login Protocol. ", Category : Experimental , T.Ylonen ,Helsinki University of Technology , 25 July 1995.

2. "Wireless Communication." วารสารทางวิชาการสื่อสารโทรคมนาคม. ปีที่ 2 ฉบับที่ 6 เดือนตุลาคม 2537 , ดร.พิสิษฐ์ ชาญเกียรติก้อง

3. "SSH Communications Security", Espoo , Finland , Copyright □ 1996 ,1997 ,1998

4. " การเขียนโปรแกรม Visual C++ " เวอร์ชัน 1.5 บนวินโดว์ เรียบเรียงโดย โชคชัย เตชพรรุ่ง

5. http://www.ssh.net/

6. http://www.uni-kar/sruhe.de/

7. http://rsa.com/

8. http://www.cs.hut.fi/crypto/algorithms.html

9. http://www.ssh.fi

10. http://www.whatis.com/des.html

11. http://www.interhack.net/pubs/des_key_crack