โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา

Secure UNIX Terminal Emulator Using Java

นายเชาวนิน ไสยสมบัติ

นายสุชาติ คุ้มมะณี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา

Secure UNIX Terminal Emulator Using Java

โดย

นายเชาวนิน ไสยสมบัติ

นายสุชาติ คุ้มมะณี

อาจารย์ที่ปรึกษา

อ.ธนา หงษ์สุวรรณ

อ.อัครเดช วัชระภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา

Secure UNIX Terminal Emulator Using Java

ผู้จัดทำ

1. นายเชาวนิน ไสยสมบัติ รหัสประจำตัว 40013248
2. นายสุชาติ คุ้มมะณี รหัสประจำตัว 40013275

_____ อาจารย์ที่ปรึกษา

(อาจารย์ธนา หงษ์สุวรรณ)

_____ อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชระภูพงษ์)

โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา

นายเชาวนิน ไสยสมบัติ
นายสุชาติ คุ้มมะณี
อ.ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อ.อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

ในปัจจุบันเมื่อผู้ใช้ต้องการเข้าใช้งานเครื่องคอมพิวเตอร์ใดๆในระบบยูนิกซ์ แอพพลิเคชันที่เราใช้งานทั่วๆไปเช่น เทลเน็ต (Telnet) จะไม่มีกรรมวิธีทำให้ข้อมูลที่ใช้เหล่านั้นปลอดภัยเนื่องจากจะส่งข้อมูลในลักษณะที่ไม่มีการเข้ารหัส ดังนั้นปริญญานิพนธ์ฉบับนี้ จึงเสนอวิธีการพัฒนาแอพพลิเคชันในรูปแบบเดียวกับเทลเน็ต แต่จะมีกลยุทธ์ที่ทำให้ข้อมูลเหล่านั้นปลอดภัยมากที่สุดโดยใช้ โพรโตคอล SSH (Secure Shell) ที่รองรับการเข้าและถอดรหัสหลากหลายประเภท เช่น RSA, DES, 3DES, IDEA และ MD5 ทำให้มีเสถียรภาพในการใช้งานเป็นอย่างมาก โพรโตคอล Secure Shell ที่นำมาใช้คือเวอร์ชัน 1.5 พัฒนาแอพพลิเคชันในฝั่งไคลเอ็นต์ขึ้นด้วยภาษาจาวาเวอร์ชัน 1.1.8 ซึ่งมีโครงสร้างเป็นออบเจ็กต์-โอเรียนเต็ดทำให้พัฒนาแอพพลิเคชันได้ง่าย อีกทั้งแอพพลิเคชันที่พัฒนาขึ้นยังสามารถนำไปใช้งานได้กับระบบปฏิบัติการที่มีจาวาเวอร์ชวลแมชชีน โดยไม่ต้องคอมไพล์หรือเขียนโค้ดขึ้นมาใหม่ และเพื่อให้การใช้งานเหมาะกับคนไทยจึงได้พัฒนาให้ทำงานร่วมกันได้กับภาษาไทย

Secure UNIX Terminal Emulator Using Java

Chaowanin Saiyasombat
Suchart Khummanee
Thana Hongsuwan Advisor
Akkradach Watcharapupong Advisor

ABSTRACT

Nowadays, when users want to use the applications in UNIX systems such as Telnet. There is no security to protect their data, because they don't use data encryption. So this thesis present the development of one application like Telnet but use SSH (Secure Shell) protocol to make it more safety. The SSH protocol supports many kinds of encryption-decryption technology such as RSA, DES, 3DES, IDEA, and MD5. We use SSH version 1.5 and develop the client application by using JAVA version 1.1.8 which supports the object-oriented architecture, that make us easier to develop the application. The JAVA using makes our application to be compatible with many operating systems that has Java Virtual Machine. Furthermore, our application supports Thai language in order to supports Thai using.

กิตติกรรมประกาศ

โครงงานและปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ได้ เนื่องจากคำแนะนำจากอาจารย์ที่ปรึกษาทั้ง 2 ท่านคือ อาจารย์ธนา หงษ์สุวรรณ และอาจารย์อัครเดช วัชระภูพงษ์ ที่คอยแนะนำ เป็นที่ปรึกษา และเอาใจใส่กับการทำโครงงานนี้เป็นอย่างดี ซึ่งทางคณะผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษาทั้งสองท่านเป็นอย่างสูง

นอกเหนือจากนี้ก็ต้องขอขอบพระคุณคณะอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอย่างยิ่งที่ได้ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ อีกทั้งภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้เอื้อเฟื้ออุปกรณ์ทรัพยากร สถานที่และอำนวยความสะดวกต่างๆในการทำโครงงานนี้ด้วย

ขอขอบใจเพื่อนๆชาว ISAG ทุกคนที่ช่วยเหลือในการทำงานและแก้ไขอุปสรรคต่างๆในการทำงานให้ผ่านพ้นไปด้วยดี เป็นที่ปรึกษาและกำลังใจที่ดีเสมอมา

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

เชาวนิน ไสยสมบัติ
สุชาติ คุ้มมะณี

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ที่มาของโครงงาน

ในปัจจุบันการเข้าใช้งานกับเครื่องเซิร์ฟเวอร์จากระยะไกลผ่านทางเครือข่าย หรือที่เรียกว่ารีโมตล็อกอิน (remote login) นั้นมีการใช้งานกันอย่างแพร่หลาย มีโปรแกรมต่างๆ ที่ใช้งานกันมากเช่น โปรแกรมเทลเน็ต (telnet) แต่เนื่องจากการส่งข้อมูลในโปรแกรมดังกล่าวเป็นข้อมูลชนิดแพลนเท็กซ์ (Plain text) ที่ไม่มีการเข้ารหัสข้อมูลทำให้ไม่มีความปลอดภัยในการเข้าใช้งาน เนื่องจากอาจโดนผู้ที่ไม่ประสงค์ดีลักลอบดักหรือปลอมแปลงข้อมูลของเราได้ ดังนั้นเพื่อเป็นการป้องกันพฤติกรรมดังกล่าว จึงต้องมีการเข้ารหัสข้อมูลที่ส่ง ในที่นี้ใช้โพรโตคอล SSH ที่ได้มีการกำหนดเป็นมาตรฐานใช้งานกันอย่างแพร่หลาย เพราะมีวิธีการพิสูจน์สิทธิ์ และมีวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพก่อนติดต่อและส่งข้อมูล แม้ว่าผู้ที่ไม่ประสงค์ดีได้รับข้อมูลเหล่านั้นไปก็ไม่อาจทราบได้ว่าเป็นข้อมูลอะไร

อย่างไรก็ดีในปัจจุบันระบบปฏิบัติการที่ใช้กับโปรแกรมติดต่อกับเซิร์ฟเวอร์มีอยู่หลากหลาย และยังมีฮาร์ดแวร์มากมายหลายรุ่นที่ใช้กับระบบปฏิบัติการเหล่านั้น ชุดคำสั่งที่ทำงานก็จะแตกต่างกันไปเช่นกัน ทำให้โปรแกรมที่เขียนในระบบหนึ่งไม่สามารถใช้งานได้กับอีกระบบ ต้องมีการแก้ไขโปรแกรมและคอมไพล์ใหม่เป็นการยุ่งยาก แต่ในปัจจุบันได้มีภาษาจาวา ที่เป็นภาษาที่ไม่ขึ้นกับแพลตฟอร์ม (Platform) หรือว่าระบบปฏิบัติการใดๆ ทำให้เมื่อโปรแกรมสร้างเสร็จแล้วสามารถนำไปทำงานกับระบบอื่นๆ ได้อย่างอิสระ

1.2 วัตถุประสงค์

1. เพื่อสร้างโปรแกรมต้นแบบเพื่อจำลองเทอร์มินอลสำหรับผู้เข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัย
2. เพื่อพัฒนาโปรแกรมบนแพลตฟอร์มของจาวา
3. เพื่อพัฒนาโปรแกรมจำลองเทอร์มินอลที่ใช้งานร่วมกับภาษาไทยได้

1.3 ขอบเขตของโครงงาน

1. สร้างโปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์ที่ใช้โพรโตคอล Secure Shell
2. สร้างโปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์ที่ใช้งานร่วมกับภาษาไทยได้โดยไม่ขึ้นกับระบบปฏิบัติการ
3. พัฒนาการเขียนโปรแกรมบนแพลตฟอร์มของจาวาเพื่อให้โปรแกรมที่พัฒนาสามารถใช้งานได้ในทุกระบบปฏิบัติการ

1.4 ผลที่คาดว่าจะได้รับ

1. โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา (ต้นแบบ)
2. โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์ที่ทำงานร่วมกับภาษาไทยได้
3. เอกสารประกอบเรื่องโพรโตคอล SSH โดยละเอียด

บทที่ 2

การเข้ารหัสและการคำนวณ

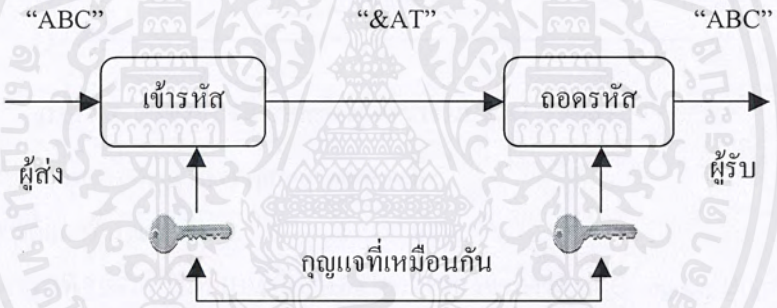
2.1 ระบบของการเข้ารหัสข้อมูล

การเข้ารหัสข้อมูลเป็นการทำให้ข้อมูลที่ต้องการเป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูลปัจจุบัน โดยอาศัยหลักการของการเข้ารหัส และการถอดรหัส

- การเข้ารหัสเป็นการเปลี่ยนรูปข้อมูล โดยผ่านรูปแบบและกระบวนการแปรรูปข้อมูล ทำให้ข้อมูลที่ส่งมีรูปแบบไม่เหมือนเดิม เพื่อทำให้ข้อมูลเป็นความลับ
- การถอดรหัสเป็นการแปลงข้อมูลที่ได้เข้ารหัสให้กลับมาเป็นข้อมูลเหมือนเดิม ซึ่งการเข้ารหัสและการถอดรหัสจะควบคุมโดยกุญแจหรือที่เรียกว่า “Key”

2.1.1 ระบบการเข้ารหัสข้อมูลโดยใช้กุญแจเดียว

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



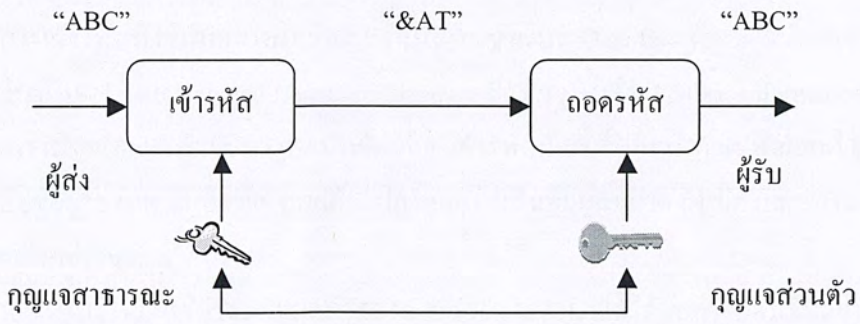
รูปที่ 2.1 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว

2.1.2 ระบบการเข้ารหัสแบบกุญแจสาธารณะ

การเข้ารหัสข้อมูลระบบนี้ประกอบด้วยกุญแจ 2 อันคือ

1. **กุญแจส่วนตัว (Private Key)** เป็นกุญแจที่ต้องเก็บเป็นความลับ
2. **กุญแจสาธารณะ (Public Key)** เป็นกุญแจที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งทั้งสองกุญแจนี้เป็นกุญแจที่ต่างกัน กรรมวิธีในการเข้าและถอดรหัสจะเป็นดังรูป

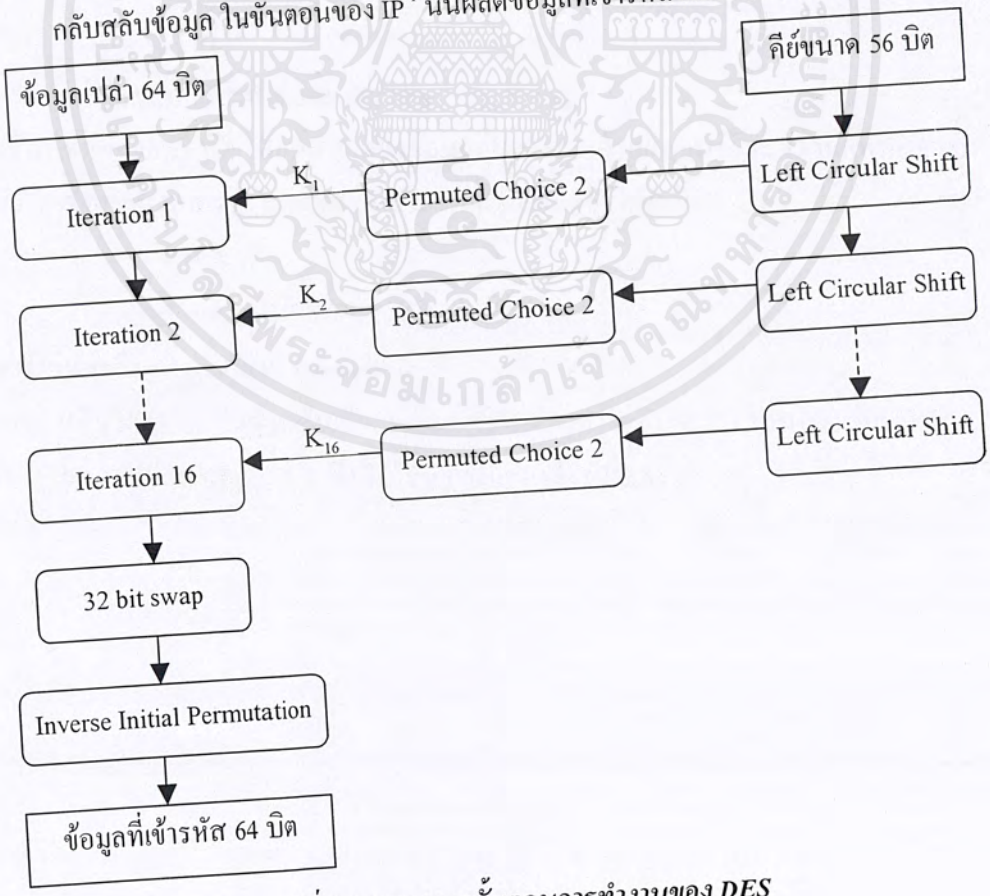


รูปที่ 2.2 แสดงการเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ

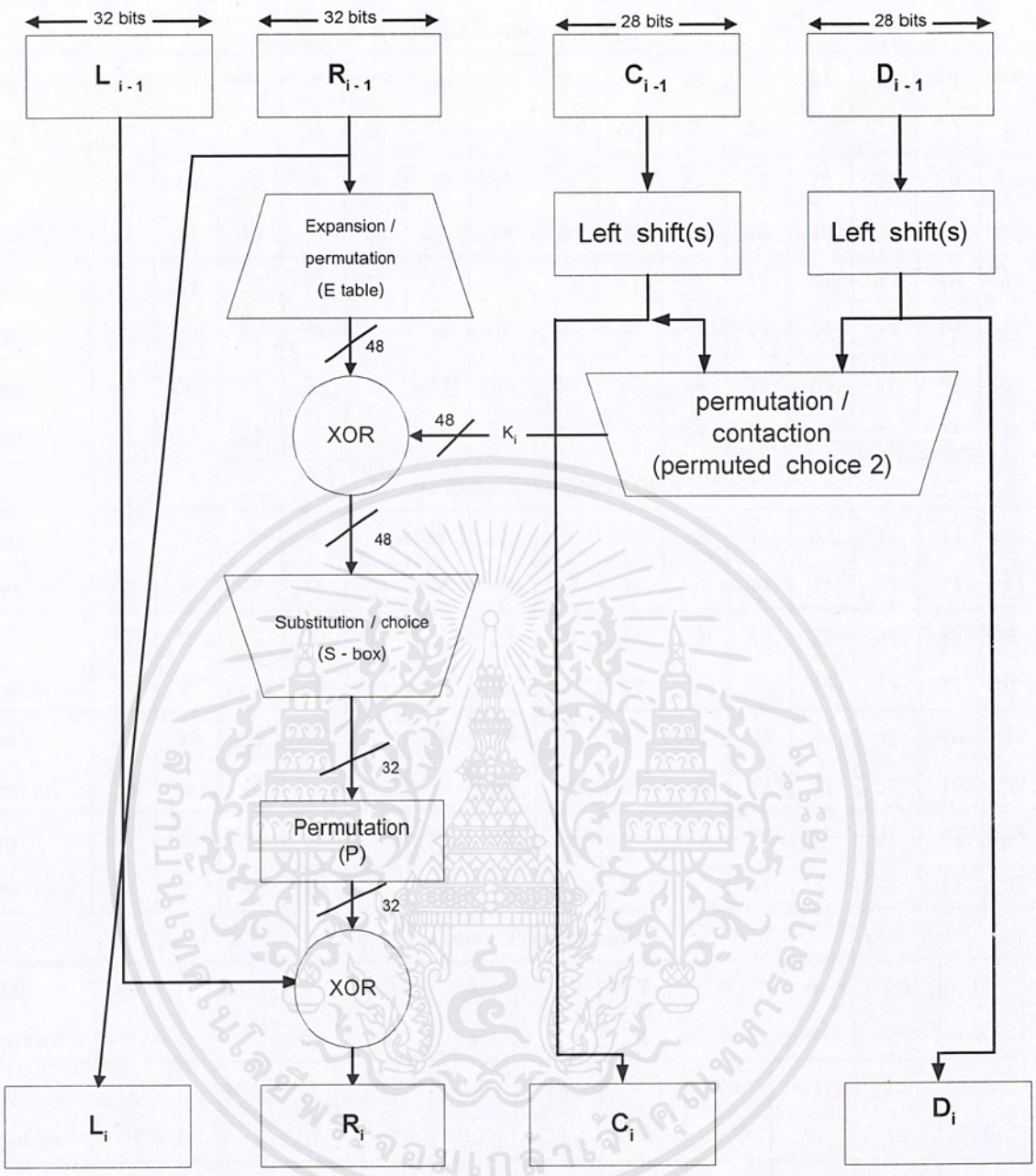
หรือไม่และมีความปลอดภัยมากน้อยแค่ไหน แต่จนถึงปัจจุบันเราก็ยังไม่พบช่องโหว่ของ DES ตามเอกสารที่ตีพิมพ์เป็นสาธารณะ แม้ว่าใช้คีย์เพียงไม่กี่บิตก็ตาม ในทางตรงกันข้ามแนวความคิดแบบ IDEA กลับใช้คีย์แบบ 128 บิต และได้รับการต้อนรับจากสาธารณะตั้งแต่ทศวรรษที่ 90 (ค.ศ. 1980-1990) IDEA มีความปลอดภัยมากกว่า DES และสามารถประมวลผลได้เร็วกว่า DES แต่อย่างไรก็ตาม IDEA ยังต้องรอการตรวจสอบจากผู้เชี่ยวชาญอีกมากถึงเรื่องช่องโหว่ของความปลอดภัย

การทำงานของ DES จะมีลักษณะดังรูป ข้อมูลที่เข้ามาในส่วนของฟังก์ชันของการเข้ารหัสมี 2 ส่วนด้วยกันคือ ข้อมูลที่ยังไม่ถูกเข้ารหัสขนาด 64 บิตและคีย์ขนาด 56 บิต ในรูปด้านซ้ายมือแสดงขั้นตอนจัดการกับข้อมูลที่ยังไม่เข้ารหัส โดยแบ่งย่อยได้เป็น 3 ขั้นตอนคือ

- ขั้นตอนที่ 1 จัดการกับข้อมูลที่ยังไม่ได้เข้ารหัสขนาด 64 บิตให้ผ่านไปยังส่วนที่เรียกว่า Initial Permutation (IP) เพื่อสลับและเรียบเรียงบิตข้อมูลใหม่เรียกว่า “Permuted Input”
- ขั้นตอนที่ 2 นำข้อมูลที่ต้องการเข้ารหัสและคีย์ที่ใช้ในการเข้ารหัส มากระทำฟังก์ชันเดียวกัน 16 ครั้งโดยรวมทั้งการทำ permutation และ substitution ซึ่งผลลัพธ์ที่ได้จากการกระทำทั้ง 16 ครั้งนั้นจะได้ข้อมูลขนาด 64 บิตที่เรียกว่า “Preoutput” ข้อมูลที่ต้องการเข้ารหัสนั้นแบ่งออกเป็นสองส่วนก่อนเข้าไปยังกระบวนการ
- ขั้นตอนที่ 3 นำ Preoutput ไปยังส่วนที่เรียกว่า Inverse Initial Permutation (IP^{-1}) ซึ่งทำหน้าที่กลับสลับข้อมูล ในขั้นตอนของ IP^{-1} นั้นผลิตข้อมูลที่เข้ารหัสขนาด 64 บิตเรียกว่า Ciphertext



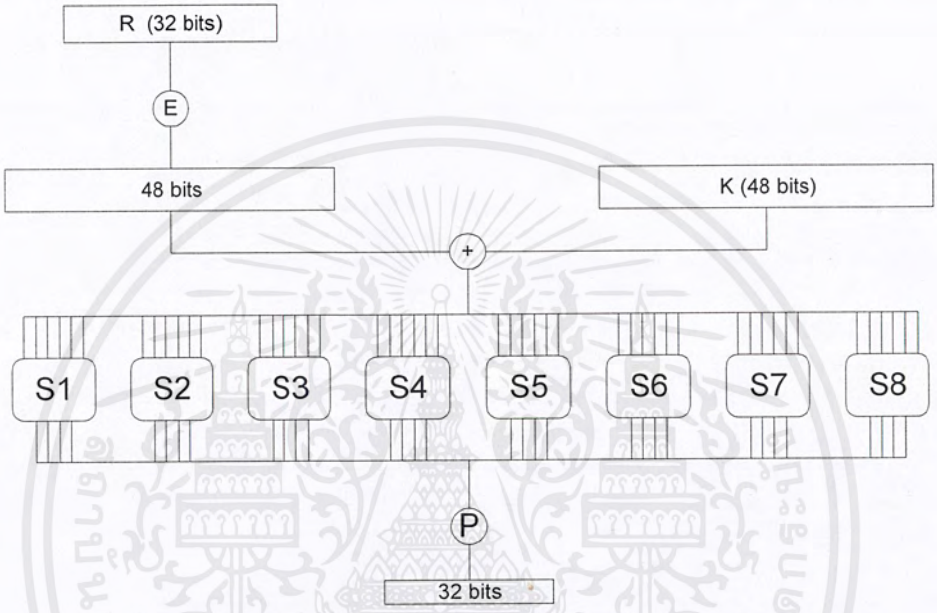
รูปที่ 2.4 แสดงการขั้นตอนการทำงานของ DES



รูปที่ 2.5 แสดงขั้นตอนการทำฟังก์ชันในแต่ละครั้งของการเข้ารหัส DES (กระทำทั้งหมด 16 ครั้ง)

จากรูปที่ 2.5 คีย์ K_i ที่ใช้ในแต่ละรอบมีขนาด 48 บิต ส่วนข้อมูลฝั่งขวามีขนาด 32 บิต ดังนั้นจึงต้องมีการขยายขนาดข้อมูลจาก 32 บิตให้เป็น 48 บิต โดยใช้ตารางที่ 2.1(c) ที่เป็นวิธีการกำหนด Expansion Permutation ซึ่งจำลองบิตเพิ่มขึ้นมาอีก 16 บิต แล้วนำผลที่ได้ทั้งหมด 48 บิตทำการ XOR กับ K_i และผลลัพธ์ที่ได้จะส่งไปยัง Substitution และ Permutation ที่ให้ข้อมูลเป็น 32 บิต

โดย Substitution ประกอบด้วยเซตของ S-box 8 ชุด ($S_1 - S_8$) ซึ่ง S-box มีอินพุตขนาด 6 บิตและเอาต์พุตขนาด 4 บิต ซึ่งตารางที่ 2.2 แสดง DES S-box กรรมวิธีในการแปลงอินพุตขนาด 6 บิตให้กลายเป็นเอาต์พุตขนาด 4 บิตมีดังนี้คือ นำบิตแรกและบิตสุดท้ายของอินพุตมาทำเป็นตำแหน่งแถวและนำ 4 บิตกลางมาเป็นตำแหน่งคอลัมน์ เช่น S_1 มีค่าเท่ากับ 011011(ขนาด 6 บิต) เรานำบิตแรกและบิตสุดท้ายซึ่งก็คือ 0 และ 1 มาเป็นตำแหน่งของแถวจะได้แถวที่ 01 หรือถือแถว 1และ 4 บิตตรงกลางที่เหลือคือ1101 จะได้ตำแหน่งของคอลัมน์คือ คอลัมน์ที่ 13 ดังนั้นค่าที่ตำแหน่งแถวที่ 1 และคอลัมน์ที่ 13 ในตารางคือ 0101



รูปที่ 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	

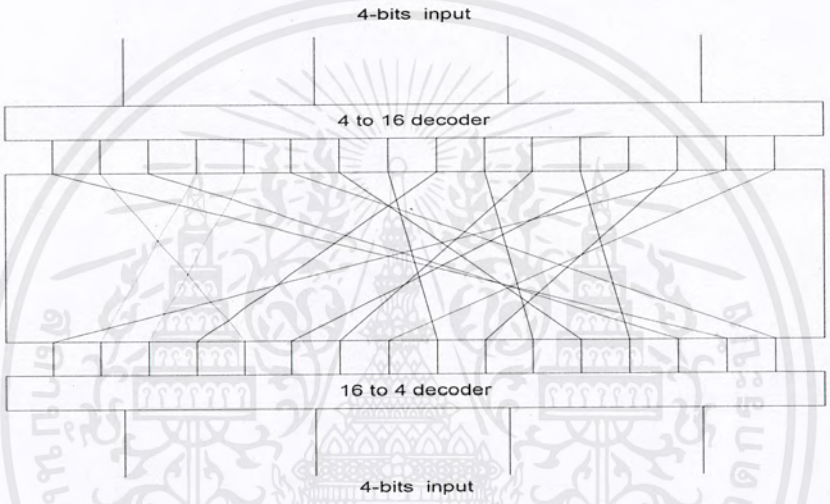
Column Number

Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Box
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	S4
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	

ตารางที่ 2.2 แสดงขั้นตอนใน S-box ทั้ง 8 ชุด

2.2.5 การสร้างคีย์ (Key Generation)

ในรูปที่ 2.5 ที่แสดงการทำฟังก์ชันในแต่ละรอบของ DES นั้น เราจะเห็นว่าคีย์ที่ใช้มีขนาด 56 บิตใช้เป็นอินพุตในการทำ Pernutation ดังในตารางที่ 2.3(a) Permute Chioce One โดยเริ่มแรกจะทำการแบ่งคีย์ออกเป็น 2 ส่วน ส่วนละ 28 บิตซึ่งเรียกว่าส่วน C และ D ในการทำฟังก์ชันแต่ละรอบ (ทั้งหมด 16 รอบ) จะมีการทำ Circular left shift หรือทำ Rotation ทั้งสองส่วน ในแต่ละรอบมีการกำหนดว่าให้เลื่อนไปกี่บิต ดังตารางที่ 2.3(c) ค่าที่ถูกเลื่อนบิตจะเป็นอินพุตของการทำฟังก์ชันรอบถัดไปและเป็นอินพุตในการทำ Permutation Choice Two ดังในตารางที่ 2.5(b) ซึ่งได้เอาต์พุตขนาด 48 บิต ใช้เป็นอินพุตของฟังก์ชันของ $f(R_{i-1}, K_i)$



รูปที่ 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

ตารางที่ 2.3 แสดงการสร้างคีย์ในแต่ละรอบฟังก์ชัน

2.2.6 การถอดรหัสข้อมูล DES

กระบวนการถอดรหัสข้อมูลโดยใช้ DES นั้นเหมือนกับขั้นตอนการเข้ารหัส ซึ่งมีขั้นตอนการทำงานดังนี้คือ นำข้อมูลผ่านการเข้ารหัสแล้วมาเป็นอินพุต ส่วนคีย์ที่ใช้ในการถอดรหัส (K_s) จะมีลำดับย้อนกลับกับคีย์ที่ใช้ในการเข้ารหัสคือ $K_{16}, K_{15}, K_{14} \ldots K_1$ เป็นคีย์ที่ใช้ในการถอดรหัส ดังรูปที่ 2.8(a) เป็นขั้นตอนการเข้ารหัสและรูป 2.8(b) เป็นขั้นตอนในการถอดรหัส

ต่อไปแสดงถึงผลลัพธ์ของขั้นตอนแรกในการกระบวนการถอดรหัส ซึ่งเอาต์พุตเท่ากับ 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

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

$$D \oplus D = 0$$

$$E \oplus 0 = E$$

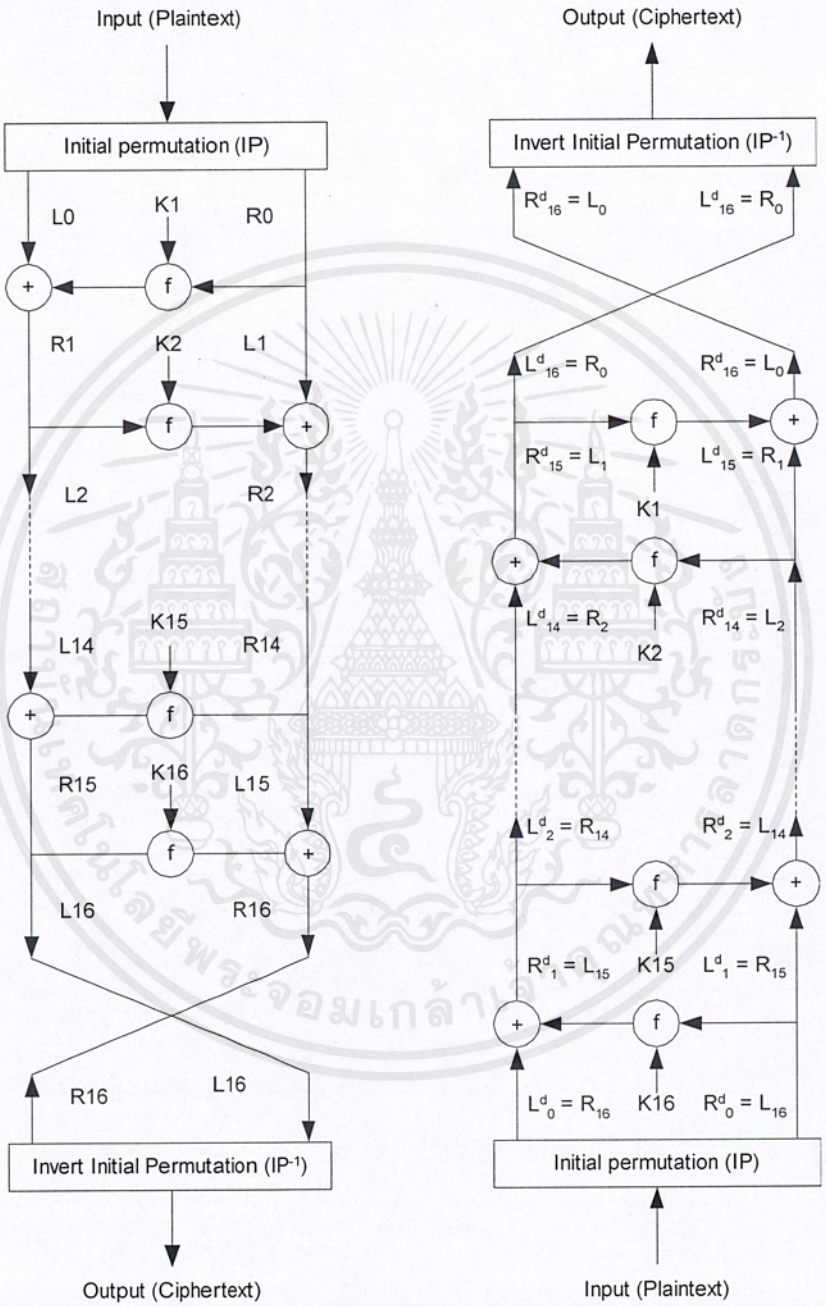
ดังนั้นเรามี Ld1 = R15 และ Rd1 = L15 จะได้เอาต์พุตของขั้นตอนแรกในการถอดรหัสคือ L15||R15 ซึ่งเราสามารถเขียนเป็นสมการในการถอดรหัสได้ดังนี้คือ

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

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

ซึ่งผลสุดท้ายเอาต์พุตที่ได้จากขั้นตอนสุดท้ายในการถอดรหัสคือ R0||L0 และนำไปสู่ขั้นตอนในการทำ Inverse Permutation จะได้ข้อมูลที่ส่งมา(plaintext) ดังสมการ

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

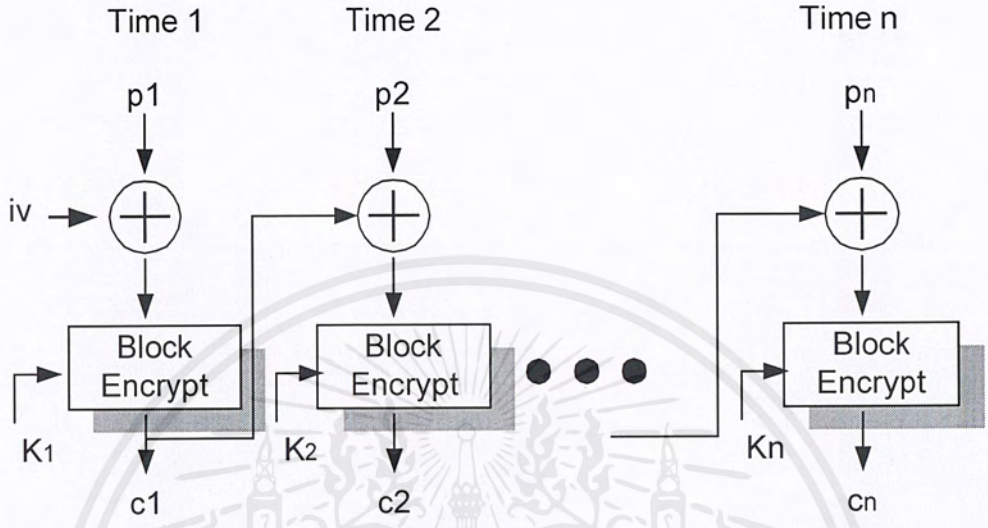


(a) ขั้นตอนการเข้ารหัส *(b) ขั้นตอนการถอดรหัส*

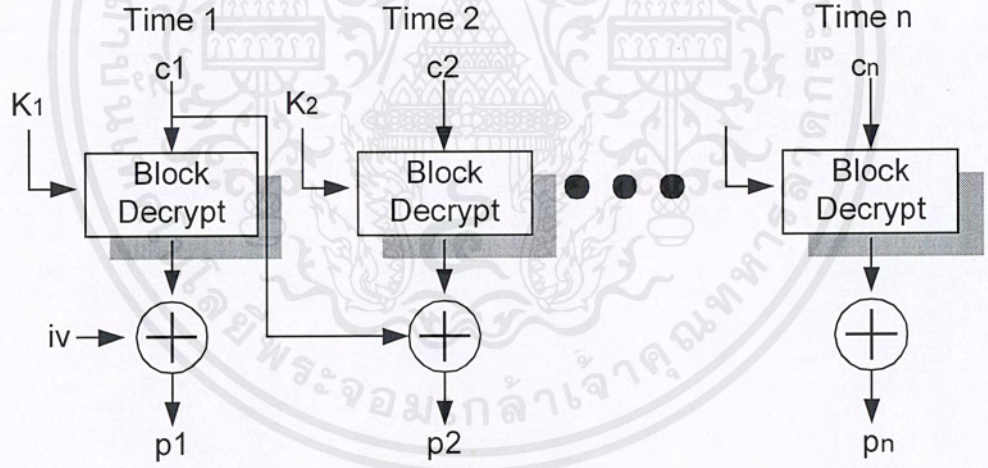
รูปที่ 2.8 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES

2.2.7 โหมด CBC (Chiper Block Chainning)

เป็นการเข้ารหัสที่พัฒนามาจาก DES ช่วยให้ข้อมูลที่ส่งทีความปลอดภัยยิ่งขึ้น โดยมีวิธีการคือ นำข้อมูลที่ผ่านการเข้ารหัสของข้อมูลตัวก่อนมา XOR กับข้อมูลที่ยังไม่ได้เข้ารหัสของตัวถัดไปก่อนจะทำการเข้ารหัสแบบ DES ตามปกติดังรูป 2.9



(a) การเข้ารหัส DES CBC



(b) การถอดรหัส DES CBC

รูปที่ 2.9 แสดงการเข้ารหัสและถอดรหัสของ DES CBC

ในการถอดรหัสข้อมูล จะทำเช่นเดียวกับการถอดรหัสข้อมูล DES ตามปกติแต่นำผลที่ได้จากการทำถอดรหัสมา XOR กับข้อมูลที่ผ่านการเข้ารหัส(Ciphertext)ตัวก่อนหน้านี้ เพื่อจะได้ข้อมูลจริงๆ(plaintext)ดังสมการ

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

สมการในการถอดรหัส

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

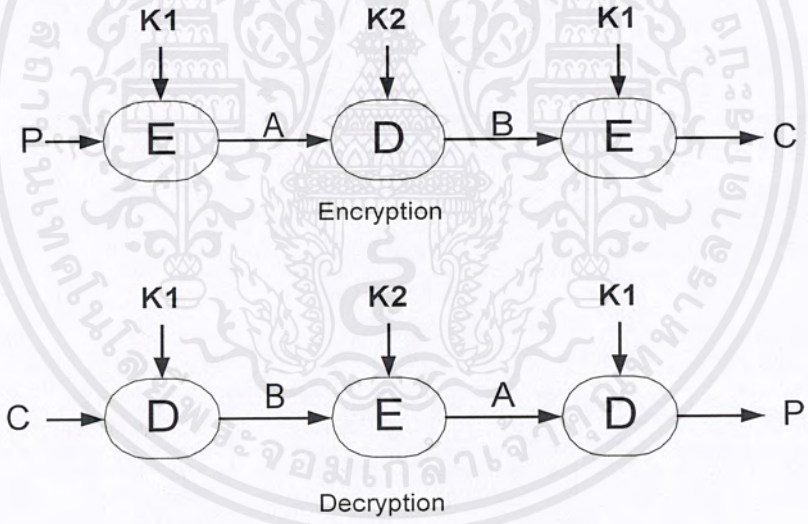
$$Dk[Cn] = Cn^{-1} \oplus Pn$$

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

สังเกตเห็นได้ว่าบล็อกแรกของข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) จะมีการนำเอาค่า iv (Initialization Vector) มาทำการ XOR กับข้อมูลที่ไม่ได้เข้ารหัสก่อนที่จะมีการเข้ารหัส DES ตามปกติ และในส่วนของการถอดรหัสข้อมูลจะใช้ค่า iv ในการถอดรหัสข้อมูลเช่นเดียวกัน ดังนั้นจึงจำเป็นต้องมีข้อตกลงกันระหว่างผู้ส่งกับผู้รับก่อนว่าใช้ค่า iv เป็นค่าใด เพื่อให้มีความปลอดภัยสูงสุดค่า iv ควรมีการป้องกันเช่นเดียวกับ Key ในการส่งค่า iv อาจทำการส่งโดยใช้การเข้ารหัสแบบ ECB

2.3 การเข้ารหัสแบบ 3DES (Triple DES)

เป็นที่ทราบกันว่ายิ่งคีย์มีความยาวมากขึ้น การถอดรหัสยิ่งทำได้ยากยิ่งขึ้น นอกจากนี้ยังเพิ่มประสิทธิภาพของการเข้ารหัสให้มากขึ้น โดยการเข้ารหัสหลายๆครั้ง อย่างไรก็ตามการเข้ารหัสครั้งที่สองโดยใช้คีย์ที่ต่างจากครั้งแรกนั้นมิได้หมายความว่าข้อมูลจะมีความปลอดภัยเพิ่มขึ้นเป็น 2 เท่า ถ้าการเข้ารหัสดังกล่าวไม่ได้เป็นการเข้ารหัสโดยใช้เทคนิคทางคณิตศาสตร์เช่น DES และการที่จะให้ DES มีความปลอดภัยมากขึ้นเป็น 2 เท่านั้นจำเป็นที่เราต้องเข้ารหัสถึง 3 ครั้ง



รูปที่ 2.10 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES

ขั้นตอนการใช้คีย์เข้ารหัสแบบ 3 ครั้งนั้นมีดังนี้

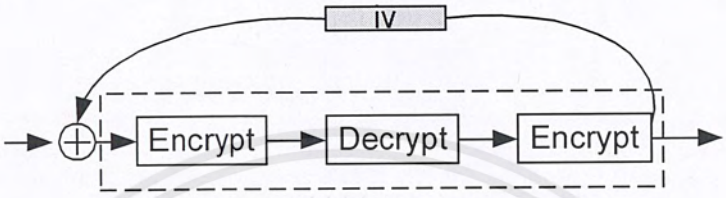
1. เข้ารหัสโดยใช้คีย์ตัวแรก
2. ถอดรหัสโดยใช้คีย์ตัวที่สอง
3. เข้ารหัสโดยใช้คีย์ตัวที่สาม

(การเข้ารหัส 3DES อาจใช้คีย์ตัวที่ 1 และ 3 เป็นตัวเดียวกันก็ได้ แต่ด้านความปลอดภัยของการเข้ารหัสจะลดลง)

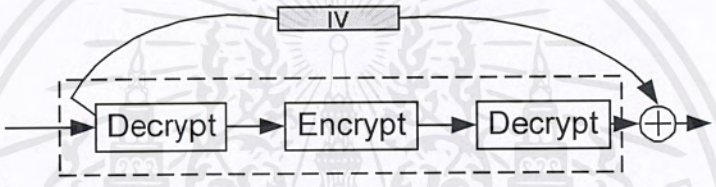
2.3.1 Triple DES โหมด CBC (3DES CBC Mode)

เนื่องจากการเข้ารหัสแบบ DES นั้นมีโหมดในการเข้ารหัสแบบ CBC ดังนั้นการเข้ารหัสแบบ 3DES ที่ได้มีการประยุกต์มาจาก DES จึงได้มีการพัฒนา 3DES นี้ให้มีโหมด CBC ด้วย ในการเข้ารหัสแบบ 3DES CBC นี้ได้พัฒนาโหมดนี้ออกเป็น 2 แบบ

- **แบบ Inner CBC** การเข้ารหัสแบบนี้มี iv เพียงตัวเดียวในการเข้ารหัสหรือถอดรหัสแบบ 3DES ในแต่ละครั้ง ดังรูป 2.11



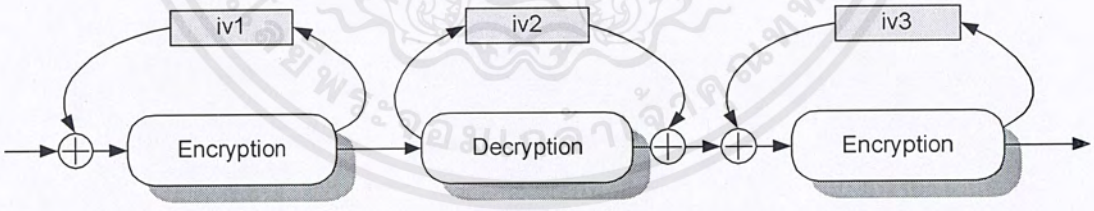
(a) การเข้ารหัส



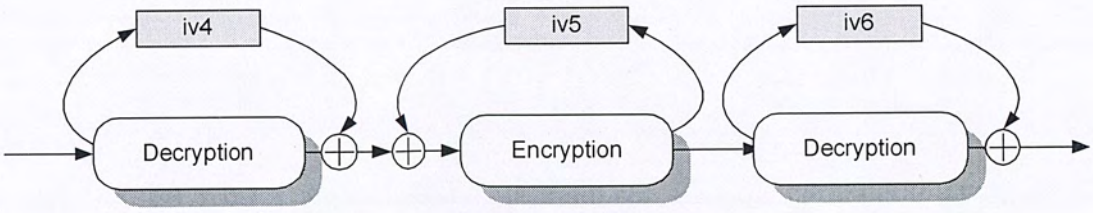
(b) การถอดรหัส

รูปที่ 2.11 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด Inner CBC

- **แบบ Outer CBC** การเข้ารหัสและถอดรหัสในแบบนี้แต่ละโมดูลจะมี iv เฉพาะตัวในแต่ละโมดูล ดังรูป 2.12



(a) การเข้ารหัส

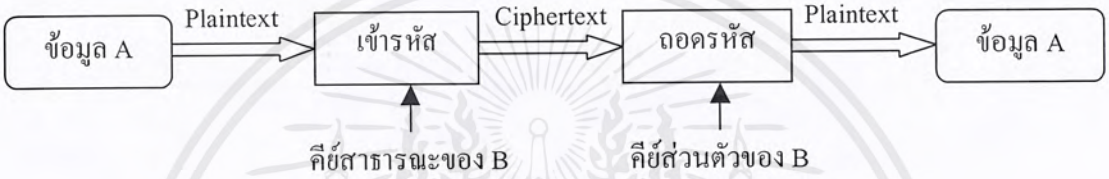


(b) การถอดรหัส

รูปที่ 2.12 แสดงการเข้ารหัสและถอดรหัสแบบ Triple DES โหมด Outer CBC

2.4 การเข้ารหัสแบบ RSA

รหัส RSA คิดค้นขึ้นในปี 1977 โดยที่ชื่อ RSA ได้มาจากอักษรตัวแรกของนามสกุลผู้ที่ร่วมกันคิดค้นคือ Ron **R**iver, Adi **S**hamir และ Leonard **A**dleman เป็นวิธีการเข้ารหัสแบบกุญแจสาธารณะที่เรียกว่าพับลิคคีย์ (Public–Key Crytosystem) เพื่อแก้ไขปัญหาการทำให้กุญแจเป็นความลับ(Secret–Key Cryptosystem) โดยสมาชิกแต่ละคนต้องมีกุญแจ 2 ชนิดคือ กุญแจส่วนตัวหรือไพรเวตคีย์ (Private Key) และกุญแจสาธารณะหรือพับลิคคีย์ (Public Key) โดยกุญแจส่วนตัวเก็บไว้เป็นความลับ ส่วนกุญแจสาธารณะนั้นจะเปิดเผยให้ใครก็ได้ที่ต้องการส่งข้อมูลให้แก่ตน หลักการทำงานของรหัสลับมีอยู่ว่าข้อมูลที่เข้ารหัสลับด้วยกุญแจสาธารณะของผู้ใด สามรถถอดรหัสได้ด้วยกุญแจส่วนตัวของผู้นั้นเท่านั้น การทำงานของระบบพับลิคคีย์สามารถอธิบายได้ดังต่อไปนี้



รูปที่ 2.13 แสดงขั้นตอนการเข้ารหัสและถอดรหัสโดยใช้คีย์ 2 ตัว

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

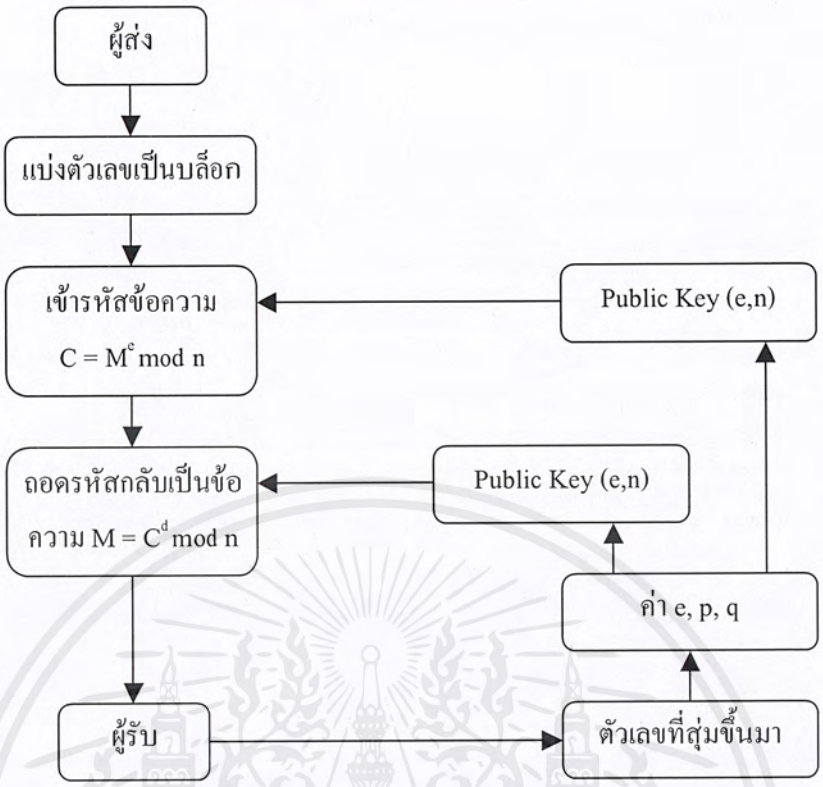
$$e*d \bmod (p-1)(q-1) = 1 \quad \text{.....(1)}$$

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

ตัวอย่างเช่น ถ้าอริสาต้องการส่งข้อเอกสารส่วนตัว m ไปให้บุญมี อริสาจะสร้างรหัสลับ c ของเอกสาร m ได้โดยการให้ $c = m^e \bmod n$ เมื่อ (n,e) เป็นพับลิคคีย์ของบุญมี เมื่อบุญมีได้รับเอกสารรหัสลับ c เขาจะถอดรหัสเพื่ออ่านเอกสาร m ได้ด้วย d เพราะความสัมพันธ์ในสมการ (1) ระหว่าง e,d และ n ทำให้

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

และเพราะมีแต่บุญมีเท่านั้นรู้ค่า d ดังนั้นบุญมีเท่านั้นจะถอดรหัสได้ คุณสมบัติสำคัญประการหนึ่งของ RSA คือจากสมการ (2) ในทางกลับกันถ้าเราเข้ารหัสด้วยไพรเวตคีย์ เราก็สามารถถอดรหัสด้วยพับลิคคีย์ได้ด้วยเช่นกัน คุณสมบัติข้อนี้เองที่ทำให้ RSA มีประโยชน์มากเพราะใช้ในการเซ็นระบบดิจิตอลได้อีกด้วย



รูปที่ 2.14 แสดงขั้นตอนการเข้ารหัสแบบ RSA

ตัวอย่างขั้นตอนการเข้ารหัสแบบ RSA ซึ่งมีขั้นตอนในการหา Key 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 นั่นเอง จากสมการที่ (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 นั้นไม่ง่ายเลยสำหรับ n ที่มีค่ามาก R.Rivest ได้คำนวณไว้ในปี 1992 ว่าจะต้องใช้เงินประมาณ 8.3 ล้านเหรียญสหรัฐในการแยกตัวประกอบของ n ที่มีความยาว 512 บิต ค่านี้อาจลดลงได้ในอนาคต ดังนั้นสำหรับข้อมูลที่มีความสำคัญมากกว่า อาจจำเป็นต้องใช้ n ที่มีค่ามากถึง 700 หรือ 1000 ก็ได้

เห็นได้ว่าไม่ว่าการเข้าหรือถอดรหัส RSA ก็จำเป็นต้องใช้การยกกำลังในระบบโมดูโล ซึ่งการยกกำลังนั้นสามารถทำได้โดยการใช้วงจรคูณระบบโมดูโลมาต่ออนุกรมกัน ดังนั้นความเร็วของทั้งการเข้าและถอดรหัสจึงขึ้นกับความเร็วของวงจรคูณระบบโมดูโลเป็นอย่างมาก นี่เองเป็นจุดอ่อนข้อหนึ่งของ RSA เมื่อเปรียบเทียบกับคีย์ที่เป็นความลับ เช่น DES เพราะปัจจุบันการคูณในระบบโมดูโลยังค่อนข้างยุ่งยาก เมื่อเปรียบเทียบกับการแทนค่าหรือสลับตำแหน่งใน 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 ใช้ในการเข้ารหัสข้อมูลที่มีขนาดใหญ่เพราะรวดเร็วกว่าในขณะที่ RSA ใช้ในระบบสื่อสารที่ไม่ยาวนักแต่ต้องการความปลอดภัยสูงในบางครั้ง RSA ยังใช้ร่วมกับ DES เพื่อเสริมจุดเด่นซึ่งกันและกัน เช่นตัวเอกสารจริงจะเข้ารหัสด้วย DES โดยที่คีย์รหัส DES จะเข้ารหัสด้วย RSA แล้วส่งไปด้วยกันหรือส่งไปก่อน แต่ในบางครั้ง DES อย่างเดียวก็พอแล้วหากการแลกเปลี่ยนคีย์สามารถทำได้อย่างปลอดภัยเพียงพอ หรือในกรณีที่ผู้ส่งและผู้รับเป็นบุคคลเดียวกัน เช่น ฮาร์ดดิสก์ในคอมพิวเตอร์ส่วนตัวหรือข้อมูลส่วนตัวในสมาร์ตการ์ด

2.5 การคำนวนแบบ MD5

เป็นอัลกอริทึมที่ใช้แยกแยะข้อความ (message digest) จากข้อความต่างๆ เพื่อให้ออกมามีขนาด 128 บิต โดย input จะจัดการทีละบล็อก ๆ ละ 512 บิต โดยมีขั้นตอนต่อไปนี้

ขั้นที่ 1 เติม Padding Bits ข้อความต้องต่อเติมให้มีความยาวที่ถูกโมดูโลด้วย 512 แล้วได้ 448 หากไม่ถึงหรือเกินให้เติม padding เข้าไปโดยเติมค่า 1 ในบิตเริ่มต้นและบิตถัดไปให้ค่า 0 จนครบ เพื่อใช้เนื้อที่ 64 บิตที่เหลือ (512 ลบ 448) ใส่ขนาด ของข้อความ(จำนวนบิต)

ขั้นที่ 2 เติมความยาว ใส่ขนาดของข้อความโดยมีค่าไม่เกิน 2^{64}

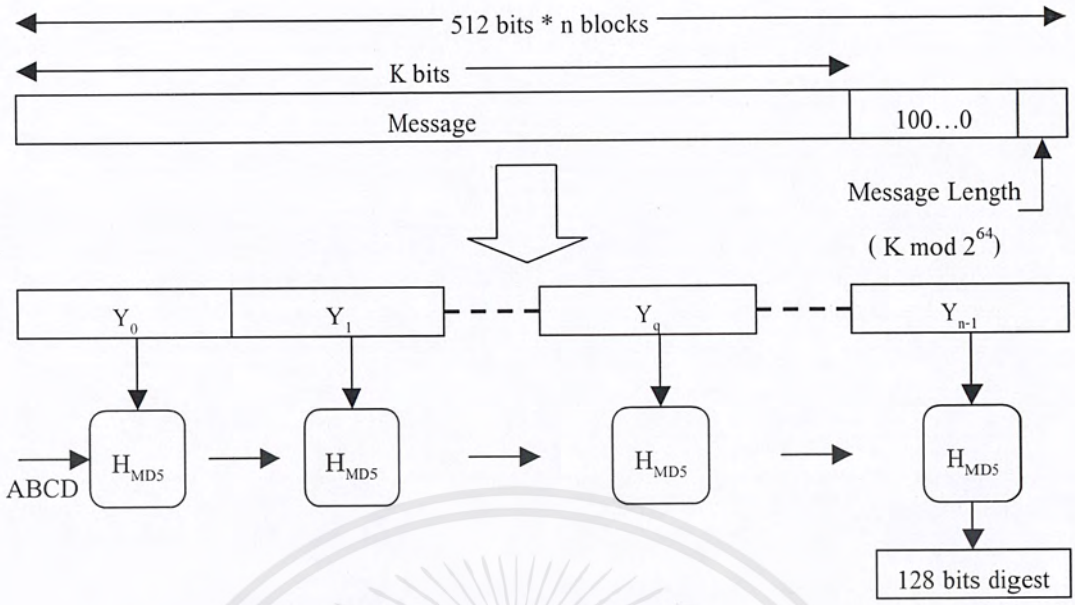
ขั้นที่ 3 กำหนดค่าเริ่มของ MD บัฟเฟอร์ บัฟเฟอร์มีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการ hash โดยบัฟเฟอร์เริ่มต้นประกอบด้วย 32 บิตไบต์ 4 ตัว (A,B,C,D) ซึ่งมีค่าต่อไปนี้

A = 01234567

B = 89ABCDEF

C = FEDCBA98

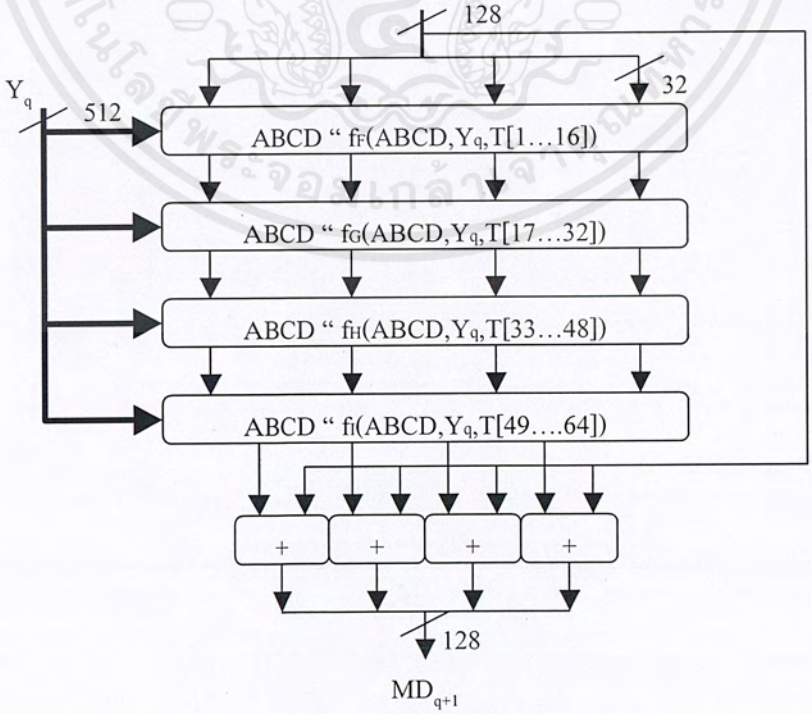
D = 76543210



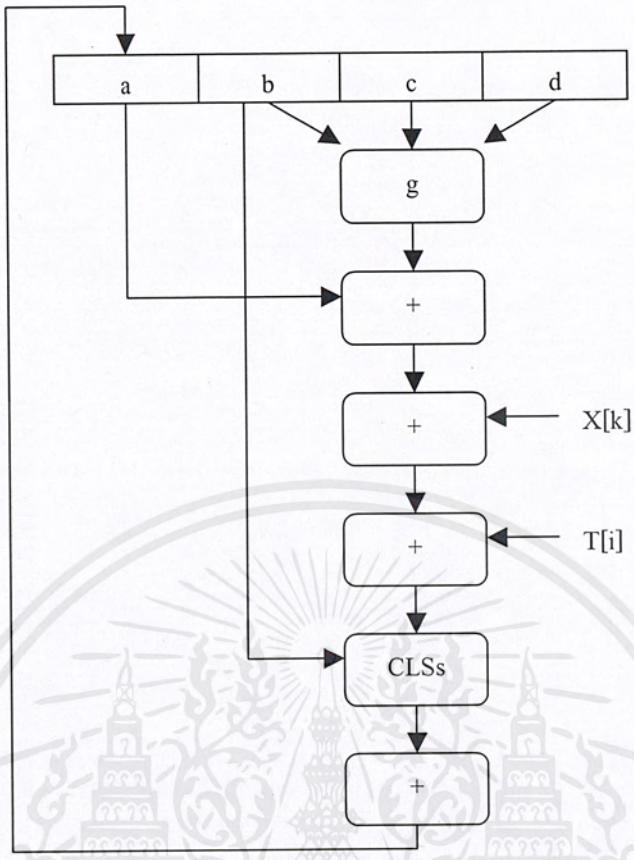
รูปที่ 2.15 แสดงการคำนวณแบบ MD5

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)

แต่ละรอบใช้อินพุตขนาด 512 บิตบล็อกในการจัดการ(Y_q) และ 128 บิตบัฟเฟอร์ที่มีค่า A,B,C,D และทำการอัพเดตค่าในบัฟเฟอร์ ในแต่ละรอบต้องมีการใช้ 1 ใน 4 ของ 64 ค่าในตาราง T[1…64] ซึ่งสร้างมาจากฟังก์ชัน sine โดย T[i] มีค่าเป็นจำนวนเต็มของ 232 * abs(sin(i)) โดยที่ i มีหน่วยเป็น radians



รูปที่ 2.16 กระบวนการทำในหนึ่งบล็อกของ MD5



รูปที่ 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)	$(b \cdot c) \vee (b \cdot \overline{d})$
FG	G(b, c ,d)	$(b \cdot d) \vee (c \cdot \overline{d})$
FH	H(b, c ,d)	$b \oplus c \oplus d$
FI	I(b, c ,d)	$c \oplus (b \cdot \overline{d})$

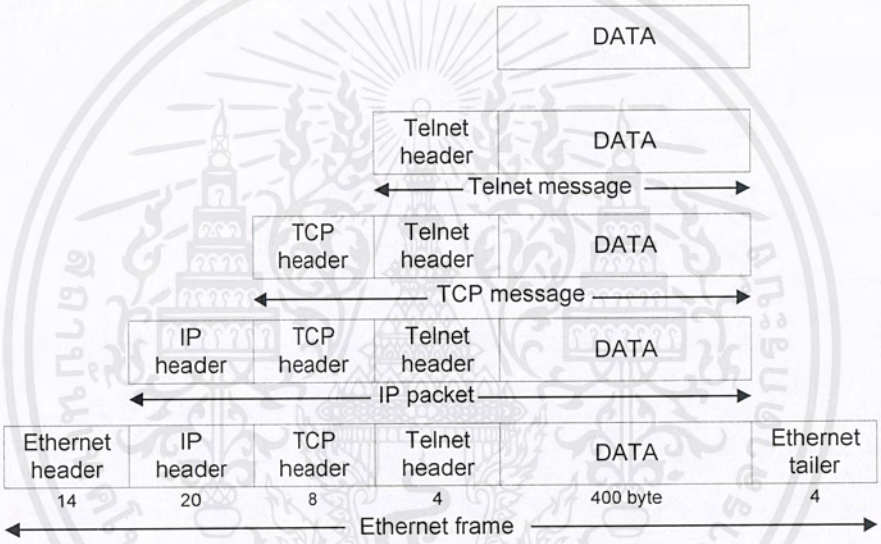
ขั้นที่ 5 Output นำเอาต์พุตที่ได้จากการทำครั้งแรก (128 บิต) ซึ่งได้เป็น A,B,C,D ใหม่มาทำการโพรเซสกับบล็อกถัดไปจนครบทั้งหมด ซึ่งได้ผลลัพธ์ขนาด 128 บิตด้วย

บทที่ 3
ภาพรวมของโปรแกรมเทลเน็ต

เทลเน็ต คือ โปรแกรมประยุกต์ที่ทำให้ผู้ใช้สามารถ ล็อกอินผ่านเทอร์มินอลใดๆ เพื่อขอใช้บริการจากโฮสต์ใด ๆ ที่เป็นเครื่องให้บริการที่อยู่ห่างไกลและใช้โพรโตคอล TCP / IP ในการติดต่อระหว่างเทอร์มินอลกับเครื่องให้บริการนั้นๆ นั่นเอง

3.1 เทลเน็ตทำงานอย่างไร

เทลเน็ต เป็นโปรแกรมประยุกต์ที่ทำงานอยู่บนแอพพลิเคชันของโพรโตคอล TCP/IP ดังนั้นข้อมูลจากผู้ใช้ก่อนที่จะส่งออกไปในระบบเครือข่ายต้องมีการเพิ่มส่วนหัวในชั้นต่างๆดังรูป



รูปที่ 3.1 แสดงลักษณะของเฟรมบน TCP/IP

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)			

รูปที่ 3.2 แสดงรายละเอียดของ TCP message

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)				

รูปที่ 3.3 แสดงรายละเอียดของแพ็กเก็ต IP

ส่วนพอร์ตบริการของเทลเน็ตนั้น โดยทั่วไปแล้วจะอยู่พอร์ตบริการเบอร์ 23 ดังนั้น เมื่อมีการเรียกใช้โปรแกรมเทลเน็ต การติดต่อสื่อสารจะกระทำกับพอร์ต 23 ที่เครื่องให้บริการเสมอ

3.2 เทลเน็ตมีหลักการทำงานเบื้องต้นอย่างไร

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

1. ที่เครื่องที่ผู้ใช้ (Client) ทำการเริ่มโปรแกรมเทลเน็ต
2. เทลเน็ตจากเครื่องผู้ใช้พยายามติดต่อกับเครื่องให้บริการ ถ้าติดต่อได้เครื่องให้บริการจะตอบ (ACK) กลับ ถ้าไม่ได้ ก็จะแจ้งผิดพลาด (ABORT) แล้วออกจากโปรแกรมไป
3. ถ้าติดต่อได้แล้ว เครื่องผู้ใช้ก็จะขอทำการติดต่อกับเครื่องให้บริการ โดยถ้าเครื่องให้บริการพร้อมให้บริการแล้วก็จะส่งสัญญาณ (ACK) ตอบกลับมา พร้อมทั้งส่ง Escape Character ที่ใช้ในการทำเทอร์มินอลเสมือน (Virtual Terminal)
4. เครื่องผู้ใช้เมื่อได้รับสัญญาณ (ACK) ตอบกลับแล้วจะรอข้อมูลจากผู้ใช้ป้อนชื่อและรหัสผ่านแล้วทำการส่งข้อมูลเหล่านั้นไปให้เครื่องให้บริการ
5. เครื่องให้บริการจะตรวจสอบชื่อและรหัสผ่านและทำการตอบกลับว่าล็อกอิน (Login) สำเร็จหรือไม่ถ้าสำเร็จก็จะเข้าสู่กระบวนการของเชลล์ (Shell process) ต่อไป

3.3 Network Virtual Terminal (NVT)

เนื่องจากเทลเน็ตในทางปฏิบัติ เมื่อผู้ใช้ทำการล็อกอินกับเครื่องให้บริการใดๆก็ตาม เทลเน็ตจะทำการล็อกอินผ่าน NVT เพื่อให้ ผู้ใช้สามารถใช้เทอร์มินอลของตนในการล็อกอินเข้าไปใช้บริการและอุปกรณ์อื่นๆ ของเครื่องให้บริการได้

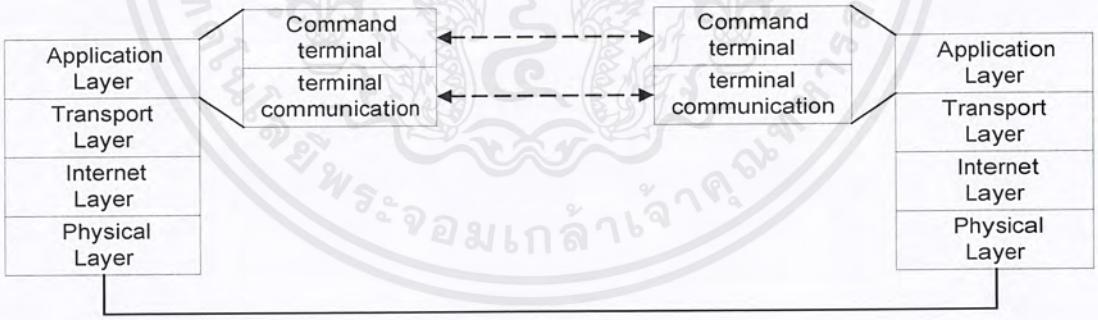
3.3.1 โครงสร้างหลักและหลักการของ NVT

NVT เป็นหลักการที่ใช้ส่งข้อมูลระหว่างแอพพลิเคชันของเทอร์มินอล (terminal application) ที่อยู่ต่างเครื่องและต่างสิ่งแวดล้อมกัน ซึ่งได้แก่ ความแตกต่างในเรื่องของระบบปฏิบัติการ (OS) ตลอดจนถึงความแตกต่างในเรื่องโพรโตคอลที่ใช้ในระดับบน (Upper-Level) ของแอพพลิเคชั่นที่ทำงานอยู่ต่างเครื่องกันนั้น ซึ่งหลักการของ NVT นี้ ทำให้แอพพลิเคชันที่ทำงานอยู่บนเครื่องใช้บริการและแอพพลิเคชันที่ทำงานอยู่บนเครื่องให้บริการ ซึ่งมีสิ่งแวดล้อมที่ต่างกันสามารถติดต่อกันได้ เปรียบเสมือนโปรแกรมเทลเน็ตที่ทำงานอยู่บนเครื่องใช้บริการที่ต้องติดต่อกับ Telnetd ที่ทำงานอยู่บนเครื่องให้บริการนั่นเอง

หลักการของการทำ NVT มีอยู่ว่า เรามองเทอร์มินอลที่รันแอพพลิเคชันทั้ง 2 ที่ต้องการติดต่อกันนั้นเป็น 2 เทอร์มินอล คือ Physical Terminal กับ Logical Terminal

Physical Terminal คือ เทอร์มินอลที่รันแอพพลิเคชันนั้นจริงๆ ซึ่ง Physical Terminal ที่รันแอพพลิเคชันทั้ง 2 อาจแตกต่างกันในหลายๆเรื่องดังที่ได้กล่าวมาแล้ว

Logical Terminal คือเทอร์มินอลในความคิด โดยเรากำหนดมาตรฐานต่างๆในการติดต่อให้เหมือนกันทั้ง 2 เครื่องและเมื่อ Logical Terminal บนเครื่องทั้ง 2 เหมือนกัน จึงทำให้ Logical Terminal ทั้ง 2 เครื่องคุยกันได้ และ Logical Terminal ของแต่ละเครื่องก็จะแมปข้อมูล (MAP Data) ที่ติดต่อกันนั้น ให้อยู่ในรูปแบบที่ Physical Terminal ของเครื่องตนเองเข้าใจ ดังนั้นแอพพลิเคชันที่รันอยู่บน Physical Terminal ที่ต่างกันจึงสามารถติดต่อและแลกเปลี่ยนข้อมูลกันได้ โดยไม่มีปัญหาเรื่องสภาพแวดล้อมของแต่ละเครื่องเข้ามาเกี่ยวข้องเลย ด้วยหลักการที่กล่าวทั้งหมดนี้ในทางปฏิบัติแล้วเราจะแบ่งชั้นแอพพลิเคชั่น (Application Layer) ออกเป็น 2 ชั้นย่อยๆ (Sublayer) ดังรูป



รูปที่ 3.4 แสดงโครงสร้างการทำ NVT

Command Terminal เป็นชั้นย่อย (Sublayer) ที่รันแอพพลิเคชันนั้นอยู่จริง โดยที่ชั้นย่อยนี้ มีสภาพแวดล้อมต่างๆตามเครื่องที่ทำงานอยู่ ซึ่งชั้นย่อยนี้ ก็เปรียบเสมือน Physical Terminal นั่นเอง

Terminal Communication เป็นชั้นย่อยที่ทำหน้าที่แปลงสิ่งแวดล้อมที่ไม่เหมือนกันทั้ง 2 เครื่องให้เป็นมาตรฐานเดียวกัน เพื่อให้แอพพลิเคชันทั้ง 2 สามารถคุยกันได้และแปลงข้อมูลที่ติดต่อกันให้อยู่ในรูปแบบที่ Physical Terminal ของตนเองเข้าใจ ซึ่งชั้นย่อยนี้ก็เปรียบเสมือน Logical Terminal นั่นเอง

3.3.2 การทำงานและออปชันของ NVT ในเทลเน็ต

การทำงานเริ่มเมื่อผู้ใช้รันเทลเน็ตที่เทอร์มินอลของตนเอง ที่สามารถติดต่อเข้าไปใน NVT ของเครื่องให้บริการที่ต้องการติดต่อด้วยได้อย่างที่ได้กล่าวมาแล้ว NVT ที่ติดต่อด้วยนั้นทำให้ผู้ใช้เหมือนได้ใช้เทอร์มินอลบนเครื่องให้บริการนั้นจริงๆ แต่อุปกรณ์ที่ผู้ใช้เห็นบนเครื่องให้บริการนั้นจริงๆแล้วเป็นเพียงการจำลองมาเท่านั้น

เพื่อให้เห็นภาพได้ชัดเจนยิ่งขึ้น คุณลองคิดถึงอุปกรณ์ในส่วนของเทอร์มินอล (Terminal Device) จริงๆดูว่าอะไรเป็นอุปกรณ์ที่เป็นอินพุต (เช่นแป้นพิมพ์) และอะไรเป็นอุปกรณ์ที่เป็นเอาต์พุต (เช่น จอภาพ, เครื่องพิมพ์) ดังนั้น NVT จะจำลองอุปกรณ์ในส่วนของเทอร์มินอลจริงๆเหล่านั้น ซึ่งก็มีแป้นพิมพ์เป็นอินพุตและเครื่องพิมพ์เป็นเอาต์พุต เครื่องพิมพ์จะพิมพ์ข้อความที่ได้รับมาจากเครื่องให้บริการบางข้อความที่เครื่องให้บริการต้องแสดงและแป้นพิมพ์ก็สร้างข้อความที่จะต้องส่งออกไปยังเครื่องให้บริการเสมือนกับว่าเทอร์มินอลที่ใช้อยู่นั้น เป็นเทอร์มินอลจริง ๆ บนเครื่องให้บริการนั่นเอง

เนื่องจากในแต่ละเครื่องให้บริการมักมีผู้ใช้จำนวนมาก ดังนั้นการทำให้ NVT ตอบสนองกับสภาพแวดล้อมของเทอร์มินอลของแต่ละผู้ใช้จึงทำได้ยาก วิธีแก้ก็คือ ใช้ออปชันต่างๆตามข้อตกลงที่แต่ละเทอร์มินอลและผู้ใช้แต่ละคนต้องการ โดยทำการตกลงกันในตอนที่ทำการติดต่อกัน ซึ่งออปชันที่ NVT เตรียมไว้มีตัวอย่างดังนี้

- ออปชันเปลี่ยนลักษณะตัวอักษรของ NVT
- ออปชันที่เลือกว่าให้ NVT ทำการส่งตัวอักษรกลับมาให้ผู้ใช้เห็นบนจอภาพหรือไม่
- ออปชันว่าเลือกโหมดส่งข้อมูลเป็นบรรทัดทีเดียวหรือเลือกโหมดส่งข้อมูลที่ละตัวอักษร

และยังมีออปชันอื่นๆอีก การขอทำออปชันต่างๆเหล่านี้ สามารถเลือกว่าจะใส่เพิ่มหรือถอนออกก็ได้ ซึ่งการตกลงเจรจาที่จะทำออปชันนั้น ทำได้โดยใช้ลำดับของข้อความ WILL, WON'T, DO และ DON'T นั่นเอง

"WILL X" เป็นข้อความที่ส่งจากด้านใดด้านหนึ่ง โดยด้านที่ส่งนั้นได้บอกอีกด้านหนึ่งที่ติดต่อด้วยว่า ตนเองจะทำออปชัน X ซึ่งถ้าอีกด้านหนึ่งยอมรับการทำออปชันนั้น ก็ตอบกลับมาด้วย "DO X" แต่ถ้าอีกด้านหนึ่งไม่ยอมรับการทำออปชันนั้น ก็ตอบกลับมาว่า "DON'T X"

"DO X" เป็นข้อความที่ส่งโดยด้านใดด้านหนึ่ง โดยที่มันต้องการให้อีกด้านหนึ่งที่ทำการติดต่อด้วยการทำออปชัน X ให้มัน ถ้าอีกด้านหนึ่งนั้นตอบรับการทำออปชัน X ก็ตอบกลับมาด้วย "WILL X" แต่ถ้าอีกด้านหนึ่งไม่ยอมทำออปชัน X ก็ทำการตอบกลับไปว่า "WON'T X" นั่นเอง ซึ่งตัวอย่างของเทลเน็ตออปชัน (Telnet option) ได้แสดงเอาไว้ในตารางภาคผนวกแล้ว

3.4 โหมดการส่งข้อมูลของเทลเน็ต

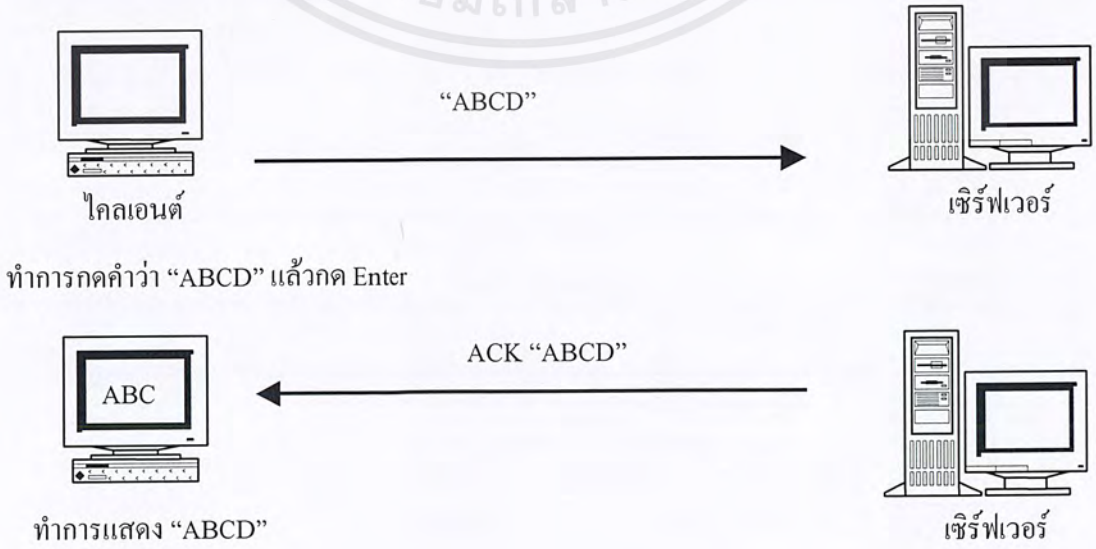
หลังจากที่ทำติดต่อระหว่างคำสั่งของเทลเน็ต (Telnet Command) บนเครื่องใช้บริการกับกระบวนการของ Telnetd บนเครื่องให้บริการแล้ว ก็มีการตกลงออปชันกัน หลังจากนั้นก็ทำการส่งข้อมูล ซึ่งโหมดการส่งข้อมูลที่เป็นคำสั่งของเทลเน็ตไปยังกระบวนการของ Telnetd มีอยู่ 2 โหมดคือ โหมดการส่งข้อมูลทีละบรรทัด (โหมด line-at-a-time) และโหมดการส่งข้อมูลทีละตัวอักษร (โหมด character-at-a-time)

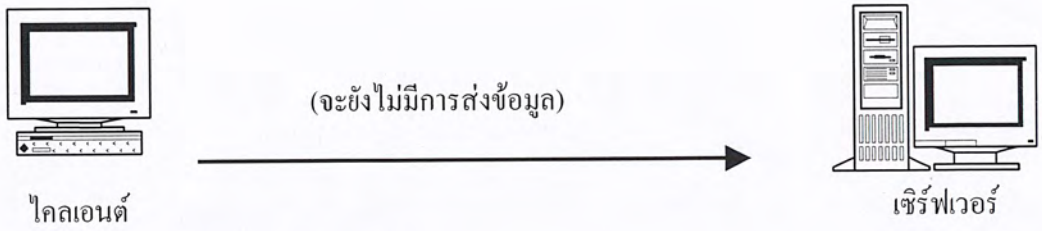
3.4.1 โหมดการส่งข้อมูลแบบทีละบรรทัด

การส่งข้อมูลในโหมดนี้ จะส่งข้อมูลทีละบรรทัดคือ ส่งเมื่อผู้ใช้มีการกดคีย์ขึ้นบรรทัดใหม่ “New line” หลังจากส่งข้อมูลจากเครื่องใช้บริการไปให้เครื่องให้บริการแล้ว เครื่องให้บริการก็จะตอบรับ (ACK) ข้อมูลที่ส่งมานั้นเท่านั้น จะไม่มีการส่งข้อความกลับมา (Echo) เพื่อแสดงผลบนหน้าจอเครื่องใช้บริการแต่อย่างใด เพราะการแสดงผลบนเครื่องใช้บริการจะควบคุมด้วยตัวของเครื่องใช้บริการเอง เนื่องจากไม่สามารถรอการส่งข้อความกลับมาจากเครื่องให้บริการได้ เพราะเครื่องใช้บริการเองต้องรอการกดคีย์เพื่อขึ้นบรรทัดใหม่ “New line” จึงจะส่งข้อมูล ดังนั้นถ้ารอข้อความที่ส่งกลับมาจากเครื่องให้บริการแล้วผู้ใช้จะไม่เห็นตัวอักษรที่ตนพิมพ์เข้าไปจนกว่าผู้ใช้กดคีย์เพื่อขึ้นบรรทัดใหม่ “New line” ซึ่งถ้าเป็นอย่างนี้แล้วจะเป็นการไม่สะดวกที่ผู้ใช้แก้ข้อความที่พิมพ์ผิด เพราะผู้ใช้ยังไม่เห็นข้อความที่พิมพ์เข้าไป

ดังนั้นเพื่อไม่ให้เกิดเหตุการณ์ดังกล่าว การส่งข้อมูลในโหมดนี้จึงให้เครื่องใช้บริการเป็นเครื่องที่ควบคุมการแสดงผลเอง ดังนั้นการส่งข้อมูลในโหมดนี้จึงไม่มีปัญหาในการใช้คีย์พิเศษเช่น Backspace หรือ Delete เพราะอย่างที่ได้กล่าวไปแล้วว่าการจัดการแสดงผลทำโดยเทอร์มินอลของผู้ใช้ และผู้ใช้ยังเป็นผู้กำหนดว่าส่งข้อมูลไปให้เครื่องให้บริการเมื่อไหร่เองอีกด้วย

แต่การส่งข้อมูลแบบทีละบรรทัดก็มีปัญหาคือ การส่งข้อมูลในโหมดนี้ เมื่อส่งข้อมูลไปแล้วเราไม่สามารถกลับไปแก้ไขข้อมูลเดิมได้อีก จึงทำให้ไม่สามารถใช้โปรแกรมประเภท Word หรือ Editor ได้ในเทลเน็ตโหมดนี้ เพราะการส่งข้อมูลแบบนี้ไม่สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้นั่นเอง ดังรูปที่ 3.5 ที่แสดงตัวอย่างในการส่งแบบทีละบรรทัด





รูปที่ 3.5 แสดงโหมดการส่งข้อมูลแบบ line-at-a-time

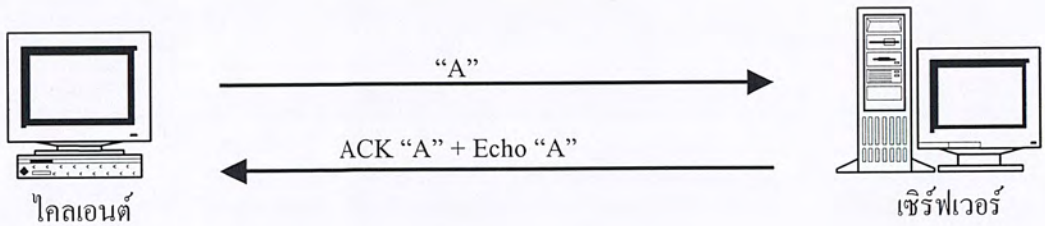
3.4.2 โหมดการส่งข้อมูลแบบทีละตัวอักษร

การส่งข้อมูลในโหมดนี้จะส่งข้อมูลที่ละตัวอักษร คือส่งทันทีที่ผู้ใช้มีการกดคีย์ใดๆ และส่งข้อมูลทีละตัวอักษร ซึ่งการส่งข้อมูลในโหมดนี้คำสั่งของเทลเน็ตที่ทำงานอยู่บนเครื่องใช้บริการจะยังไม่แสดงผลข้อมูลที่ส่งไปยังเครื่องให้บริการบนจอภาพทันที แต่จะรอให้เครื่องให้บริการตอบรับและส่งข้อความกลับมาให้จึงแสดงผลบนจอภาพ

ข้อดีของการส่งข้อมูลในโหมดนี้ คือผู้ใช้สามารถแก้ไขข้อมูลที่ส่งไปแล้วได้ง่ายกว่าการส่งข้อมูลในโหมดการส่งแบบทีละบรรทัด เพราะข้อมูลที่ส่งไปแล้วนั้นมีขนาดเพียง 1 ตัวอักษร ดังนั้นเราจึงส่งข้อมูลไปแก้ไขข้อมูลที่ส่งไปแล้วเหล่านั้นได้ แล้วด้วยเหตุนี้จึงทำให้การส่งข้อมูลในโหมดนี้สามารถใช้งานโปรแกรมจำพวก Word หรือ Editor ได้ จึงทำให้การส่งข้อมูลในโหมดนี้ เป็นที่นิยมใช้กันในปัจจุบัน

แต่การส่งข้อมูลในโหมดการส่งแบบทีละตัวอักษรนี้ก็มีปัญหาคือ การแสดงผลบนจอภาพของการส่งข้อมูลในโหมดนี้นั้นต้องแสดงจากข้อความที่ส่งกลับมาจากเครื่องให้บริการ ดังนั้นการแสดงผลบางอย่างจึงไม่สามารถทำได้ เช่น เมื่อผู้ใช้กด Key backspace แล้วโฮสต์ส่ง ASCII ของ backspace กลับมาให้เครื่องใช้บริการ แล้วเครื่องใช้บริการของผู้ใช้จะแสดงผลของ Key backspace ออกมาเป็นตัวอักขระ ASCII ของ backspace ซึ่งไม่ได้เป็นไปตามที่ผู้ใช้ต้องการ แต่สิ่งที่ผู้ใช้ ต้องการคือ ให้ Move Cursor กลับไปหนึ่งตัวอักษรและลบอักขระตัวนั้นด้วย

ดังนั้น จึงต้องมีการทำเทอร์มินอลเสมือนที่กล่าวถึงในหัวข้อต่อไป



รูปที่ 3.6 แสดงโหมดการส่งข้อมูลแบบ character-at-a-time

3.5 เทอร์มินอลเสมือน (Virtual Terminal)

เหตุผลในการทำเทอร์มินอลเสมือน ก็เพราะการส่งข้อมูลในโหมดการส่งข้อมูลทีละตัวอักษร การแสดงผลบนจอภาพของเครื่องผู้ใช้บริการจะควบคุมโดยเครื่องให้บริการ แต่เนื่องจากเครื่องใช้บริการและเครื่องให้บริการอยู่ห่างไกลกัน การติดต่อก็ทำได้โดยผ่านเครือข่ายที่เชื่อมโยงเครื่องทั้งสอง ซึ่งการส่งข้อมูลผ่านเครือข่ายนี้เอง ทำให้เครื่องให้บริการไม่สามารถควบคุมการแสดงผลของเครื่องใช้บริการด้วยการส่งข้อความกลับมาเพียงอย่างเดียวได้ ดังนั้นจึงต้องมีการทำเทอร์มินอลเสมือน เช่น เมื่อผู้ใช้กดคีย์ backspace แทนที่เครื่องให้บริการส่งตัวอักษรที่เป็น backspace กลับมาก็ส่งคำสั่งให้เครื่องใช้บริการทำการย้ายตำแหน่งเคอร์เซอร์ถอยหลังกลับ 1 ตัวอักษรแทนเป็นต้น ซึ่งคำสั่งที่เครื่องให้บริการใช้ทำเทอร์มินอลเสมือนเหล่านี้นั้น มีลักษณะเป็นชุดของคำสั่งต่างๆ โดยเริ่มต้นด้วย “Escape” character ซึ่งตกลงกันเอาไว้ระหว่างเครื่องให้บริการและเครื่องใช้บริการตั้งแต่ตอนที่ติดต่อกันตอนแรก (ในขณะที่ตกลงออปชันต่างๆ นั่นเอง) ซึ่งชุดของคำสั่งที่ใช้ในการทำเทอร์มินอลเสมือนเหล่านี้ ได้มีการกำหนดไว้เป็นมาตรฐานต่างๆไว้มากมายหลายมาตรฐาน แต่ที่นิยมใช้กันมากที่สุดได้แก่ มาตรฐานตระกูล VT ซึ่งในภาคผนวกได้มีตัวอย่างโค้ดของ VT 100 ไว้ให้แล้ว

บทที่ 4

SSH (Secure Shell)

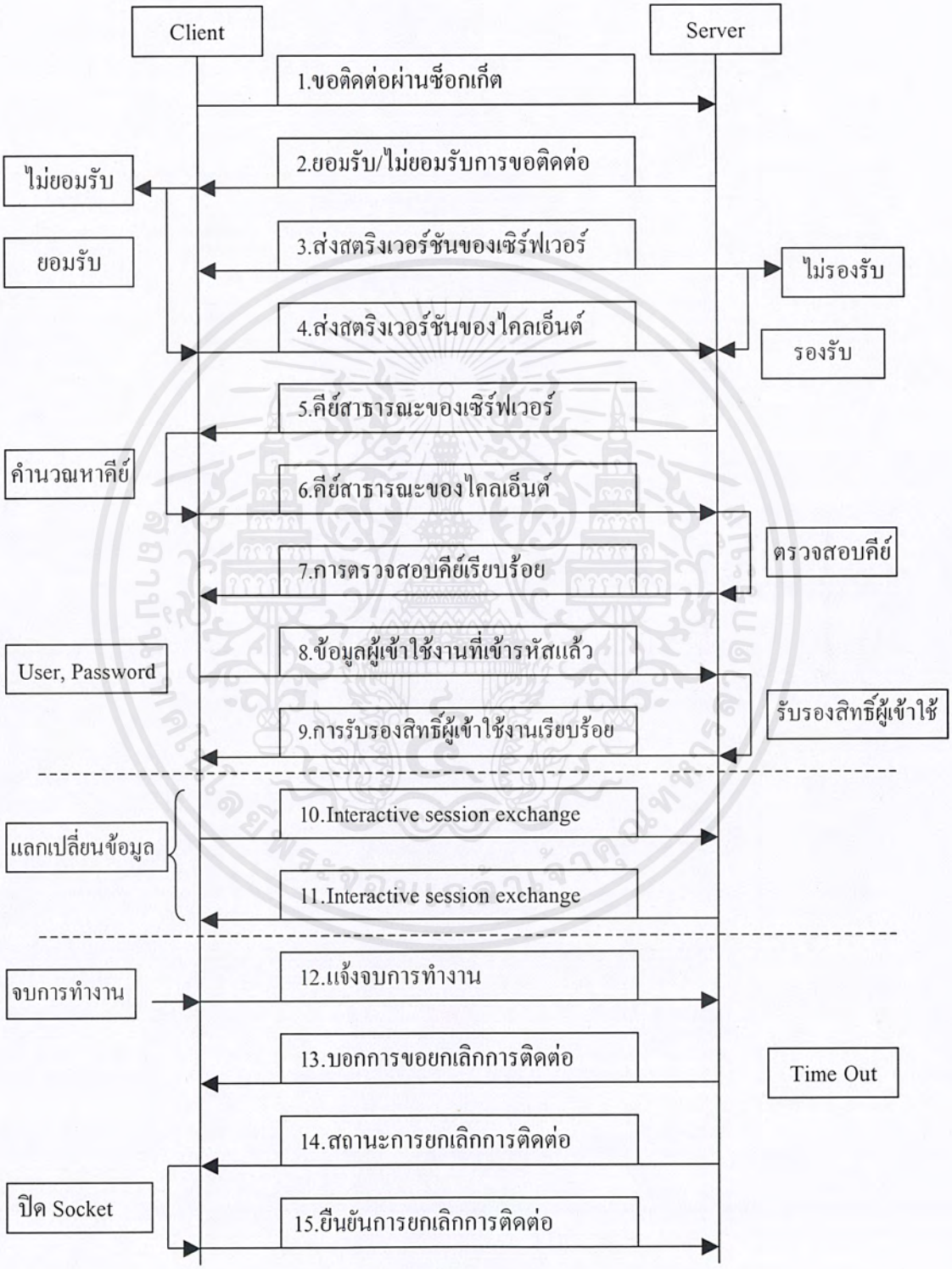
SSH เป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่ผู้ใช้สามารถล็อกอินเข้าไปใช้บริการจากเครื่องให้บริการ (Server) ที่อยู่ห่างไกลบนระบบเครือข่าย และสามารถทำการประมวลผล คำสั่งในเครื่องให้บริการที่อยู่ห่างไกลได้ (remote machine) พร้อมทั้งสามารถแลกเปลี่ยนข้อมูลบนระบบเครือข่ายอินเทอร์เน็ตได้อย่างปลอดภัย เนื่องจากมีวิธีการพิสูจน์ความเป็นเจ้าของที่มีประสิทธิภาพ (Strong Authentication) และระบบการติดต่อที่ปลอดภัย (Secure Communication) บนระบบเครือข่าย

4.1 ภาพรวมของโพรโตคอล SSH

โพรโตคอล SSH ประกอบด้วยโปรแกรมในส่วนของเครื่องที่ให้บริการ (Server) และอีกส่วนหนึ่งคือส่วนของเครื่องใช้บริการ (Client) โดยการติดต่อระหว่างเครื่องให้บริการและเครื่องใช้บริการจะติดต่อกันโดยใช้หมายเลข IP (IP Address) และ TCP/IP Socket ซึ่งเป็นการติดต่อสองทิศทาง (Bi-direction communication) โดยมีขั้นตอนดังนี้

1. การติดต่อเริ่มการติดต่อจากเครื่องใช้บริการ โดยเครื่องให้บริการรอฟังจากพอร์ตที่กำหนดไว้เฉพาะ ว่ามีเครื่องต้องการใช้บริการติดต่อเข้ามายังพอร์ตนั้นหรือไม่ ซึ่งเครื่องให้บริการสามารถติดต่อกับเครื่องใช้บริการได้หลายเครื่องพร้อมๆกัน
2. เมื่อเครื่องผู้ใช้บริการติดต่อกับเครื่องผู้ให้บริการ ก็จะตรวจสอบว่าผู้ให้บริการยอมรับการติดต่อหรือไม่ ถ้าไม่ก็จะยกเลิกการติดต่อ
3. หากว่าเครื่องผู้ให้บริการยอมรับการติดต่อจะส่งข้อความเพื่อตรวจสอบเวอร์ชันของกันและกันเรียกว่า *Version Identification String* ของตนกลับไป
4. เมื่อเครื่องใช้บริการได้รับจะแปลข้อความของผู้ให้บริการและส่งข้อความในการตรวจสอบเวอร์ชันของตนกลับไปเช่นกัน ซึ่งหน้าที่ของ *Version Identification String* คือ เป็นข้อความที่ใช้ในการตรวจสอบการติดต่อว่าเวอร์ชันของโปรแกรมและโพรโตคอลนั้นรองรับกันหรือไม่
5. หลังจากตรวจสอบเวอร์ชันของกันและกันแล้ว ข้อมูลในการติดต่อเปลี่ยนมาเป็นรูปแบบของแพ็กเก็ตซึ่งเรียกโพรโตคอลในการติดต่อนี้ว่า *Binary packet protocol* โดยเครื่องผู้ให้บริการเริ่มส่งการติดต่อโดยส่งกุญแจสาธารณะที่เรียกว่า Host Public Key, Server Key และข้อมูลอื่นๆไปยังเครื่องผู้ใช้บริการ
6. ตัวผู้ใช้บริการผลิตเซสชันคีย์ (Session Key) ซึ่งใช้ในการเข้ารหัสข้อมูลที่มีขนาด 256บิต และเข้ารหัสแบบ RSA 2 ครั้ง แล้วส่งไปให้เครื่องผู้ให้บริการพร้อมทั้งชนิดของการเข้ารหัสข้อมูล (Cipher) ที่ใช้
7. ฝั่งผู้ที่ให้บริการถอดรหัสว่าถูกต้องหรือไม่เสียก่อน หลังจากนั้นทั้ง 2 ฝั่งจึงเปิดการติดต่อกันโดยข้อมูลที่ติดต่อกันนั้นมีการเข้ารหัสโดยใช้กุญแจ (Key) และวิธีการที่ได้กำหนดไว้ข้างต้น

8. ต่อมาเครื่องผู้ใช้บริการจะตรวจสอบสิทธิ์ของตน โดยส่งหมายเลขเพื่อบ่งบอกวิธีในการพิสูจน์และพิสูจน์สิทธิ์ตามแบบที่ได้ส่งไปเซิร์ฟเวอร์
9. เครื่องผู้ให้บริการตรวจสอบและพิสูจน์สิทธิ์ว่าถูกต้องหรือไม่



รูปที่ 4.1 แสดงการขั้นตอนการทำงานในโพรโตคอล SSH

10. ถ้าถูกต้องแล้วเครื่องผู้ใช้บริการจะส่งตัวเลขเพื่อเตรียมตัวสำหรับการเรียกใช้เชลล์ (Shell) และเรียกใช้เชลล์เพื่อที่จะเข้าไปติดต่อในส่วนอินเตอร์แอ็กทีฟเซสชันโหมด (Interactive Session Mode)
11. การทำงานในโหมดนี้เป็นโหมดในการแลกเปลี่ยนข้อมูลกันและกันระหว่างผู้ที่ใช้บริการกับเครื่องผู้ให้บริการ
12. การติดต่อแลกเปลี่ยนข้อมูลสิ้นสุดการติดต่อเมื่อเครื่องผู้ให้บริการส่งคำสั่งเลิกบริการ (Exit Status) ของโปรแกรมไปยังเครื่องผู้ใช้บริการ
13. หรือหากว่าไม่มีการติดต่อใดๆกับผู้ให้บริการเลย จนกระทั่งระยะเวลาหนึ่งเกินค่าที่กำหนดไว้ ผู้ให้บริการขอการยกเลิกการติดต่อกับผู้ใช้บริการรายนั้นไป
14. เครื่องผู้ให้บริการส่งสถานะในการยกเลิกการติดต่อกับผู้ใช้บริการเพื่อให้ผู้ใช้บริการยืนยันกับสถานะที่ส่งไปดังกล่าว
15. เครื่องผู้ใช้บริการยืนยันการยกเลิกการติดต่อสื่อสารไปยังเครื่องผู้ให้บริการ แล้วยกเลิกการติดต่อกับเครื่องผู้ให้บริการ

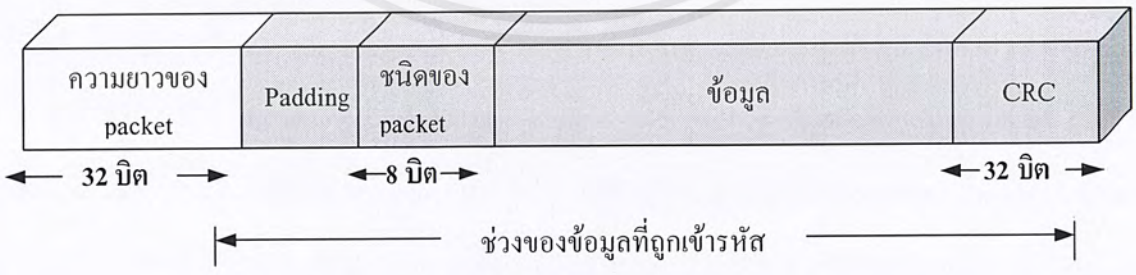
ซึ่งขั้นตอนคร่าวๆในการติดต่อสื่อสารของโพรโตคอล SSH สามารถสรุปได้ดังรูปที่ 4.1

4.2 ลักษณะและโพรโตคอลของข้อมูล

จากภาพรวมของโพรโตคอลที่ได้กล่าวมาข้างต้น ทำให้เราทราบวิธีการทำงาน ขั้นตอนการติดต่อและรูปแบบข้อความที่ใช้ติดต่อกัน แต่ก่อนที่เราจะศึกษาขั้นตอนการทำงานอย่างละเอียด เรามาทำความรู้จักกับคำที่ได้กล่าวมาข้างต้นก่อนว่าแต่ละตัวคืออะไร ทำหน้าที่อย่างไร และข้อความที่ใช้มีอะไรบ้าง

4.2.1 Binary Packet Protocol

ภายหลังจากการทำการตรวจสอบเวอร์ชันของกันและกัน ทั้งเครื่องให้บริการและเครื่องผู้ใช้บริการจะส่งแพ็กเก็ตที่มีรูปแบบเฉพาะ ซึ่งเรียกว่า ***Binary Packet Protocol*** รูปแบบของแพ็กเก็ตเป็นดังนี้



รูปที่ 4.2 แสดงฟิลด์ต่างๆ บน แพ็กเก็ต Binary Packet Protocol

ความยาวของแพ็กเก็ต : มีขนาด 32 บิตซึ่งเป็นจำนวนเต็มบวก แบ่งออกเป็น 8 บิต/ไบต์ บิตแรกเป็นบิตที่มีค่าสูงสุด ความยาวของแพ็กเก็ต ไม่นับรวมฟิลด์ของความยาวของแพ็กเก็ตและ Padding ความยาวสูงสุดของแพ็กเก็ต คือ 26244 ไบต์

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

ชนิดของแพ็กเก็ต : มีขนาด 8 บิต ค่า 255 กันไว้สำหรับใช้ในอนาคต

ข้อมูลที่ส่ง : ข้อมูล (มีลักษณะเป็นแบบเลขฐานสอง) ที่ส่ง โดยขึ้นกับชนิดของแพ็กเก็ต จำนวนของข้อมูลมีขนาดเท่ากับ ความยาวของแพ็กเก็ตลบ 5

CRC : เป็นข้อมูลที่ใช้ตรวจสอบ โดยมีขนาด 32 บิต โดยนำเอา padding, ชนิดของแพ็กเก็ตและฟิลด์ข้อมูล มาทำ CRC(Cyclic Redundancy Check) ด้วยโพลีโนเมียล 0xedb88320 โดย CRC นี้จะคำนวณก่อนการเข้ารหัส

แพ็กเก็ตนี้ยกเว้นฟิลด์ของความยาว จะเข้ารหัสโดยอัลกอริทึมที่เลือกใช้ โดยความยาวของส่วนที่เข้ารหัส (padding + ชนิดของแพ็กเก็ต + ข้อมูล + Check) เป็นจำนวนเท่าของ 8 ไบต์

4.2.2 การเข้ารหัสแพ็กเก็ต (Packet Encryption)

แพ็กเก็ตของโพรโตคอลนี้รองรับการเข้ารหัสได้หลายชนิด โดยตั้งแต่เริ่มต้นการติดต่อ ตัวเซิร์ฟเวอร์จะส่งบิทมาส์ก (bitmask) ของวิธีการเข้ารหัสแบบต่างๆ ที่เครื่องให้บริการรองรับ ตัวเครื่องผู้ใช้บริการจะเลือกวิธีการในการเข้ารหัสข้อมูล และสร้างเซสชันคีย์ ขนาด 256 บิต และส่งไปให้เครื่องผู้ให้บริการ

วิธีการเข้ารหัสที่รองรับและโค้ดของวิธีการเข้ารหัสแต่ละแบบคือ

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

ตารางที่ 4.1 แสดงวิธีการเข้ารหัสที่ SSH รองรับ

4.2.3 ชนิดของข้อมูลที่เข้ารหัส (Data Type Encoding)

ฟิลด์ข้อมูลของแต่ละแพ็กเก็ตจะเก็บข้อมูลที่เข้ารหัสตามวิธีที่ได้อธิบายไป ซึ่งประกอบรายการของข้อมูลหลายๆตัว แต่ละรายการมีรหัสในแต่ละแบบและนำมาใช้โดยการนำแต่ละรายการมาต่อกัน

ข้อมูลแต่ละชนิดถูกเก็บตามนี้คือ

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

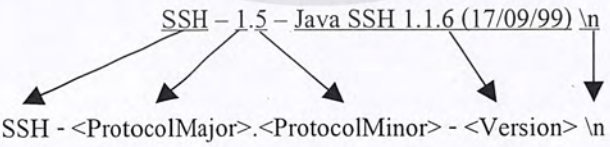
4.2.4 หมายเลข TCP/IP พอร์ต (TCP/IP Port Number & Other option)

ตัวผู้ให้บริการจะรอติดต่อบนโพรโตคอล TCP/IP พอร์ตที่ 22 ส่วนผู้ใช้บริการจะใช้พอร์ตใดของตนก็ได้ในการติดต่อเข้ากับเครื่องผู้ให้บริการ แต่ถ้าเครื่องใช้บริการต้องการใช้การติดต่อแบบ .rhosts หรือ /etc/hosts.equiv จะต้องติดต่อจากพอร์ตที่สงวนไว้ (เบอร์พอร์ตที่ต่ำกว่าเบอร์ 1024)

4.3 รายละเอียดขั้นตอนการติดต่อ

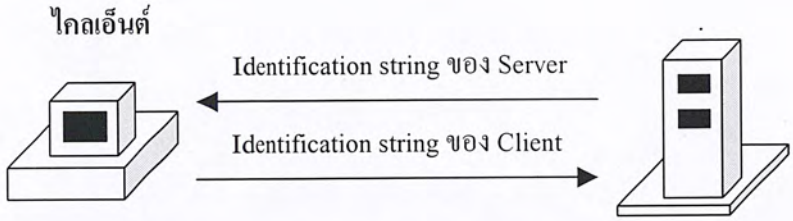
4.3.1 ช่วงตรวจสอบเวอร์ชัน (Protocol Version Identification)

เมื่อผู้ใช้บริการติดต่อไปยังเครื่องให้บริการได้แล้ว เครื่องให้บริการจะส่งข้อความเฉพาะที่ใช้ในการตรวจสอบเวอร์ชัน ที่เรียกว่า Version identification sting ซึ่งมีรูปแบบคือ **"SSH-<protocol major>.<protocolminor> - <version> \n "** ซึ่ง <protocolmajor> และ <protocolminor> คือ ตัวเลขจำนวนเต็มที่บอกเวอร์ชันของโพรโตคอลที่ใช้ ส่วน < version > เป็นเวอร์ชันของซอฟต์แวร์ที่ฝั่งผู้ใช้บริการใช้ (ไม่เกิน 40 ตัวอักษร) ดังตัวอย่าง (\n คือเครื่องหมายขึ้นบรรทัดใหม่)



รูปที่ 4.3 แสดงตัวอย่างของ Identification String

ส่วนผู้ใช้บริการเมื่อได้รับข้อความมาจะทำการแปล ถ้าไม่รองรับเวอร์ชันของโพรโตคอลก็ปิดการติดต่อ แต่ถ้ารองรับเครื่องผู้ใช้บริการจะส่งข้อความที่เป็นข้อมูลของตัวเองลักษณะเดียวกับที่เครื่องผู้ให้บริการส่งมากลับไป เมื่อเครื่องผู้ให้บริการได้รับจะตีความหมาย ถ้าเป็นเวอร์ชันของโพรโตคอลที่รองรับตรงกัน จะส่งข้อความแรกกลับไป ซึ่งข้อความอยู่ในรูปของ binary packet protocol



รูปที่ 4.4 แสดงการตรวจสอบเวอร์ชัน Identification

4.3.2 ช่วงการแลกเปลี่ยนกุญแจ (Key Exchange)

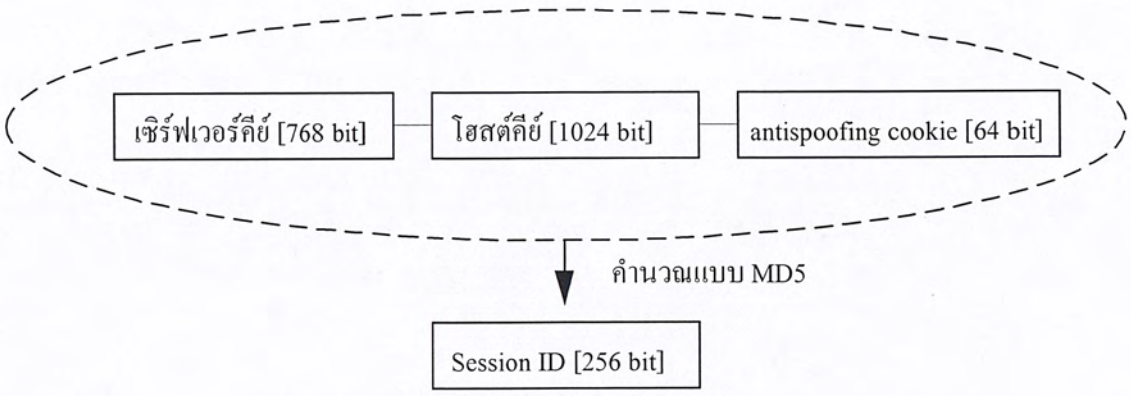
หลังจากที่มีการตรวจสอบเวอร์ชันว่ารองรับกันได้แล้ว จะเข้าสู่การแลกเปลี่ยนกุญแจ เนื่องจากการเข้ารหัสของข้อมูลต่างๆ ที่ใช้ติดต่อกันทั้งสองฝ่ายต้องมีกุญแจเดียวกันในการเข้ารหัสและถอดรหัสของข้อมูลนั้นๆ ซึ่งต้องมีวิธีการทำให้ทั้งคู่มีกุญแจเดียวกัน โดยที่ผู้อื่นไม่ทราบว่ากุญแจนี้คืออะไร ซึ่งมีขั้นตอน คือ

หลังจากที่เซิร์ฟเวอร์ได้รับ Identification String จากไคลเอ็นต์และตรวจสอบว่ารองรับแล้ว เซิร์ฟเวอร์จะส่งข้อความในรูปแบบ Binary Packet Protocol ที่เป็นชนิดของ SSH_SMSG_PUBLIC_KEY ซึ่งประกอบด้วย

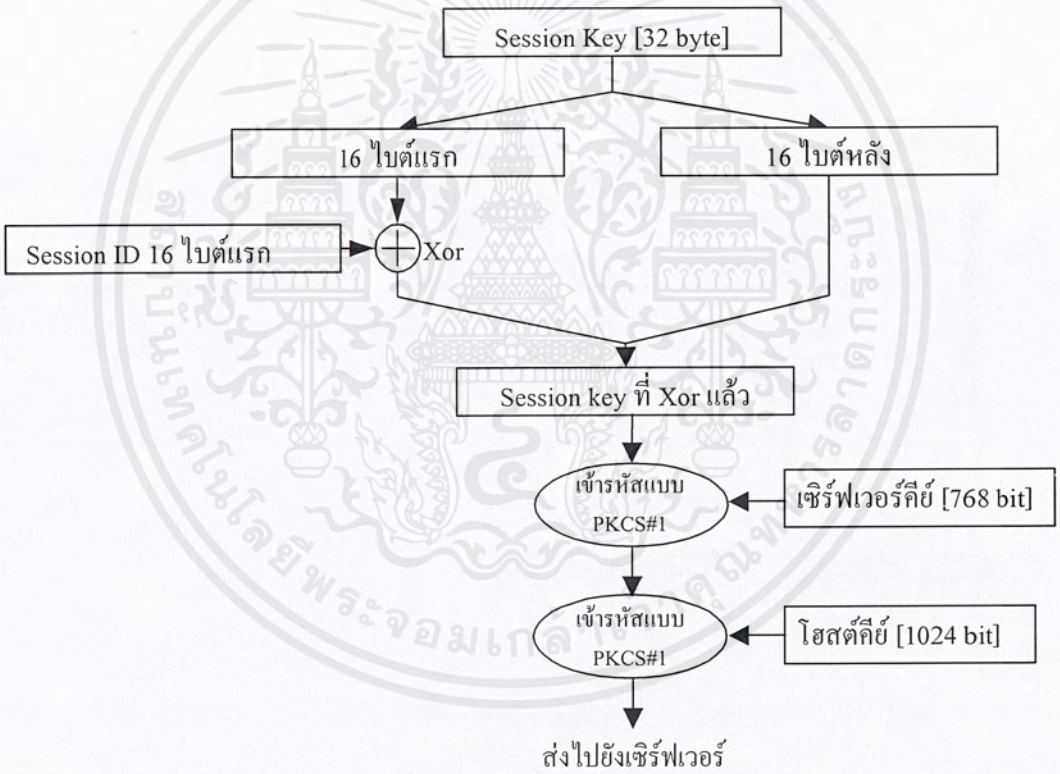
- **ค่าที่สุ่มขึ้นมาขนาด 64 บิต (cookie)** เพื่อให้ไคลเอ็นต์ส่งข้อมูลนี้กลับมา ทำให้ยากต่อการปลอมแปลง IP
- **โฮสต์คีย์ (host key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ มีขนาด 1024 บิต
- **เซิร์ฟเวอร์คีย์ (server key)** เป็นกุญแจสาธารณะของเซิร์ฟเวอร์ที่มีการสร้างใหม่ทุกๆ ชั่วโมง มีขนาด 768 บิต
- **การเข้ารหัสที่รองรับ (supported ciphers)** เป็นชนิดของการเข้ารหัสที่เซิร์ฟเวอร์รองรับ
- **การพิสูจน์สิทธิ์ที่รองรับ (supported authentication method)** เป็นวิธีของการพิสูจน์สิทธิ์ที่เซิร์ฟเวอร์รองรับ

เมื่อไคลเอ็นต์ได้รับข้อความ SSH_SMSG_PUBLIC_KEY ไคลเอ็นต์จะสร้างกุญแจ (เรียกว่า session key) ที่ใช้สำหรับเข้ารหัสแพ็กเก็ตต่างๆ ขึ้นมาโดยการสุ่มค่าขึ้น (กุญแจมีขนาด 32 ไบต์) เพื่อส่งไปให้เซิร์ฟเวอร์ ก่อนที่ส่งจะนำเอาเซสชันคีย์ 16 บิตแรก มาเอ็กซ์คลูซีฟออร์ กับ session ID (มีขนาด 16 ไบต์) แล้วนำมาต่อรวมกับ 16 บิตหลังของเซสชันคีย์ จากนั้นจึงเข้ารหัสแบบ PKCS#1 (ซึ่งเป็นการเข้ารหัสแบบ RSA) 2 ครั้งด้วยโฮสต์คีย์ และ เซิร์ฟเวอร์คีย์ โดยใช้กุญแจที่สั้นกว่าก่อน (ซึ่งคือ เซิร์ฟเวอร์คีย์)

สำหรับ Session ID เป็นค่าที่ทั้งสองฝั่งคำนวณหาได้ โดยการนำเอาโฮสต์คีย์ ต่อด้วย เซิร์ฟเวอร์คีย์ ต่อด้วย cookie แล้วนำมาคำนวณด้วย อัลกอริทึมแบบ MD5 ซึ่งทั้งสองฝั่งได้ค่านี้เช่นเดียวกัน

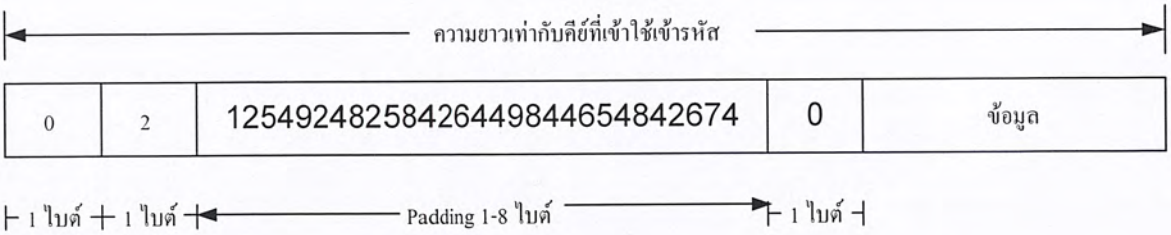


รูปที่ 4.5 แสดงการคำนวณหา Session ID



รูปที่ 4.6 แสดงการเข้ารหัสเซสชันคีย์ก่อนส่งให้เซิร์ฟเวอร์

สำหรับการเข้ารหัสแบบ PKCS#1 นั้นใช้มาตรฐานของ RSA มีลักษณะของการเข้ารหัสเหมือน RSA โดยให้ข้อมูลที่เข้ารหัสนั้น ให้ไบต์แรกมีค่าเป็น 0 ไบต์ถัดมามีค่าเป็น 2 (ตามมาตรฐาน) แล้วตามด้วยค่าที่สุ่มขึ้นและไม่เท่ากับ 0 ในตำแหน่งถัดมาที่ไม่ได้ใช้ ตามด้วยไบต์ 0 และตามด้วยข้อมูลที่ต้องการเข้ารหัส แล้วนำข้อมูลนี้มาเข้ารหัสแบบ RSA (C = p^e mod n โดย C คือผลการเข้ารหัส, p คือ ข้อมูลที่เข้ารหัส, e คือ พลับบลิคคีย์เอ็กซ์โปเนนท์ และ n คือ พลับบลิคคีย์โมดูลัส)

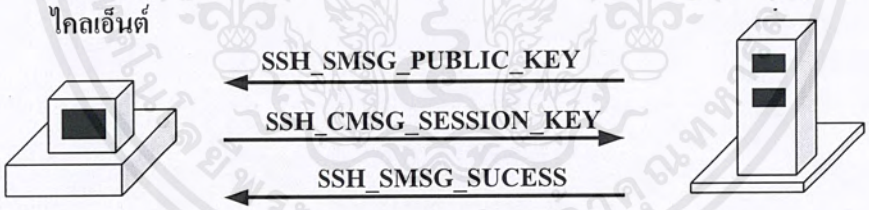


รูปที่ 4.7 แสดงตัวอย่างของข้อมูลที่ทำการเข้ารหัส PKCS#1

หลังจากที่ได้ข้อมูลที่เป็นคีย์ (ที่ทำการเข้ารหัสข้างต้น) ไคลเอ็นต์ก็จะส่งแพ็กเก็ตที่มีชนิดเป็น SSH_CMSG_SESSION_KEY โดยมีข้อมูล ต่างๆ ดังต่อไปนี้ ตามลำดับ

- ชนิดของการเข้ารหัส
- cookie ที่ได้จากเซิร์ฟเวอร์
- เซสชันคีย์ที่เข้ารหัสข้างต้น
- โพรโตคอลเวอร์ชันที่ใช้

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเก็ต SSH_CMSG_SESSION_KEY จะถอดรหัสและคำนวณหา เซสชันคีย์ออกมา หากลักษณะของข้อมูลถูกต้องจะส่งแพ็กเก็ตกลับไปว่าเรียบร้อยแล้ว โดยมีชนิดว่า SSH_SMSG_SUCCESS ซึ่งข้อมูลนี้เข้ารหัสด้วยวิธีการตามที่ไคลเอ็นต์เลือกมา



รูปที่ 4.8 ขั้นตอนการส่งของแพ็กเก็ตต่างๆ ในช่วงการแลกเปลี่ยนคีย์

4.4 ช่วงตรวจสอบชื่อและพิสูจน์สิทธิ์ (Declare User Name & Authentication Phase)

ไคลเอ็นต์ส่งแพ็กเก็ตชนิด SSH_CMSG_USER ไปยังเซิร์ฟเวอร์ ซึ่งมีชื่อของผู้ใช้ในแพ็กเก็ตนี้ เมื่อเซิร์ฟเวอร์ได้รับจะตรวจสอบว่ามีผู้ใช้คนนั้นในรายการหรือไม่ และตรวจสอบการตรวจสอบสิทธิ์ที่ต้องใช้ แล้วตอบกลับด้วยแพ็กเก็ตชนิด SSH_SMSG_SUCCESS ถ้าผู้ใช้คนนี้ถูกกำหนดไว้ว่าไม่ต้องการให้มีการพิสูจน์สิทธิ์ (ไม่ต้องใช้รหัสผ่าน) หรือ SSH_SMSG_FAILURE ถ้าผู้ใช้จำเป็นต้องมีการพิสูจน์สิทธิ์

สำหรับในกรณีที่ไม่มีชื่อของผู้ใช้คนนี้ เซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_FAILURE มาเช่นกัน ทำให้ผู้ใช้ไม่ทราบว่าผู้ใช้คนนี้มีอยู่หรือเปล่า

เมื่อไคลเอ็นต์ได้รับแพ็กเก็ต SSH_SMSG_FAILURE ไคลเอ็นต์จะส่งวิธีการในการพิสูจน์สิทธิ์ให้กับเซิร์ฟเวอร์ ซึ่งมีวิธีการพิสูจน์สิทธิ์ที่รองรับ ดังต่อไปนี้

- ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv
- ตรวจสอบสิทธิ์ด้วยวิธี RSA
- ตรวจสอบสิทธิ์จากไฟล์ .rhosts และวิธี RSA
- ตรวจสอบสิทธิ์จากรหัสผ่าน (password)

ถ้าเซิร์ฟเวอร์ได้รับวิธีการพิสูจน์สิทธิ์จากไคลเอ็นต์ หากรองรับวิธีการจะส่งแพ็กเก็ตชนิด SSH_SMSG_SUCCESSS แต่หากไม่รองรับจะส่งแพ็กเก็ตชนิด SSH_SMSG_FAILURE เวลาสำหรับการพิสูจน์สิทธิ์จะกำหนดไว้ไม่ให้เกิน 5 นาที (ถ้าหากเกิน เซิร์ฟเวอร์จะยกเลิกการติดต่อ) สำหรับวิธีการตรวจสอบสิทธิ์ต่างๆมีรายละเอียดดังนี้

4.4.1 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RHOTS ตามด้วยชื่อของผู้ใช้ฝั่งไคลเอ็นต์ เมื่อเซิร์ฟเวอร์บนระบบ UNIX ได้รับ ก็ตรวจสอบจากไฟล์ /etc/host.equiv และจากไฟล์ .rhosts ในไดเรกทอรีของผู้ใช้เอง การติดต่อแบบนี้ต้องใช้พอร์ตเฉพาะพิเศษ

กรรมวิธีการพิสูจน์สิทธิ์ด้วยระบบที่เชื่อถือฝั่งรีโมตนี้ (trusts the remote host) สามารถที่จะแอบอ้างข้อมูลได้โดยวิธีการปลอมแปลงเลข IP แต่จะถูกป้องกันไว้จากโพรโตคอล (เพราะทุกๆ แพ็กเก็ต เข้ารหัสอยู่)

4.4.2 ตรวจสอบสิทธิ์ด้วยวิธี RSA

สำหรับขั้นตอนและวิธีการในการตรวจสอบสิทธิ์ด้วยวิธี RSA มีดังนี้คือ

- ไคลเอ็นต์ส่ง SSH_CMSG_AUTH_RSA ตามด้วยพลับบลิคคีย์โมดูลัสของไคลเอ็นต์ (ต่างจากวิธี .rhosts + RSA ที่ส่งค่ากุญแจสาธารณะทั้งหมดไป แต่วิธีนี้ส่งเฉพาะค่า n ที่เป็นโมดูลัส ของพลับบลิคคีย์)
- เซิร์ฟเวอร์ตอบกลับทันทีด้วย SSH_SMSG_FAILURE ถ้ามันไม่พบคีย์ของการพิสูจน์สิทธิ์ของไคลเอ็นต์นี้ในตัวมัน (ค่ากุญแจสาธารณะทั้งหมดของไคลเอ็นต์) ตรงกันข้ามมันจะสร้าง challenge ขึ้นมาเพื่อเข้ารหัสด้วยกุญแจสาธารณะของไคลเอ็นต์
- เมื่อไคลเอ็นต์ได้รับจะถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับ session ID ที่มีอยู่ มาคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE
- เมื่อเซิร์ฟเวอร์ได้รับจะตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge + session ID => ทำ MD5) ถ้าถูกต้องจะส่ง SSH_SMSG_SUCCESS ในทางตรงกันข้ามจะส่ง SSH_SMSG_FAILURE และปฏิเสธการพิสูจน์สิทธิ์นี้

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

เป็นวิธีที่เพิ่มจากวิธีการตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/host.equiv โดยมีการพิสูจน์สิทธิ์แบบ RSA ด้วย โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RHOSTS_RSA ตามด้วยชื่อของผู้ใช้และกุญแจสาธารณะของไคลเอ็นต์

- เริ่มแรก เซิร์ฟเวอร์ตรวจสอบตามปกติเช่นเดียวกับตรวจสอบจากไฟล์ .rhosts และ /etc/host.equiv ถ้าไม่ถูกต้องจะส่ง SSH_SMSG_FAILURE กลับมา ตรงกันข้ามมันจะตรวจสอบกุญแจสาธารณะในตัว ถ้าหากไม่พบจะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อ
- ถ้าเซิร์ฟเวอร์รู้โฮสต์คีย์ของเครื่องไคลเอ็นต์ มันจะตรวจสอบจากโฮสต์คีย์ที่ได้มาว่าตรงกันหรือไม่ ถ้าไม่จะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อเช่นกัน
- ถ้าหากถูกต้องเซิร์ฟเวอร์จะส่ง SSH_SMSG_AUTH_RSA_CHALLENGE ที่บรรจุด้วย challenge ที่ถูกเข้ารหัสสำหรับไคลเอ็นต์ โดย challenge มีขนาดเป็น 32 บิตที่สุ่มขึ้นมา แล้วจัดข้อความและเข้ารหัสด้วย กุญแจสาธารณะของไคลเอ็นต์ (วิธีการเช่นเดียวกับที่เข้ารหัส session key)
- ไคลเอ็นต์เมื่อได้รับจะถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับ session ID ที่มีอยู่ มาคำนวณแบบ MD5 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งผลลัพธ์กลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE
- เมื่อเซิร์ฟเวอร์ได้รับจะตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge + session ID => ทำ MD5) ถ้าถูกต้องก็จะส่ง SSH_SMSG_SUCCESS ในทางตรงกันข้ามจะส่ง SSH_SMSG_FAILURE และปฏิเสธการพิสูจน์สิทธิ์นี้

4.4.4 ตรวจสอบสิทธิ์จากรหัสผ่าน password

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_PASSWORD ตามด้วยข้อความ password (ข้อความนี้ยังไม่มีการทำอะไร แต่ถูกทำการเข้ารหัสด้วยกลไกของแพ็กเก็ตโดยอัตโนมัติ)

เมื่อเซิร์ฟเวอร์ได้รับจะตรวจสอบ password และส่ง SSH_SMSG_SUCCESS ถ้าการพิสูจน์สิทธิ์ถูกต้อง และในทางตรงกันข้ามจะส่ง SSH_SMSG_FAILURE

4.5 ช่วงการเตรียมการ (Preparatory Operation)

หลังจากที่พิสูจน์สิทธิ์แล้ว เซิร์ฟเวอร์จะรอคำร้องขอจากไคลเอ็นต์ และจัดการกับคำสั่งที่เข้ามาและเซิร์ฟเวอร์ตอบกลับด้วย SSH_SMSG_SUCCESS เมื่อคำร้องขอที่เข้าถูกจัดการเสร็จสิ้น แต่ในกรณีตรงกันข้ามเมื่อคำร้องขอที่เข้ามานั้นไม่สามารถจัดการได้หรือว่าจัดการไม่สำเร็จจะส่ง SSH_SMSG_FAILURE ซึ่งข้อความช่วงนี้เป็นการเตรียมการสำหรับในช่วงต่อไป

4.6 Interactive Session และ Exchange of Data

ในช่วง interactive session ข้อมูลทุกตัวถูกเขียนโดย Shell หรือคำสั่งที่ทำงานอยู่บนเครื่องเซิร์ฟเวอร์เพื่อส่งออกไปยังส่วนแสดงผลหรือส่วนแสดงข้อผิดพลาดบนเครื่องไคลเอ็นต์ ส่วนอินพุตมาจากส่วนรับข้อมูลบนเครื่องไคลเอ็นต์ ซึ่งส่งไปยังโปรแกรมบนเครื่องเซิร์ฟเวอร์

การแลกเปลี่ยนข้อมูลทั้งหมดเป็น อะซิงโครนัส (asynchronous) โดยการส่งสามารถเกิดขึ้นได้ทุกเวลา และไม่ต้องมีการตอบสนอง (TCP/IP ปกติมีการสร้างความน่าเชื่อถืออยู่ในตัวแล้ว และแพ็กเก็ตโพรโตคอลจะป้องกันการเข้ามายุ่งหรือทำการปลอมแปลง IP)

เมื่อไคลเอ็นต์ได้รับ EOF จากส่วนรับข้อมูล มันจะส่ง SSH_CMSG_EOF การแลกเปลี่ยนข้อมูลและ interactive mode สิ้นสุดลงเมื่อเซิร์ฟเวอร์ส่ง SSH_SMSG_EXITSTATUS เพื่อที่แสดงว่าการติดต่อกับตัวไคลเอ็นต์ได้สิ้นสุดลง ส่วนไคลเอ็นต์หรือเซิร์ฟเวอร์สามารถยกเลิกการติดต่อ เมื่อไรก็ได้โดยส่งข้อความ SSH_MSG_DISCONNECT หรือปิดการเชื่อมต่อ(close the connection)

4.7 การยกเลิกการติดต่อ (Termination of the Disconnection)

โดยปกติการยกเลิกการติดต่อเริ่มโดยเซิร์ฟเวอร์ซึ่งส่ง SSH_SMSG_EXITSTATUS หลังจากที่โปรแกรมจบลง ทางฝั่งไคลเอ็นต์ตอบสนองกับข้อความที่ได้รับมาดังกล่าว โดยการส่ง SSH_CMSG_EXIT_CONFIRMATION และปิดซ็อกเก็ตของตัวเองลง ส่วนเซิร์ฟเวอร์เมื่อได้รับข้อความจึงค่อยปิดซ็อกเก็ตลง เป้าหมายสำคัญสำหรับการยืนยันในการยกเลิกคือ ในบางระบบอาจสูญเสียข้อมูลที่ส่งไปก่อนหน้านี้เมื่อซ็อกเก็ตถูกปิด และการยกเลิกในฝั่งไคลเอ็นต์ก่อนทำให้เกิด TCP/IP TIME_WAIT[RFC 0793] ทำให้เซิร์ฟเวอร์รอรับคำตอบจากฝั่งไคลเอ็นต์ที่ไม่ได้ใช้ และเซิร์ฟเวอร์ต้องสูญเสียทรัพยากรไป

ถ้าในระหว่างโปรแกรมมีสัญญาณที่ทำให้ยกเลิก ฝั่งตัวเซิร์ฟเวอร์จะส่งแพ็กเก็ตชนิด SSH_MSG_DISCONNECT พร้อมกับข้อความที่เกี่ยวข้อง ถ้าการติดต่อถูกปิด file descriptor ที่ชี้ไปยังโปรแกรมถูกปิดลงและเซิร์ฟเวอร์จะ exit แต่ถ้าโปรแกรมรันบน tty ตัว kernel จะส่งสัญญาณ SIGHUP เมื่อฝั่ง pty master ถูกปิด

บทที่ 5

ภาษาจาวา

ภาษาจาวาสร้างขึ้นโดยบริษัทซันไมโครซิสเต็มส์ ประกาศให้เป็นภาษาที่ทำงานบนอินเทอร์เน็ต โดยใช้ข้อได้เปรียบที่ว่า โปรแกรมจาวาที่สร้างบนเครื่องหนึ่งสามารถนำไปทำงานได้กับเครื่องอื่นๆ แม้ว่าแต่ละเครื่องอาจมีทรัพยากรที่แตกต่างกัน เช่น มีหน่วยประมวลผลที่ไม่เหมือนกัน หรือหน่วยควบคุมส่วนกลาง(CPU) คนละรุ่น หรือคนละยี่ห้อก็ตาม ไม่ต้องมีการคอมไพล์โปรแกรมใหม่ จึงเปลี่ยนบทบาทของอินเทอร์เน็ตไปโดยสิ้นเชิง จากคุณสมบัติอันโดดเด่นดังกล่าวจึงทำให้ อินเทอร์เน็ตกลายเป็นอันหนึ่งอันเดียวมากขึ้น เพราะสิ่งที่ส่งมาทางอินเทอร์เน็ตไม่จำเป็นจะต้องมีการแยกกันอยู่ระหว่างข้อมูลกับโปรแกรมอีกต่อไป นั้นคือ เว็บเพจทั้งหลายไม่ใช่เอกสารที่ทำงานแบบ Passive แต่จะเป็นเอกสารที่ทำงานได้แบบ Active

สิ่งที่ทำให้จาวาเป็นสิ่งที่น่าตื่นเต้นอีกประการคือนำแนวคิดของการเขียนโปรแกรมเชิงวัตถุเอามาใช้กับภาษาจาวา ประกอบด้วยเทคโนโลยีการเขียนโปรแกรมด้วยคอมโพเน็นต์ (Component) และวิชวลโปรแกรมมิง (Visual programming) จึงทำให้แนวโน้มของภาษาจาวาค่อนข้างสดใส

5.1 คุณสมบัติของภาษาจาวา

1. ไม่ผูกติดกับเครื่องใดเครื่องหนึ่ง (Platform independent) คือ ภาษาจาวาสามารถทำงานได้ทุกสภาพแวดล้อมที่คอมพิวเตอร์เครื่องนั้นๆ มีทรัพยากรที่แตกต่างกัน เพียงแต่ต้องมีการติดตั้งอินเทอร์พรีเตอร์ที่ให้การสนับสนุนเท่านั้นก็สามารถทำงานได้ โดยการทำงานของอินเทอร์พรีเตอร์จะรันโปรแกรมที่มีส่วนขยายเป็น .Class ที่ได้รับการคอมไพล์ด้วยคอมไพเลอร์ของจาวาคือ Java.exe หรือบางครั้งอาจเรียกไฟล์ที่มีส่วนขยายเป็น .Class ว่าโปรแกรมไบต์โค้ด (Byte code) เพราะมีขนาดเพียง 1 ไบต์ทุกคำสั่ง
2. เป็นภาษาที่เรียนรู้แล้วเข้าใจง่าย คือ มีกลไกของภาษาจำนวนไม่มาก และไม่ซับซ้อนเพื่อให้ผู้ใช้สามารถเรียนรู้ได้อย่างรวดเร็ว ไวยากรณ์ส่วนใหญ่นำมาจากภาษา C++ เพื่อให้ผู้ที่คุ้นเคยกับภาษา C และ C++ ไม่ต้องศึกษาไวยากรณ์ ภาษาจาวาตัดความซับซ้อนของภาษา C++ บางอย่างออกไปแต่ไปเพิ่มความสามารถให้คอมไพเลอร์จัดการให้แทน เช่นภาษาจาวาไม่พรีโพรเซสเซอร์คอมมานด์ (Preprocessor commands) ฟังก์ชันโพรโตไทป์ (Function Prototype) และเฮดเดอร์ไฟล์ (Header File) และเมื่อภาษาจาวาเป็นภาษาเชิงวัตถุแล้วจึงไม่จำเป็นต้องมีโครงสร้างอย่างเช่น สตรักเจอร์ (Structure), ยูเนียน (Unions), บิตฟิลด์ (Bit fields), และ enumerated type รวมทั้งการทำ Typedef ก็ไม่มีความจำเป็นด้วย และการที่ทำการออกแบบภาษาให้มีความรอบคอบกว่า C++ มาก ดังที่เห็นได้จากกลไกการทำมัลติเพิลอินเฮอริแทนซ์ (Multiple inheritance) และกลไลที่ทำให้เสียแบบแผนการเขียนโปรแกรมเชิงวัตถุที่ดี อย่างเช่น Friend method และ Goto statement ก็ตัดออกไปด้วยและให้การสนับสนุนการเขียนโปรแกรมประเภท กราฟฟิกยูสเซอร์อินเทอร์เฟซ มัลติทาสกิง เน็ตเวิร์กโปรแกรมมิง และปลั๊กอินโปรแกรม ช่วยให้ผู้ใช้งานศึกษาได้อย่างรวดเร็ว

3. ต้องมีการดักจับความผิดพลาดที่เกิดขึ้นกับโปรแกรมให้ได้มากที่สุด และต้องไม่ล้มเหลวง่ายๆ ความหมายคือ ภาษาจาวาให้การสนับสนุนคุณสมบัติที่เรียกว่าความแข็งแกร่ง (robustness) หรือว่าความคงทนนั้นเอง
    - ภาษาจาวาใช้กลไกที่เรียกว่า เอ็กซ์เซปชันแฮนดิง (Exception handing) ซึ่งช่วยให้โปรแกรมจัดการความผิดพลาดที่เกิดขึ้นในขณะที่โปรแกรมกำลังทำงาน โดยโปรแกรมไม่ต้องล้มเหลวและหยุดลง
    - จาวาตัดกลไกของภาษาบางอย่างออกไปเพื่อให้เกิดความผิดพลาดที่น้อยลงเช่น ตัดการอ้างถึงแอดเดรสของตัวแปร และการใช้พอยน์เตอร์ (Pointer) เพื่อไม่ให้มีโอกาสได้ใช้พอยน์เตอร์ในการอ้างตำแหน่งที่ไม่เหมาะสม
    - จาวาไม่มีการควบคุมการคืนหน่วยความจำให้กับระบบ แต่มีกลไกที่เรียกว่า Automatic garbage collector ซึ่งทำหน้าที่จัดเก็บหน่วยความจำที่ไม่ได้ใช้งานคืนให้กับระบบแบบอัตโนมัติ
    - จาวาเน้นความถูกต้องของชนิดข้อมูลอย่างเคร่งครัด แต่ก็ทำให้เป็นจุดอ่อนของภาษาจาวาคือไม่มีความยืดยุ่นเกี่ยวกับชนิดของข้อมูล เช่น มีการตรวจสอบการเปลี่ยนชนิดข้อมูล (casting) ว่าถูกต้องหรือไม่ และการอ้างถึงสมาชิกในอาร์เรย์หรือสตริงอยู่ในขอบเขตที่ถูกต้องหรือไม่

4. ให้ความปลอดภัยกับเครื่องอื่นๆ ที่ติดต่อด้วยในระบบเครือข่าย คือ โปรแกรมที่เขียนด้วยจาวามักส่งไปยังเครือข่ายเพื่อไปทำงานที่เครื่องอื่นๆ โดยมีข้อกำหนดหลายอย่างที่ทำให้ไม่สามารถสร้างโปรแกรมเพื่อส่งไปทำอันตรายต่อเครื่องที่รับโปรแกรมนั้นไปทำงาน โดยความปลอดภัยแบ่งได้หลายระดับดังนี้
    - ***ระดับล่างสุด*** จาวาไม่อนุญาตให้มีการใช้ พอยน์เตอร์ อ้างหน่วยความจำได้โดยตรง และตรวจสอบขอบเขตการใช้งานของอาร์เรย์อย่างเคร่งครัด จึงทำให้ไม่มีสิทธิ์เขียนหรืออ่านหน่วยความจำที่โปรแกรมนั้นไม่มีสิทธิ์
    - ***ระดับกลาง*** ในที่โปรแกรมทำงานจะมีโปรแกรมที่เรียกว่า Byte code verifier ตรวจสอบการทำงานของโปรแกรมว่าถูกต้องหรือไม่ หากพบว่าการกระทำใดไม่สมควรก็จะปฏิเสธการกระทำนั้น
    - ***ระดับบนสุด*** มีการรักษาความปลอดภัยที่เรียกว่า Sandbox model นั่นคือโปรแกรมที่ดาวน์โหลดมาจากเครื่องอื่น ถือเป็นโปรแกรมที่ไม่น่าไว้ใจ (Untrusted codes) และนำโปรแกรมที่โหลดมานี้เก็บไว้ใน Sandbox โปรแกรมที่อยู่ในนี้มีข้อจำกัดหลายอย่างซึ่งถูกควบคุมโดย Secure Manager เช่น ไม่สามารถเขียนหรืออ่านไฟล์ได้ เป็นต้น

5.2 วิธีการแปลไวยากรณ์ของจาวา

การประมวลผลของจาวานั้นช้ากว่าภาษาอย่าง C, C++ มากเพราะต้องปฏิบัติคำสั่งด้วยอินเทอร์พรีเตอร์ ซึ่งเฉลี่ยแล้วช้ากว่าประมาณ 20 เท่า จากไฟล์ของโปรแกรมภาษาจาวาที่มีส่วนขยายเป็น .java คอมไพล์ด้วย Javac.exe เพื่อให้เป็นไฟล์ .Class และส่วนขยายนี้จะทำงานกับจาวาเวอร์ชวลแมชีน (Java Virtual Machine JVM) ภายในของ JVM มีหน่วยประมวลผลสมมุติที่เรียกว่า เวอร์ชวลโพรเซสเซอร์ ทำหน้าที่ประมวลผลคำสั่งของ JVM เมื่อมีการติดตั้ง JVM ไว้ในเครื่องที่ดาวน์โหลดโปรแกรม ที่มีส่วนขยาย .Class ไปทำงาน JVM บนเครื่องนั้นจะแมปปิง (Mapping) จากคำสั่ง ไบต์โค้ด (Byte Code)ไปเป็น เนทีฟโค้ด (Native Code) ของเครื่องนั้นๆ แล้วทำงาน

ปัจจุบันมีการค้นคว้าหลายอย่างเพื่อเพิ่มความเร็วของภาษา Java เช่นการสร้างหน่วยประมวลผลที่สามารถทำงานคำสั่งของ JVM ได้โดยตรง หรือพัฒนาอินเทอร์พรีเตอร์ซึ่งไม่ทำงานคำสั่งของ JVM ทีละคำสั่ง แต่จะเปลี่ยนโปรแกรม Java Class ทั้งโปรแกรมเป็นเนทีฟโค้ดแล้วจึงทำงานเนทีฟโค้ดนั้น วิธีการนี้เรียกว่า Just In Time (JIT) เนื่องจากคล้ายกับทำการคอมไพล์โปรแกรมก่อนจะเริ่มทำงาน โปรแกรมจะทำงานเร็วขึ้นอย่างมาก เพราะคำสั่งที่ถูกทำซ้ำบ่อยๆ เช่นคำสั่งที่อยู่ในประโยคทำซ้ำหรือฟังก์ชันที่เรียกใช้บ่อยๆ จะแปลเพียงครั้งเดียว ไม่ต้องแปลทุกครั้งที่จะทำงาน และยังทำให้นำเทคนิคการปรับปรุงโปรแกรมให้เร็วขึ้น (Optimization) ที่มีใช้ในคอมไพเลอร์ทั่วไปมาใช้ได้ด้วย

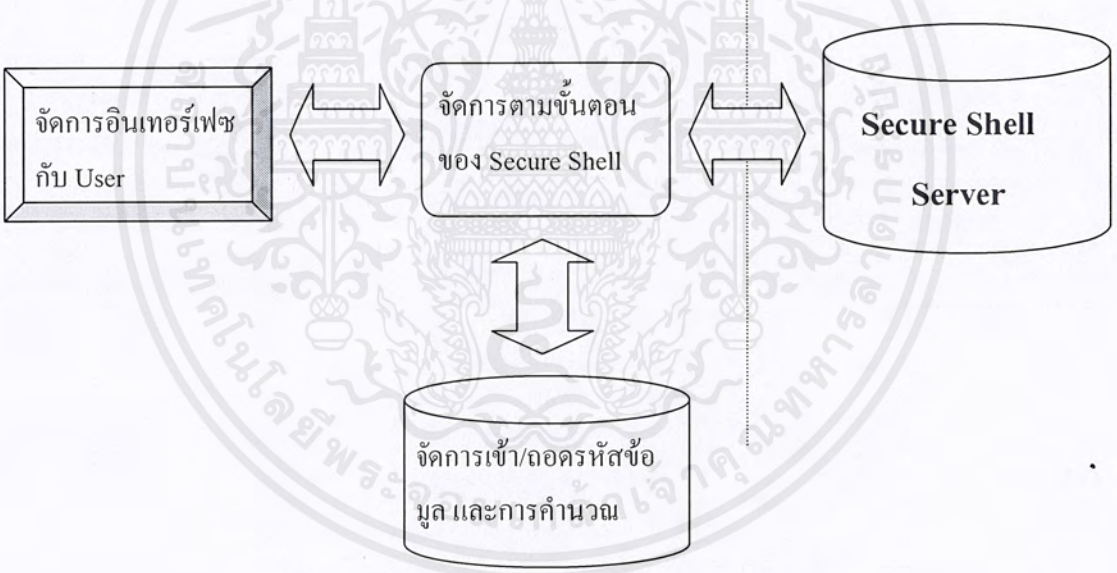
บทที่ 6

การออกแบบ

เนื่องจากการติดต่อและใช้งานบนระบบที่มี Secure Shell แตกต่างจากการติดต่อแบบปกติหลายประการ คือ ข้อมูลต่างๆ ที่ติดต่อกันจะต้องมีการรักษาความปลอดภัย โดยมีการเข้ารหัสข้อมูลที่ใช้ในการสื่อสารกัน และต้องมีการปฏิบัติตามขั้นตอนของ Secure Shell อย่างเคร่งครัดด้วย แม้ว่ามีผู้ดักจับเพื่ออ่านข้อมูลก็ไม่สามารถทราบได้ว่าเป็นข้อมูลอะไร

ดังนั้น ในการออกแบบโปรแกรมเพื่อที่จะใช้สำหรับติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ทำงานอยู่ ให้สามารถทำงานได้อย่างมีประสิทธิภาพ และสมบูรณ์นั้น จำเป็นต้องออกแบบให้ทำงานตรงตามมาตรฐานของ Secure Shell ที่ได้กำหนดไว้ โดยต้องใช้ทฤษฎีพื้นฐานจากบทต่างๆ ที่ได้กล่าวมานำมาประยุกต์และดัดแปลงสำหรับการออกแบบ ซึ่งได้แบ่งการออกแบบเป็น 3 ส่วน

1. ส่วนที่การจัดการอินเทอร์เฟซกับ User และทำตัวเองเป็นตัวเทอร์มินอลจำลอง
2. ส่วนในการจัดการข้อมูล ตามขั้นตอนของ Secure Shell
3. ส่วนในการทำการเข้ารหัส/ถอดรหัสและการคำนวณข้อมูล



รูปที่ 6.1 แสดงความสัมพันธ์ของส่วนต่างๆ ที่ออกแบบ

6.1 The Socket API

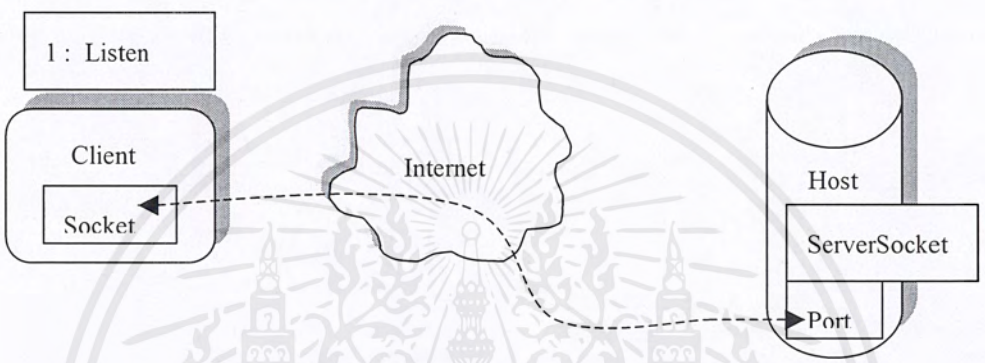
หัวใจสำคัญของการติดต่อสื่อสารกับเครื่องผู้ให้บริการในระบบ TCP/IP จำเป็นต้องทราบถึงอุปกรณ์ที่ใช้ในการติดต่อ ซึ่งอุปกรณ์นี้เรียกรวม ๆ ว่า Application Program Interface (API) ซึ่งกลุ่มของ API ที่ใช้ในการติดต่อผ่านเครือข่ายที่สำคัญคือ ซ็อกเก็ต (Socket) โดย ซ็อกเก็ตในแอพพลิเคชันนี้แบ่งออกเป็น 2 ลักษณะคือ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

เซิร์ฟเวอร์ซ็อกเก็ต (Server Socket) ทำหน้าที่รอการติดต่อจากเครื่องของผู้ขอใช้บริการ ตัวของเซิร์ฟเวอร์ซ็อกเก็ตนี้จะฝังอยู่ทางฝั่งของเซิร์ฟเวอร์ เมื่อมีผู้ขอใช้บริการร้องขอก็จะทำการค้นหาพอร์ตว่างและจับจองไว้สำหรับเป็นช่องทางการติดต่อ ตามขั้นตอนคร่าว ๆ 4 ขั้นตอนดังนี้

ขั้นตอนที่ 1 ฝั่งของผู้ใช้บริการทำการเปิดซ็อกเก็ต โดยระบุถึงชื่อของโฮสต์และพอร์ตที่ให้บริการ ในขั้นตอนนี้ขอยกตัวอย่างของโค้ดภาษาจาวา

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

จากตัวอย่างของโค้ดข้างบนจะทำให้เกิดการทำงานดังนี้



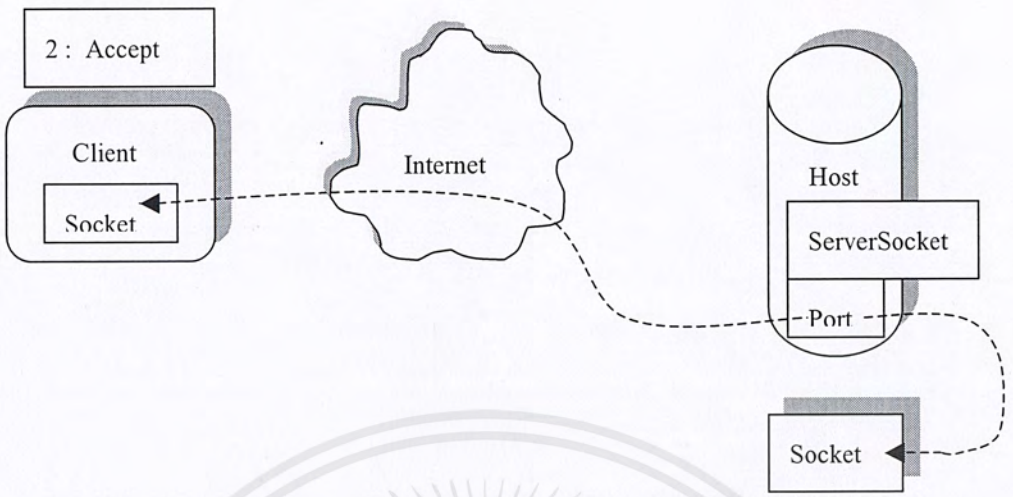
รูปที่ 6.2 ผู้ใช้บริการทำการเปิดซ็อกเก็ต

ผู้ให้บริการจะเปิดซ็อกเก็ตของตัวเองรออยู่แล้ว เมื่อผู้ใช้บริการเปิดซ็อกเก็ตจะทำให้เซิร์ฟเวอร์ค้นหาพอร์ตว่าง ๆ สำหรับสร้างช่องทางการติดต่อ

ขั้นตอนที่ 2 ผู้ใช้บริการรอ (Accept) จนกว่าเซิร์ฟเวอร์หาพอร์ตว่างได้ และจัดการกับตัวของเซิร์ฟเวอร์เองให้พร้อมสำหรับให้บริการ โค้ดข้างล่างจะแสดงการรอของฝั่งผู้ใช้บริการ

```
Try
{
 if((ClientSocket = new Socket(Host,Port)) == null)  ;request contact server and Accept
}
catch(UnknownHostException e)
{
 Display("Host "+Host+" not found\n");
}
catch(IOException e)
{
 Display(" Failed to connect to host "+Host+"\n");
}
```

จากตัวอย่างของโค้ดในขั้นตอนที่ 2 ทำให้เกิดการทำงานดังนี้



รูปที่ 6.3 ผู้ให้บริการทำการค้นหาพอร์ต และทำการเปิดพอร์ต

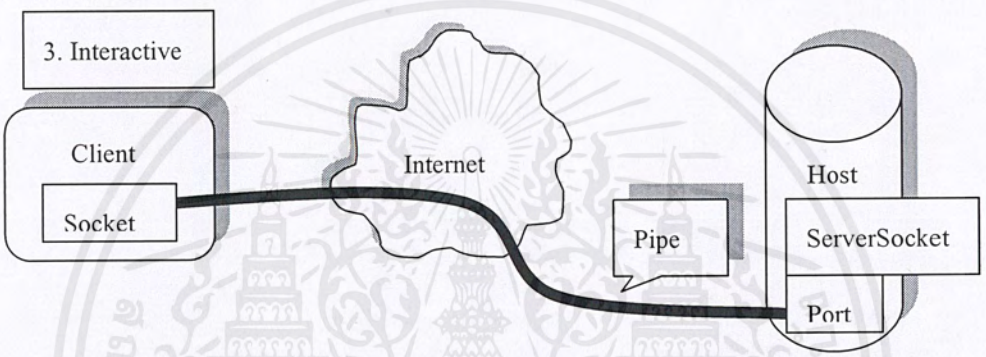
จากโค้ดข้างบนเมื่อมีการผิดพลาดประการใดจากการขอเปิดพอร์ตจะมีการดักจับความผิดพลาดด้วยเพื่อไม่ต้องการให้ระบบล้มเหลว เมื่อผู้ให้บริการพร้อมก็ส่งสัญญาณตอบรับกลับไปให้ยังผู้ใช้บริการได้ทราบเพื่อดำเนินขั้นตอนต่อไป

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

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

```
os = new BufferedOutputStream(ClientSocket.getOutputStream());  ;prepare create
Pipe and output buffer
}
catch(IOException e)
{
Display(" Failed to put stream on socket");
System.exit(5);
}
```

จากตัวอย่างของโค้ดในขั้นตอนที่ 3 ทำให้เกิดการทำงานดังนี้

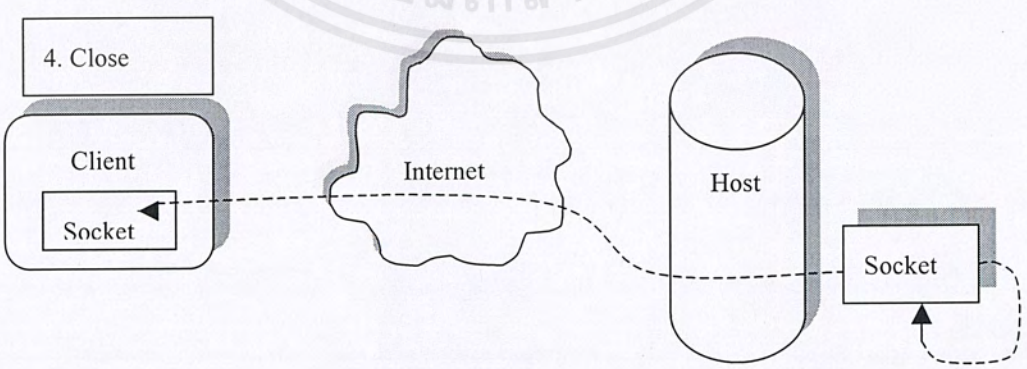


รูปที่ 6.4 แสดงการสร้างเส้นทางติดต่อ (Pipe line) และการสร้างบัฟเฟอร์

ขั้นตอนที่ 4 เป็นขั้นตอนสุดท้ายเมื่อผู้ใช้บริการต้องการปิดการติดต่อ จากโค้ดข้างล่างแสดงโปรแกรมการปิดการติดต่อ

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

จากตัวอย่างของโค้ดข้างบนทำให้เกิดการทำงานดังนี้



รูปที่ 6.5 แสดงการปิดซ็อกเก็ต

เมื่อผู้ใช้บริการทำการปิดซ็อกเก็ตตัวเองก็ส่งผลให้ผู้ให้บริการปิด Port และซ็อกเก็ตของตัวเองด้วยเช่นกัน

6.1.1 ภาพรวมของฟังก์ชันต่าง ๆ ที่ใช้สำหรับการติดต่อกับผู้ให้บริการบนระบบ TCP/IP

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

ในการใช้ซ็อกเก็ตในการติดต่อจะต้องมีการเรียกใช้ System Call โดยมีการส่งผ่านค่าพารามิเตอร์ที่จำเป็นลงไป ซึ่งจะใช้ฟังก์ชันของซ็อกเก็ต API ในการเรียก System Call ต่างๆ ดังต่อไปนี้

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

รูปที่ 6.6 แสดงลำดับการจัดการกับซ็อกเก็ตบน TCP/IP

6.2 ส่วนต่างๆ จากการออกแบบ

จากที่ได้กล่าวมาในหัวข้อของการออกแบบว่าแอพพลิเคชันนี้จะประกอบด้วย 3 ส่วนหลักๆ คือ

1. ส่วนที่การจัดการอินเทอร์เฟซกับ User และทำตัวเองเป็นตัวจำลองเทอร์มินอล
2. ส่วนในการจัดการข้อมูล ตามขั้นตอนของ Secure Shell
3. ส่วนในการทำการเข้ารหัส/ถอดรหัสและการคำนวณข้อมูล

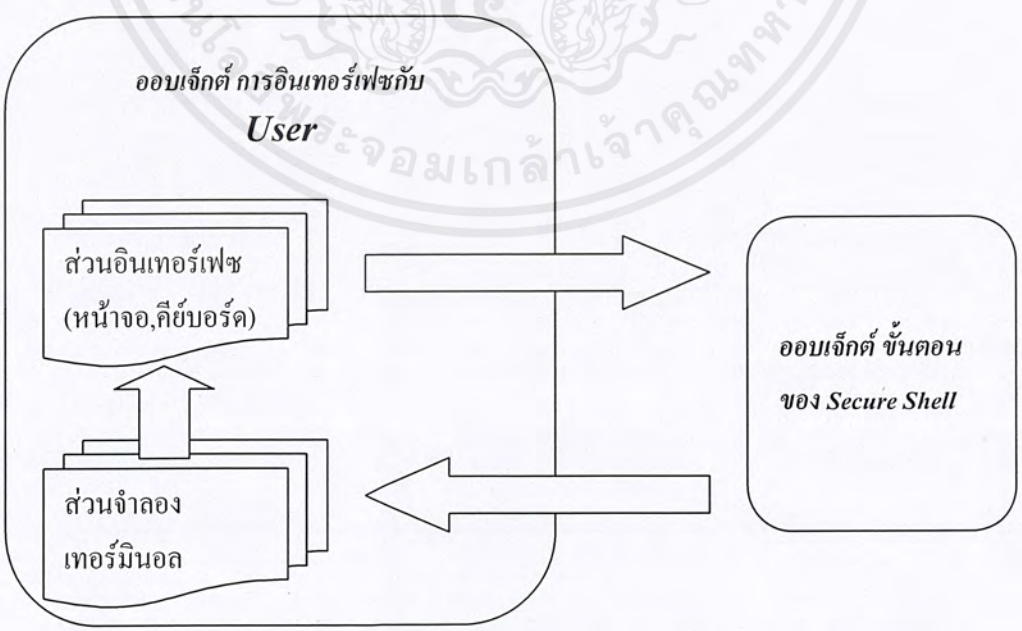
ในการออกแบบของแต่ละส่วนนั้นจะออกแบบด้วยลักษณะโครงสร้างแบบเชิงวัตถุ (Object Oriented) โดยแต่ละส่วนจะมีส่วนประกอบ และรายละเอียดต่าง ๆ ดังต่อไปนี้

6.2.1 ส่วนจัดการอินเทอร์เฟซกับ User และส่วนจำลองเทอร์มินอล

ในส่วนแรกนี้ จะเป็นการออกแบบสำหรับนำข้อมูลของผู้ใช้และข้อมูลที่ได้รับมา แสดงผลบนหน้าจอของผู้ใช้ สำหรับแอพพลิเคชันนี้เขียนด้วย Java Application ซึ่งมีคลาสที่ใช้สำหรับการสร้าง User Interface บรรจุอยู่ใน Package java.awt.* โดยมีฟังก์ชันต่าง ๆ ที่ใช้ในการสร้าง UI มากมาย

ส่วนของตัวจำลองเทอร์มินอลที่คอยให้การสนับสนุนการแสดงผลและทำการแปลความหมายของข้อมูลที่ได้รับเข้ามาจากส่วนของการจัดการตามขั้นตอนของ Secure Shell ว่าข้อมูลนั้น ๆ มีความหมายว่าอย่างไร เช่น ข้อมูลอยู่บรรทัดใด สีอะไร เป็นต้น แล้วทำการส่งผลลัพธ์ที่ได้แปลเรียบร้อยแล้ว ไปแสดงผลยังส่วนอินเทอร์เฟซ สำหรับเทอร์มินอลที่ได้ทำการออกแบบนี้เป็นโค้ดของเทอร์มินอลชนิด VT100

สำหรับการรับข้อมูลจากอินพุตที่ผู้ใช้ป้อนเข้ามา ข้อมูลต่างๆ ที่รับมาไม่จำเป็นต้องผ่านตัวจำลองเทอร์มินอล ซึ่งจะส่งข้อมูลส่วนนี้ไปประมวลผลตามขั้นตอนของ Secure Shell อัตโนมัติ ในส่วนนี้จะมีออบเจ็กต์ที่ใช้ในการจัดการจำลองเทอร์มินอล คือ SSH_Terminal



รูปที่ 6.7 แสดงการติดต่อของข้อมูลในส่วนจัดการอินเทอร์เฟซและจำลองเทอร์มินอล

ในออบเจ็กต์นี้ประกอบด้วยออบเจ็กต์การทำงานต่างๆ เพื่อใช้ในการอินเทอร์เฟซกับผู้ใช้และเทอร์มินอลจำลองดังนี้

1. **ออบเจ็กต์ DATA_TERMINAL** จะเป็นออบเจ็กต์ชนิด Final ซึ่งไม่สามารถที่จะเปลี่ยนแปลงข้อมูลใดๆ ของออบเจ็กต์นี้ได้ สร้างไว้เพื่อทำหน้าที่เก็บตัวแปลโกลบอล ที่ใช้งานร่วมกันหมดของแอพพลิเคชันนี้ โดยข้อมูลจะประกอบไปด้วย ค่าคงที่ต่าง ๆ ของรหัส vt100 เช่น ESC, Backet, DELL, BELL เป็นต้น

2. **ออบเจ็กต์ Display** เป็นออบเจ็กต์ที่ขยายมาจากออบเจ็กต์ DATA_TERMINAL เพื่อต้องการที่จะใช้ค่าคงที่ต่าง ๆ จากออบเจ็กต์ที่ขยายมา หน้าที่ของออบเจ็กต์นี้จะเป็นตัวจำลองเทอร์มินอลในมาตรฐานของรหัส vt100 ในการตอบสนองกับ User โดยประกอบไปด้วยฟังก์ชันการทำงานต่างๆ ดังนี้

   - ***Start*** เป็นฟังก์ชันสำหรับการการเริ่มต้นการเซตขนาดของ font และเทอร์มินอล
   - ***ClearCoordinateScreen*** ทำหน้าที่ลบย่านของข้อมูลที่แสดงบนเทอร์มินอลและบัฟเฟอร์โดยสามารถที่จะระบุถึงตำแหน่งเริ่มต้นและสุดท้ายตามที่เราต้องการได้
   - ***ScrollUp, ScrollDown*** ทำหน้าที่เลื่อนข้อมูลบนเทอร์มินอลและ Buffer ขึ้นหรือลงทั้งหมดโดยเลื่อนทีละหนึ่งครั้ง
   - ***SetGraphicsColor*** ทำหน้าที่เปลี่ยนสีของตัวอักษรที่จะแสดงบนเทอร์มินอล
   - ***DrawString*** ทำหน้าที่เขียนข้อความลงบนเทอร์มินอลและบัฟเฟอร์ตามที่ได้รับเข้ามา
   - ***ClearScreenAll*** ทำหน้าที่ลบข้อมูลทั้งหมดที่อยู่บนเทอร์มินอลและบัฟเฟอร์
   - ***LineFeed*** ทำหน้าที่เลื่อนข้อมูลบนเทอร์มินอลขึ้น 1 บรรทัด
   - ***Run*** ควบคุมการเขียนข้อมูลลงบนเทอร์มินอลและเนื่องจากตัวมันเองเป็นเธรดจึงทำหน้าที่ในการอัพเดต ข้อมูลบนเทอร์มินอลตามเวลาที่กำหนดไว้
   - ***CheckCodeFirstFormString*** ทำหน้าที่ในการตรวจสอบข้อมูลตัวแรกที่ได้รับมาว่าเป็นรหัสชนิดอะไร เช่น ESC หรือ [(Backet) เป็นต้น
   - ***MapfontThai*** ทำหน้าที่ในการเปรียบเทียบรหัสของแอสกีที่เข้ามาว่าเป็นพญัชนะตัวใดในภาษาไทย และทำการเขียนลงบนเทอร์มินอลโดยเป็นอักษรภาษาไทย
   - ***FONT_THAI*** ไม่ใช่ฟังก์ชันการทำงานแต่เป็นหัวใจในการเขียนภาษาไทยบนเทอร์มินอลเพราะมีหน้าที่เก็บข้อมูลที่เป็นภาษาไทย

3. **ออบเจ็กต์ Xterminal** หน้าที่หลักๆ ของ ออบเจ็กต์นี้คือสร้างรูปแบบของ User Interface (UI) ให้สะดวกและเหมาะสมกับการใช้งาน เช่นการจัดวางหน้าตาของปุ่มต่างๆ ที่ใช้ทำงาน และหน้าที่อีกอย่างที่สำคัญคือ คอยจัดการกับเหตุการณ์ (Event) ต่างๆ ที่เกิดขึ้นกับแอพพลิเคชันนี้ ขณะ

กำลังทำงานอยู่ เช่นการกดคีย์บอร์ด หรือการคลิกเมาส์เป็นต้น โดยมีฟังก์ชันการทำงานต่าง ประกอบไปด้วย

- ***Xterminal*** ซึ่งเป็น Constructor ที่ทำหน้าที่ในการจัดเลย์เอาต์ (Layout) ของตัวเทอร์มินอล ให้เหมาะสมกับการใช้งาน
- ***HandleEvent*** ทำหน้าที่ดักจับเหตุการณ์ต่างๆ ที่พิเศษออกไป เช่น การปิดการทำงานของเทอร์มินอล หรือ การกดคีย์พิเศษอื่นๆ อย่างเช่นการกดคีย์คอนโทรลกับคีย์ใดๆ เป็นต้น
- ***InitialConnect*** ทำหน้าที่ตอบสนองเมื่อเริ่มต้นการติดต่อกับผู้ให้บริการ
- ***Action*** ทำหน้าที่ในการดักจับเหตุการณ์ที่เกิดจากการกดปุ่มต่างๆ บนเทอร์มินอล และส่งการควบคุมที่เหมาะสมไปยังฟังก์ชันต่างๆ เพื่อให้ทำงาน เช่น การกดปุ่ม Connect จะทำให้มีการใช้ฟังก์ชัน InitialConnect สำหรับเริ่มการติดต่อกับผู้ให้บริการเป็นต้น
- ***KeyDown*** ทำหน้าที่คอยดักจับคำสั่งที่ส่งมาจากคีย์บอร์ด และส่งคำสั่งที่ได้เหล่านี้ไปควบคุม หรือนำไปเข้ารหัสเพื่อจะส่งไปให้ยังผู้ให้บริการ ซึ่งจะกล่าวในส่วนต่อๆ ไป

4. **ออบเจ็กต์ SshTerminal** หน้าที่ของออบเจ็กต์นี้จะทำการแปลรหัสของ VT100 ที่ได้รับมาจากผู้ให้บริการเปลี่ยนเป็นคำสั่งที่ใช้ควบคุมเทอร์มินอลของผู้ใช้ โดยมีฟังก์ชันการทำงานดังนี้

- ***ChangeAttribute*** ทำหน้าที่เปลี่ยนคุณสมบัติบางประการของเทอร์มินอล เช่น เปลี่ยนสีของ Fore ground เป็นต้น
- ***DisplayChar*** ทำหน้าที่ตัดคำของรหัสควบคุมต่างๆ ที่ได้รับมา ออกมาแสดงผลยังเทอร์มินอลให้เหมาะสมตามรหัสควบคุมนั้น ๆ

6.2.2 ส่วนจัดการข้อมูลตามขั้นตอนของโพรโตคอล Secure Shell

ในส่วนนี้จะต้องเขียนโปรแกรมขึ้นเพื่อจัดการกับข้อมูลตามมาตรฐาน Secure Shell และต้องสอดคล้องกับการทำงานของส่วนอินเทอร์เฟซด้วย และคุณสมบัติที่ดีอีกประการหนึ่งของแอพพลิเคชันนี้คือจะมีการตรวจจับความผิดพลาดที่คาดไม่ถึง ซึ่งอาจจะทำให้ระบบล้มเหลวได้ ในส่วนจัดการข้อมูลตามขั้นตอนของโพรโตคอล Secure Shell นี้ประกอบไปด้วยออปเจ็กต์ต่างๆ ดังนี้

1. **ออบเจ็กต์ SshIO** หน้าที่สำคัญคือ เป็นออบเจ็กต์ที่ติดต่อกับผู้ให้บริการจริงๆ โดยการเปิดซ็อกเก็ต (Socket) และรอการตอบสนองจากผู้ให้บริการรวมทั้งการปิดซ็อกเก็ตด้วย และหน้าที่สำคัญอีกอย่างคือใช้รับและส่งแพ็กเก็ตผ่านโพรโตคอล TCP/IP ประกอบด้วยฟังก์ชันที่สำคัญ คือ

- ***Connect*** เป็นฟังก์ชันสำหรับเปิดการติดต่อเพื่อขอเข้าใช้บริการจากเซิร์ฟเวอร์ และหากมีการผิดพลาดประการใด ๆ ก็ตามอย่างเช่น ไม่สามารถติดต่อกับผู้ให้บริการได้ โปรแกรมจะมีความสามารถที่จะดักจับข้อผิดพลาดที่เกิดขึ้นได้

- ***Disconnect*** ทำหน้าที่ปิดซ็อกเก็ตที่ติดต่ออยู่ และปิดสายของสตรีมหรือไปป์ของข้อมูลด้วย
- ***Run*** เป็น Thread ที่ทำหน้าควบคุมการขั้นตอนการทำงานทั้งหมดของ Secure Shell เช่น ควบคุมขั้นตอนการแลกคีย์ ขั้นตอนการถอดรหัส ขั้นตอนการแลกเปลี่ยนข้อมูล และฟังก์ชันนี้จะทำงานตลอดไปจนกว่าจะจบการทำงานหรือยกเลิกการติดต่อ
- ***Sent*** ทำหน้าที่ส่งข้อมูลแบบไม่มีการเข้ารหัสไปยังผู้ให้บริการ
- ***GetVersionString*** ทำหน้าที่อ่านและแปลข้อมูลจากผู้ให้บริการในช่วงของการพิสูจน์เวอร์ชัน โดยข้อมูลนี้จะประกอบไปด้วยส่วนต่างๆ เช่น เวอร์ชันของ Secure Shell ที่ใช้งาน และหมายเหตุสั้นๆ
- ***ReceiveServerData*** ทำหน้าที่รับข้อมูลต่อจากช่วงของการพิสูจน์เวอร์ชัน โดยฟังก์ชันนี้จะทำการวิเคราะห์ข้อมูลที่เข้ามาและแยกข้อมูลออกเป็นชนิดต่างๆ เช่น ข้อมูลความยาว ข้อมูล Padding ข้อมูลสำหรับใช้ในการตรวจสอบ CRC เป็นต้น
- ***IsCipherSupported*** ทำหน้าที่ตรวจสอบว่าเป็นการเข้ารหัสชนิดใดและรองรับหรือไม่ เช่น การเข้ารหัสชนิด RSA,DES,3DES เป็นต้น
- ***GenerateSessionId*** ทำหน้าที่ให้กำเนิด Session Id โดยการนำเอา โฮสต์คีย์(host key)ต่อกับเซิร์ฟเวอร์คีย์(server key)และต่อกับคุกกี้(cookie key) แล้วนำมาเข้าฟังก์ชัน MD5 เพื่อให้เหลือ 16ไบต์
- ***GenerateSessionKey*** ทำหน้าที่ให้กำเนิดข้อมูลที่เป็น Session key มีขนาด 32 ไบต์
- ***SecureRandom*** ทำหน้าที่ให้กำเนิดข้อมูลที่สุ่มขึ้นมาตามความยาวที่ต้องการ
- ***InitCipher*** ทำหน้าที่ระบุถึงชนิดของการเข้ารหัสที่ต้องการและทำการเซ็ตค่าของคีย์ที่ต้องการเข้าและถอดรหัส
- ***SendSessionKey*** ทำหน้าส่งคีย์ที่คำนวณได้กลับไปยังผู้ให้บริการ คีย์ที่ส่งไปประกอบไปด้วย ชนิดของการเข้ารหัส , cookie , Session key , Protocol เป็นต้น
- ***IsSuccess*** ทำหน้าที่ตรวจสอบว่าการส่งข้อมูลในแต่ละครั้งนั้นเกิดข้อผิดพลาดหรือไม่ ถ้ามีข้อผิดพลาด ทางผู้ให้บริการจะทำการส่งชนิด FAILURE กลับมา ถ้าไม่มีข้อผิดพลาดประการใดผู้ให้บริการก็จะส่งข้อมูลชนิด SUCCESS กลับมา

2. **ออบเจ็กต์ SSHBufferInputStream และ SSHBufferOutputStream** ใช้สร้างพื้นที่สำหรับเก็บข้อมูลที่เรียกว่า Buffer สำหรับรองรับข้อมูลที่ใช้รับและส่งให้มีขนาดเพียงพอ ช่วยแบ่งเบาภาระของ IO อีกทางหนึ่งด้วยเพราะไม่จำเป็นต้องมีการติดต่อกับ IO ทุกครั้งเมื่อมีการกดคีย์ใดๆ ประกอบไปด้วยฟังก์ชันต่าง ๆ ดังนี้
   - ***PduByteArrayInputStream*** จัดเตรียมข้อมูลที่มีลักษณะเป็นไบต์อาร์เรย์
   - ***Checksum*** ทำหน้าที่ตรวจสอบข้อมูลในรูปแบบของ CRC

- ***readFrom*** ทำหน้าที่อ่านข้อมูลออกมาจากบัฟเฟอร์ พร้อมทั้งถอดรหัสและทำการเช็ค CRC ด้วย
- ***writeTo*** เขียนข้อมูลที่ได้เข้ารหัสแล้วลงยังบัฟเฟอร์พร้อมกับทำการส่งไปยังผู้ให้บริการด้วย

3. **ออบเจ็กต์ DataInputStream และ DataOutputStream** ทำหน้าที่เขียนและอ่านข้อมูลชนิดต่าง ๆ ลงไปยังบัฟเฟอร์ โดยออบเจ็กต์นี้จะถูกเรียกใช้งานโดย ออบเจ็กต์ SSHBufferInputStream โดยมีฟังก์ชันการทำงานดังนี้
   - ***readBigInteger*** *และ* ***writeBigInteger*** ทำหน้าที่อ่านและเขียนข้อมูลให้อยู่ในรูปของ BigInteger
   - ***readString*** *และ* ***writeString*** ทำหน้าที่เขียนและอ่านข้อมูลที่อยู่ในรูปของไบต์อาร์เรย์ให้อยู่ในรูปของสตริง

6.2.3 ส่วนในการเข้ารหัส/ถอดรหัสและการคำนวณของข้อมูล

ในแอพพลิเคชันนี้จะประกอบด้วยออบเจ็กต์ของการเข้ารหัส การถอดรหัสและการคำนวณ ข้อมูลต่างๆ โดยมีออบเจ็กต์ต่างๆ ดังนี้

1. **ออบเจ็กต์ DES** ทำหน้าที่ในการเข้าและถอดรหัส โดยสามารถที่จะเซ็ตจำนวนของคีย์ที่ต้องการเข้ารหัสได้แต่จะต้องมีขนาดเท่าของ 8 เช่น 8, 16, 32, 64 เป็นต้น ฟังก์ชันต่างๆ มีดังนี้
   - ***encrypt*** ทำหน้าที่เข้ารหัสข้อมูล
   - ***decrypt*** ทำหน้าที่ถอดรหัสข้อมูล
   - ***setKey*** กำหนดขนาดของคีย์ที่ทำการเข้ารหัส

2. **ออบเจ็กต์ 3DES** ทำหน้าที่ในการเข้ารหัสเช่นเดียวกับ DES แต่จะใช้คีย์ที่แตกต่างกันในการเข้ารหัสแต่ละครั้ง และทำการเข้ารหัสทั้งหมด 3 ครั้ง ฟังก์ชันต่าง ๆมีดังนี้
   - ***encrypt*** ทำหน้าที่เข้ารหัสข้อมูล 3 ครั้ง คือ คีย์1 คีย์2 และคีย์3 ตามลำดับ
   - ***decrypt*** ทำหน้าที่ถอดรหัสข้อมูล 3 ครั้งเช่นเดียวกันแต่กลับกันคือ คีย์3 คีย์2 และคีย์1 ตามลำดับ
   - ***setKey*** กำหนดขนาดของคีย์ที่ทำการเข้ารหัส แต่คีย์ควรมีขนาดเป็นเท่าของ 8 เช่น 8, 16, 32, 64 เป็นต้น

3. **ออบเจ็กต์ต่างๆ ที่สนับสนุนการเข้ารหัสแบบ RSA**
   - ***PrivateKey*** เก็บข้อมูลชนิด Private key
   - ***PublicKey*** เก็บข้อมูลชนิด Public key
   - ***RSAPrivateKey*** เก็บข้อมูลของ RSA ที่เป็นชนิด Private Key

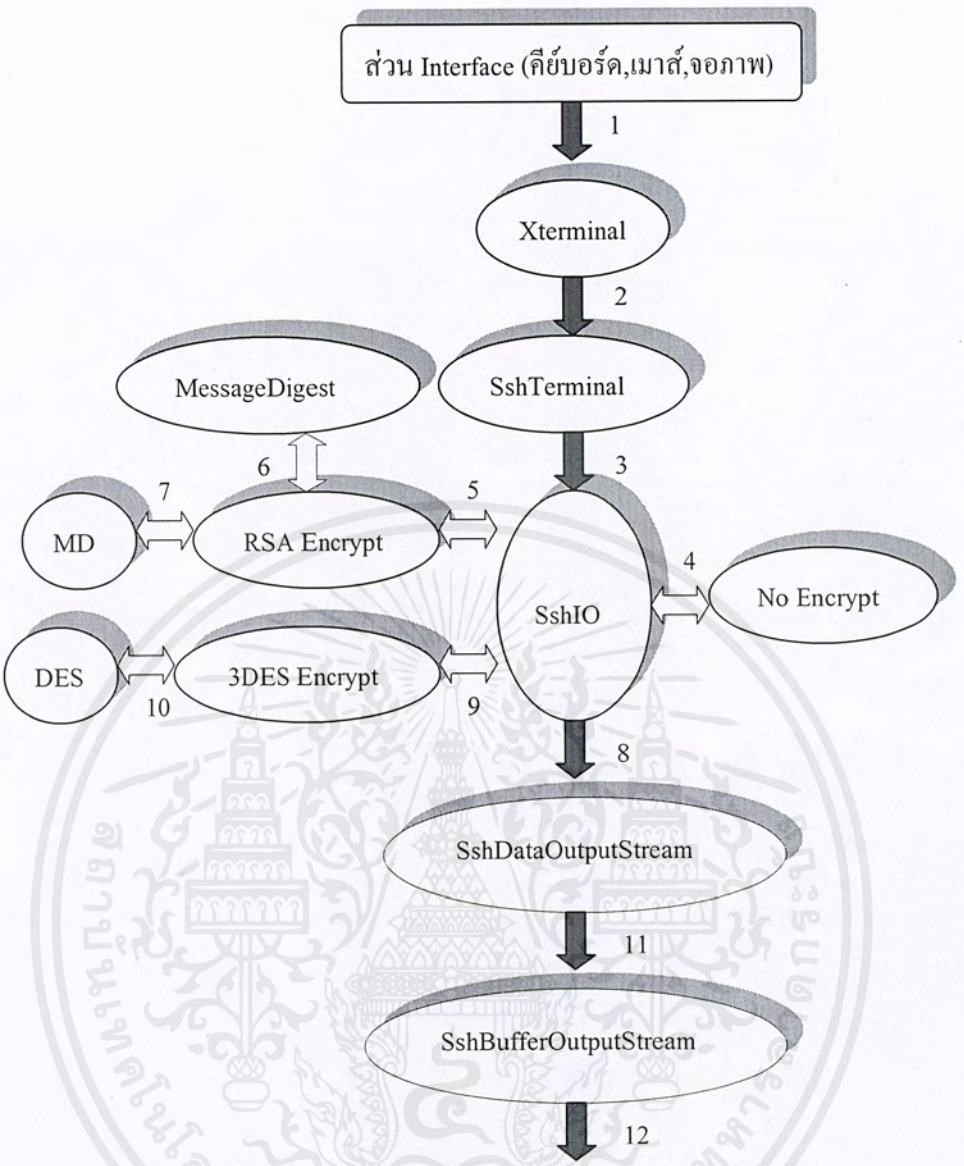
- ***RSAPublicKey*** เก็บข้อมูลของ RSA ที่เป็นชนิด Public Key
- ***RSACipher*** ทำหน้าที่คำนวณแบบ RSA
- ***RSAKey*** ทำหน้าที่ในการอำนวยความสะดวกต่างๆ เช่น การดึงคีย์ขึ้นมาใช้งานหรือ เก็บค่าต่าง ที่จำเป็นไว้
- ***MD5*** ใช้ในการคำนวณแบบ MD5 ซึ่งจะทำให้ข้อมูลต่างๆ ที่เข้ามามีขนาดเหลือเพียง 16 ไบต์ (256 บิต)
- **Cipher** ทำหน้าที่สร้าง Instance ของการเข้ารหัสชนิดต่าง ๆ ตามต้องการ
- **KeyPair** ทำหน้าที่เก็บคีย์คู่ซึ่งประกอบไปด้วย Private และ Public key เข้าด้วยกันเพื่อสะดวกและเป็นระเบียบกับการใช้งาน
- ***SecureRandom*** ทำหน้าที่สุ่มค่าที่ปลอดภัยสำหรับการทำ Session key และใช้สำหรับเป็น Padding ด้วย

6.3 ทิศทางของข้อมูลและการทำงานของออบเจ็กต์ต่าง ๆ ในการทำงานจริง

ในหัวข้อนี้จะไม่นำเสนอถึงออบเจ็กต์ไดอะแกรมต่าง ๆ ที่สร้างขึ้นเพราะมีรายละเอียดของการออกแบบมากแต่จะได้แสดงไว้ในส่วนของภาคผนวก ในส่วนนี้จะแสดงถึงทิศทางการไหลของข้อมูลที่เกิดในแต่ละออปเจ็กต์ที่ทำงานขณะที่แอพพลิเคชันนี้ทำงานจริง ๆ โดยจะแบ่งทิศทางการไหลของข้อมูลออกเป็น 2 ทิศทางคือ ทิศทางของข้อมูลที่ส่งไปยังผู้ให้บริการ และข้อมูลที่ได้รับจากผู้ให้บริการ โดยมีรายละเอียดดังรูปที่ 6.8

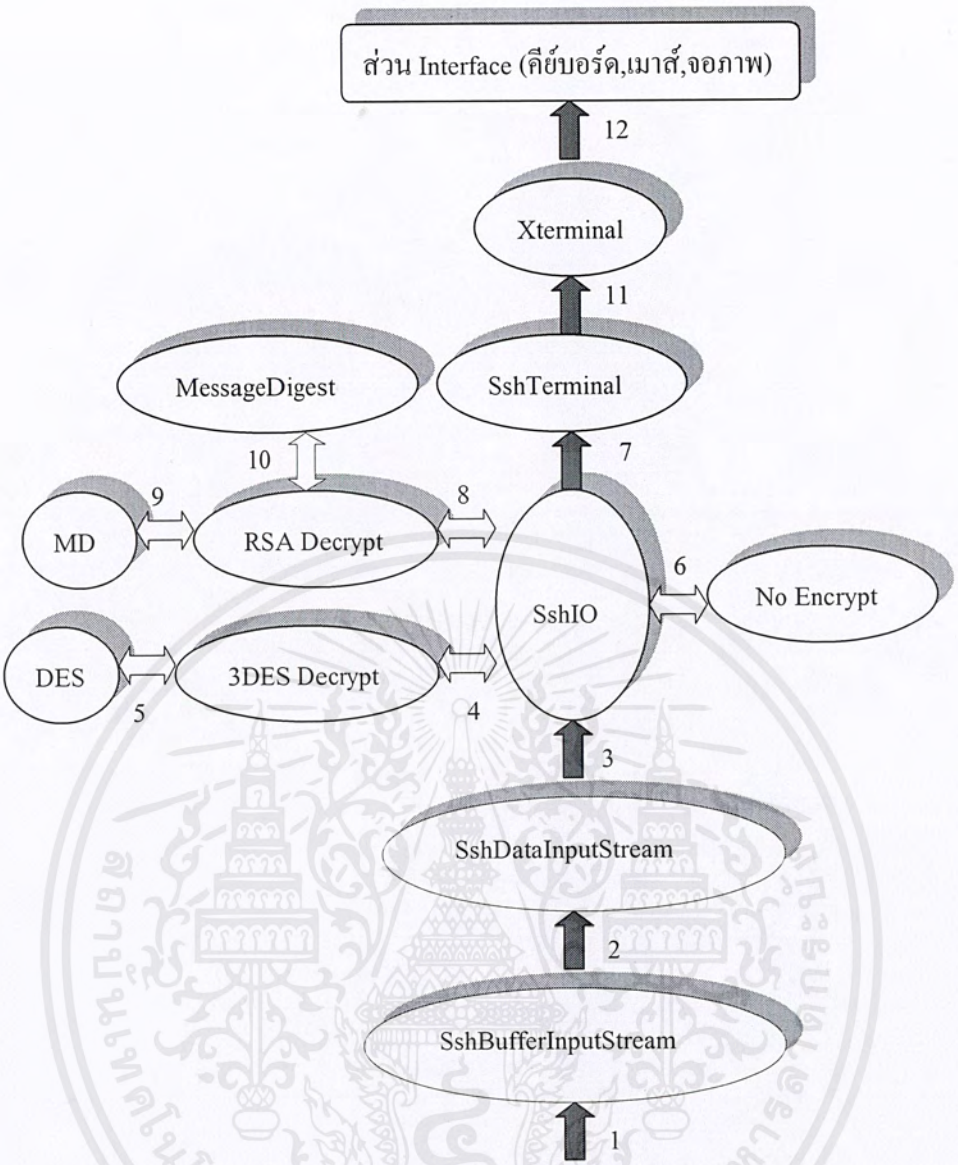
ในรูปที่ 6.8 แสดงไดอะแกรมการส่งข้อมูลไปยังเซิร์ฟเวอร์โดยแต่ละหมายเลขที่ลำดับการทำงานๆไว้ดังต่อไปนี้

1. มีอินพุตเข้ามาจากส่วนอินเทอร์เฟซเช่น มีการกดคีย์บอร์ดหรือเมาส์
2. อินพุตที่ได้จากการกดคีย์บอร์ดหรือเมาส์จะทำให้เกิดอีเวนท์ขึ้น และจะถูกดักจับจากฟังก์ชันต่าง ๆ ในออบเจ็กต์ Xterminal เช่น ฟังก์ชัน Keydown จะทำการดักจับการกดคีย์บอร์ดเป็นต้น
3. จากขั้นตอนที่ 2 จะมีการส่งผ่านค่าที่เกิดจากกิจกรรมต่าง ๆ เข้ามายังออบเจ็กต์ SshTerminal เพื่อนำมาแปลงเป็นรหัสมาตรฐานของ VT100
4. ในกรณีที่ยังไม่มีการเข้ารหัสข้อมูลจะส่งผ่านไปยังขั้นตอนต่อไป
5. เป็นขั้นตอนของการคำนวณค่าของ Session Key และทำการเข้ารหัสคีย์ด้วย RSA เพื่อส่งไปยังผู้ให้บริการ
6. ขั้นตอนการทำงานของ MessageDigest ที่ให้การสนับสนุนการเข้ารหัสด้วย RSA
7. ขั้นตอนของการคำนวณแบบ MD5 โดยข้อมูลที่ได้จากการคำนวณจะมีขนาด 128 บิต (16 ไบต์)
8. เป็นข้อมูลที่ได้จากการเข้ารหัสแล้ว(ในกรณีที่มีการเข้ารหัส) และถ้าไม่มีการเข้ารหัสก็จะได้ข้อมูลที่เป็นชนิด VT100



รูปที่ 6.8 แสดงออบเจ็กต์และการทำงานเมื่อจะส่งข้อมูลไปยังเซิร์ฟเวอร์

9. ใช้ในกรณีที่ผ่านขั้นตอนของการแลกเปลี่ยนคีย์เรียบร้อยแล้ว จำเป็นจะต้องมีการส่งข้อมูลที่มีการเข้ารหัสอีกแบบเพื่อต้องการเพิ่มความเร็วของการส่งข้อมูลในที่นี้ใช้การเข้ารหัสชนิด 3DES
10. ในการใช้งานของ 3DES นั้นจำเป็นจะต้องเปลี่ยนคีย์ที่แตกต่างกัน จำนวนครั้งของการเข้ารหัสขึ้นอยู่กับผู้ใช้ว่าต้องการเข้าอย่างไร
11. ข้อมูลที่ได้ ทั้งจากการเข้ารหัสหรือไม่เข้ารหัส จะเขียนลงในบัฟเฟอร์ที่เตรียมไว้ โดยออบเจ็กต์ SshDataOutPutStream นี้จะมีฟังกต์ชันที่คอยให้การสนับสนุนการเขียนข้อมูลต่างชนิดกันลงในบัฟเฟอร์
12. ขั้นตอนสุดท้ายจะทำการสร้างแพ็กเก็ตชนิด Binary packet protocol และจัดเรียงข้อมูลตามรูปแบบมาตรฐานของ Secure Shell และส่งออกไปยังผู้ให้บริการ



รูปที่ 6.9 แสดงออบเจ็กต์และการทำงานเมื่อรับข้อมูลจากเซิร์ฟเวอร์

จากไดอะแกรมการรับข้อมูลจากเซิร์ฟเวอร์ข้างต้นแต่ละหมายเลขที่กำกับไว้จะมีความหมายดังต่อไปนี้

1. มีแพ็กเก็ตผ่านเข้ามาทางซ็อกเก็ต และทำการอ่านจากซ็อกเก็นเข้ามาเก็บไว้ในบัฟเฟอร์
2. นำแพ็กเก็ตที่ได้รับมาทำการวิเคราะห์โดยการจำแนกชนิดออกเป็นส่วน ๆ ตามมาตรฐานของ Secure Shell เช่น ความยาว ,padding เป็นต้น
3. นำข้อมูลที่มีการแยกชนิดแล้วมา และทำการเปลี่ยนข้อมูลให้เหมาะสมเพื่อใช้สำหรับการคำนวณและถอดรหัส เช่น เปลี่ยนข้อมูลจากไบต์อะเรย์ให้กลายเป็น ข้อมูลชนิด BigInteger สำหรับการคำนวณชนิด RSA
4. ทำหน้าที่ถอดรหัสข้อมูลที่เข้ารหัสด้วย 3DES

5. เป็นการถอดรหัสด้วย DES ตามจำนวนครั้งของคีย์ที่เข้ามา
6. ในกรณีที่ไม่มีการเข้ารหัสก็ไม่มีความจำเป็นที่จะต้องมีการถอดรหัส
7. เป็นข้อมูลที่อยู่ในรูปของรหัส VT100 เมื่อได้มีการถอดรหัสแล้ว
8. เป็นการถอดรหัสด้วย RSA ใช้ในกรณีที่มีการแลกเปลี่ยนคีย์เท่านั้น และใช้เพียงครั้งเดียว
9. เป็นการคำนวณแบบ MD5 เพื่อให้ได้ข้อมูลออกมา 16 ไบต์
10. เป็นการคำนวณแบบ MessageDigest
11. ข้อมูลที่ได้จากการถอดรหัสแล้วจะเข้ามายังออบเจ็กต์ SshTerminal เพื่อนำมาวิเคราะห์และทำการจำแนกรหัสVT100ที่ใช้สำหรับควบคุมในการแสดงผลบนเทอร์มินอล
12. เมื่อทำการจำแนกคำสั่งทั้งหมดแล้วก็จะนำคำสั่งเหล่านั้นส่งไปแสดงผลยังจอภาพต่อไป

บทที่ 7

การทดลองและผลการทดลอง

หลังจากที่ได้ออกแบบและพัฒนาโปรแกรมแล้ว ต้องมีการทดสอบโปรแกรมที่พัฒนาขึ้นมา โดยทดลองติดตั้งโปรแกรมกับเครื่องที่มีระบบปฏิบัติการต่างๆ แล้วล็อกอินเข้าใช้งานเซิร์ฟเวอร์ที่เปิดบริการ Secure Shell

7.1 จุดประสงค์ของการทดลอง

- เพื่อทดสอบความเข้ากันได้ของโปรแกรมที่พัฒนาขึ้นกับโพรโตคอลมาตรฐานของ Secure Shell ให้สามารถใช้งานได้จริงและการทำงานสื่อสารระหว่างทั้งสองฝั่งไม่ผิดพลาด
- เพื่อทดสอบความเข้ากันได้กับระบบปฏิบัติการต่างๆ ที่ใช้งานจาวาแอพพลิเคชั่นกับโปรแกรมที่พัฒนาขึ้นให้ทำงานร่วมกันได้
- เพื่อทดสอบความสามารถในการใช้งานภาษาไทยบนเทอร์มินอลที่ติดต่อ โดยไม่ขึ้นกับระบบและสภาพแวดล้อมในการทำงานของเครื่องคอมพิวเตอร์ที่ทำงาน
- เพื่อใช้เป็นแนวทางในการนำเอาระบบความปลอดภัยที่ใช้ในโพรโตคอล ไปประยุกต์ใช้กับระบบการรักษาความปลอดภัยทางข้อมูลอื่นๆ

7.2 การเตรียมอุปกรณ์และสภาวะการที่ทำงานเพื่อทดลองโปรแกรม

อุปกรณ์ที่ใช้ในการทดสอบโปรแกรมและอุปกรณ์ที่ต้องใช้ในระบบมีดังต่อไปนี้

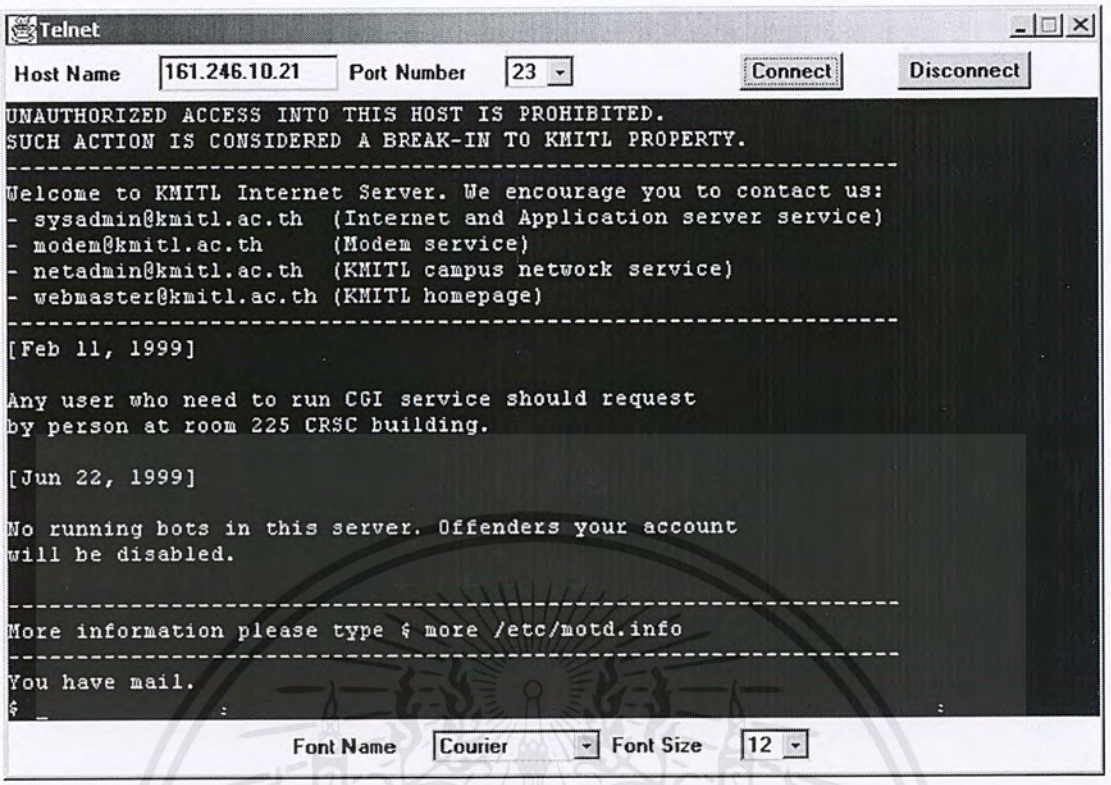
- เครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรม Java Develop Kit (JDK) เวอร์ชัน 1.1.8 ขึ้นไป ไม่ขึ้นกับระบบปฏิบัติการ
- เครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรม Java Virtual Machine
- เครื่องคอมพิวเตอร์ที่ให้บริการเป็นเซิร์ฟเวอร์และติดตั้ง Secure Shell Server ที่ให้การสนับสนุนเวอร์ชัน 1.5 หรือต่ำกว่า
- ระบบฮาร์ดแวร์ที่ใช้ซีพียูความเร็ว 133MHz ขึ้นไป

7.3 การทดลองโปรแกรมและผลการทดลอง

การทดลองโปรแกรมจะแบ่งออกเป็นช่วงๆ เนื่องจากการพัฒนาโปรแกรมมีขั้นตอนที่มาก เพื่อไม่ให้เกิดความสับสนและให้เกิดความสะดวกในการพัฒนา ซึ่งแบ่งขั้นตอนการทดลองและพัฒนาได้ดังนี้

7.3.1 การสร้างเทอร์มินอลจำลองการทำงาน

การพัฒนาโปรแกรมในส่วนของเทอร์มินอลจำลองการทำงาน เป็นส่วนที่สำคัญมากส่วนหนึ่งและต้องพัฒนาเป็นส่วนแรก เนื่องจากเทอร์มินอลจำลองนั้น จะใช้แสดงผลข้อมูลที่ติดต่อกับเซิร์ฟเวอร์หรือหน้าจอในการทำงานนั่นเอง โดยได้พัฒนาให้มีความเข้ากันได้กับมาตรฐาน VT100 ทดสอบโดยการสร้างเทอร์มินอลจำลองเทลเน็ต โดยที่ยังไม่มีส่วนของการเข้ารหัส และติดต่อกับเซิร์ฟเวอร์ที่พอร์ต 23 ก่อน ผลการทดลองได้ดังรูปที่ 7.1



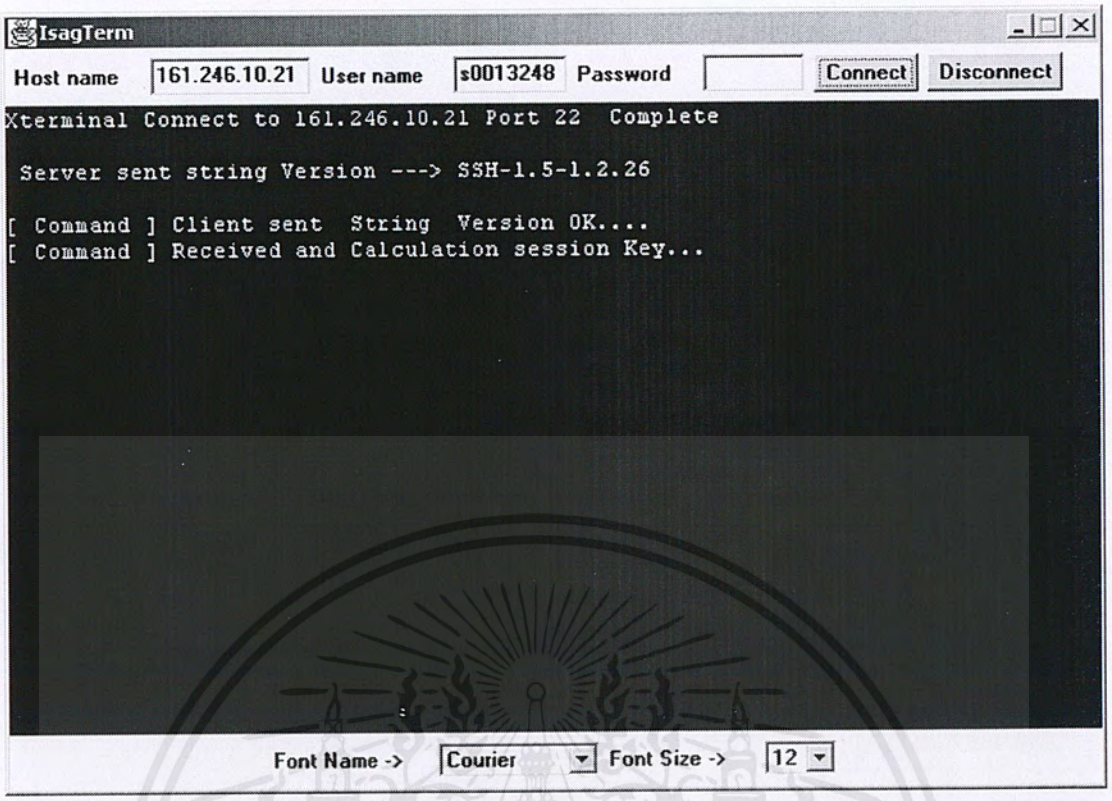
รูปที่ 7.1 แสดงหน้าจอของเทอร์มินอลจำลองชนิดไม่มีการเข้ารหัส

7.3.2 การสร้างการติดต่อกับเซิร์ฟเวอร์

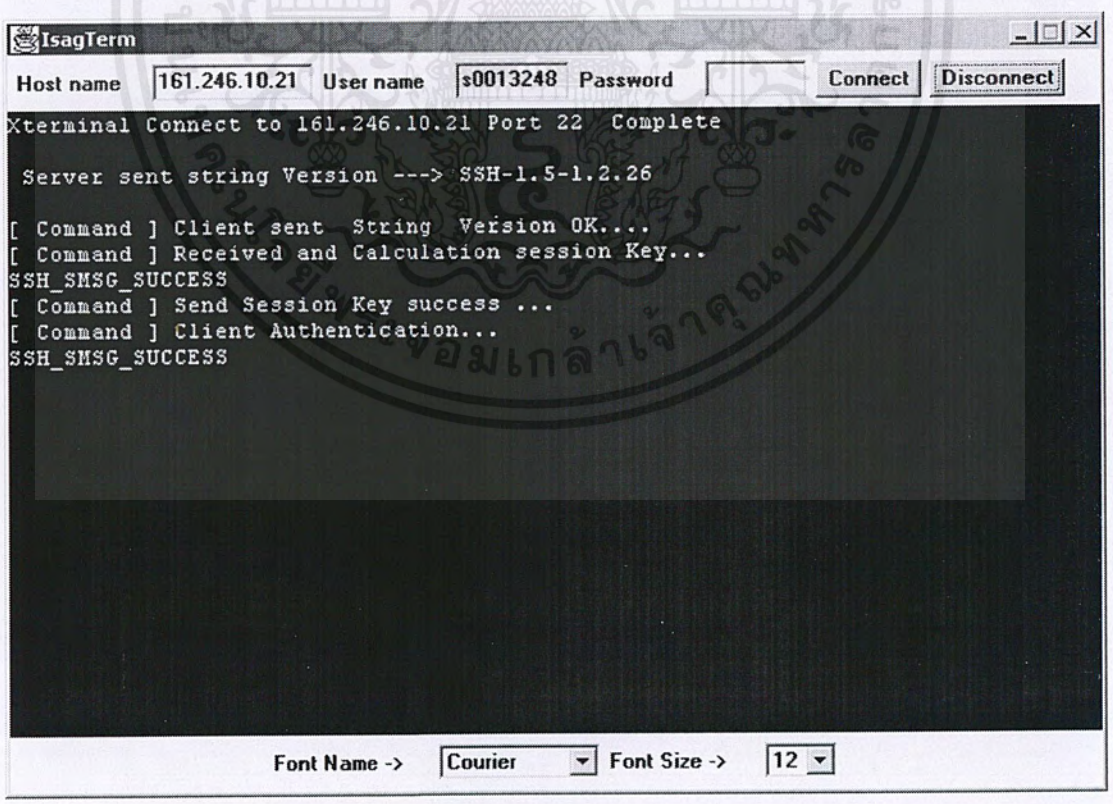
เมื่อเราทดสอบเทอร์มินอลจำลองเรียบร้อยแล้ว ขั้นตอนต่อมาก็คือการสร้างการติดต่อกับเซิร์ฟเวอร์แบบปลอดภัยตามขั้นตอนของ Secure Shell โดยจะแบ่งเป็นขั้นตอนย่อยดังต่อไปนี้

- **ช่วงการตรวจสอบเวอร์ชัน** จะเริ่มเมื่อมีการติดต่อกับเซิร์ฟเวอร์ โดยการกดปุ่ม Connect ซึ่งเป็นการเริ่มการติดต่อโดยจะส่งข้อความเวอร์ชันของฝั่งไคลเอนต์และเซิร์ฟเวอร์ให้แก่กัน เพื่อให้เกิดความเข้ากันได้ในการทำงานดังรูปที่ 7.2
- **ช่วงการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัส** เมื่อตรวจสอบเวอร์ชันของโพรโตคอลที่ใช้เรียบร้อยแล้วก็จะเข้าสู่ขั้นตอนการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัสซึ่งกันและกัน โดยจะมีพับบลิคคีย์เซิร์ฟเวอร์คีย์และข้อมูลอื่นๆ ที่จำเป็นในการเข้ารหัส การทำงานดังรูปที่ 7.3 โดยในแอพพลิเคชันนี้จะใช้การแลกเปลี่ยนคีย์ด้วยวิธีการ RSA
- **ช่วงการตรวจสอบผู้ใช้และพิสูจน์สิทธิ์** หลังจากตกลงวิธีการเข้ารหัสคีย์และขั้นตอนการติดต่อเรียบร้อย จะนำข้อมูลผู้ใช้และรหัสผ่านส่งไปยังเซิร์ฟเวอร์เพื่อตรวจสอบ ซึ่งขั้นตอนนี้จะมีการเข้ารหัสแล้วตามวิธีการที่ได้ตกลงกันในขั้นตอนที่แล้ว และการตรวจสอบสิทธิ์ของผู้ใช้จะตรวจสอบจากไฟล์พาสเวิร์ดที่เซิร์ฟเวอร์ การทำงานดังรูปที่ 7.4

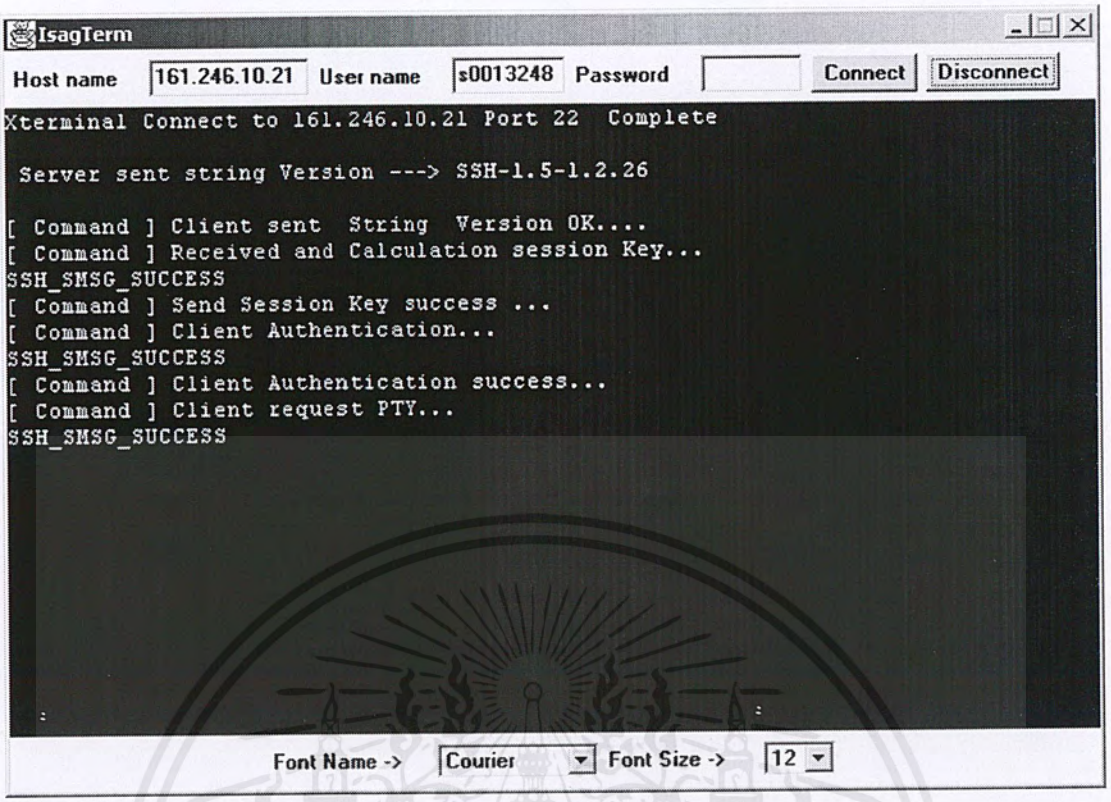
เมื่อใช้โปรแกรม NetXray ดักจับข้อมูลที่ได้เทียบกับเทอร์มินอลปกติที่ไม่มีการเข้ารหัส (เทลเน็ต) จะได้ดังรูปที่ 7.5 จะพบว่าไม่ข้อมูลที่อ่านจากเทอร์มินอล Secure Shell นั้นไม่สามารถอ่านได้เหมือนกับของเทอร์มินอลปกติ (เทลเน็ต)



รูปที่ 7.2 แสดงการติดต่อกับเซิร์ฟเวอร์ในขั้นตอนการตรวจสอบเวอร์ชัน



รูปที่ 7.3 แสดงการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัสระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์



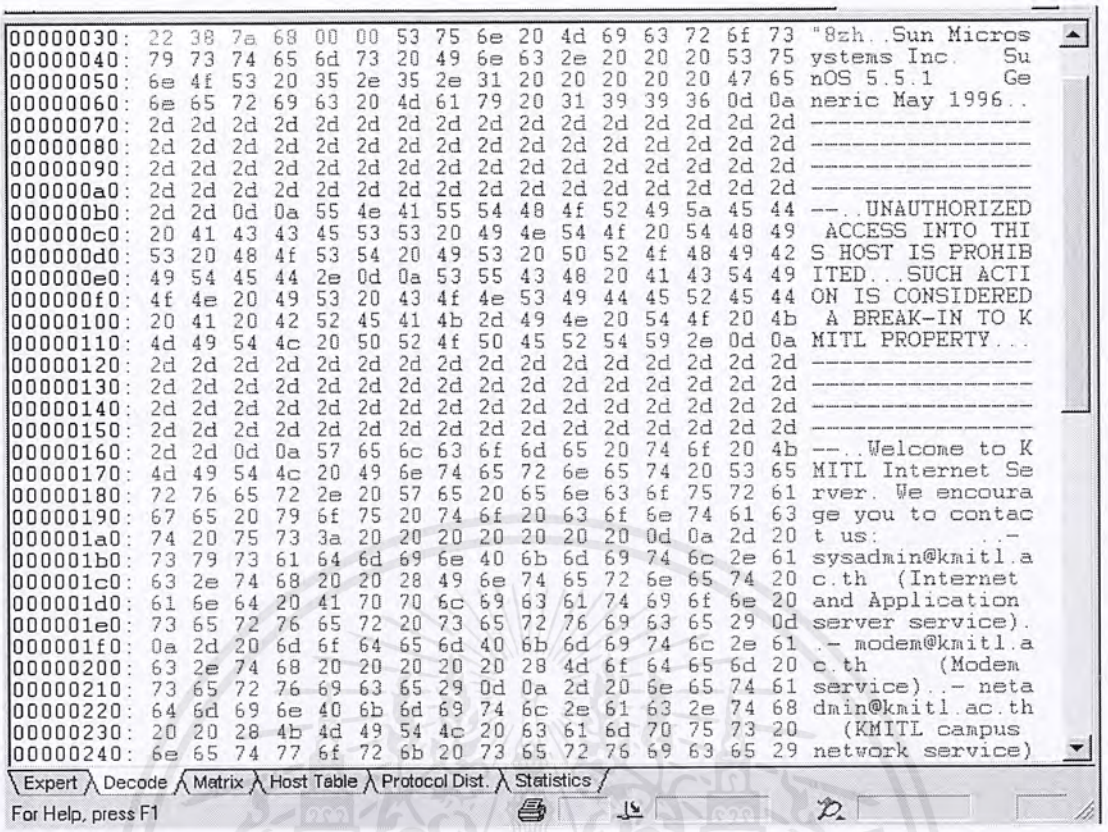
รูปที่ 7.4 แสดงการตรวจสอบและพิสูจน์สิทธิ์ผู้ใช้

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^ãÛh\¬▏ >§

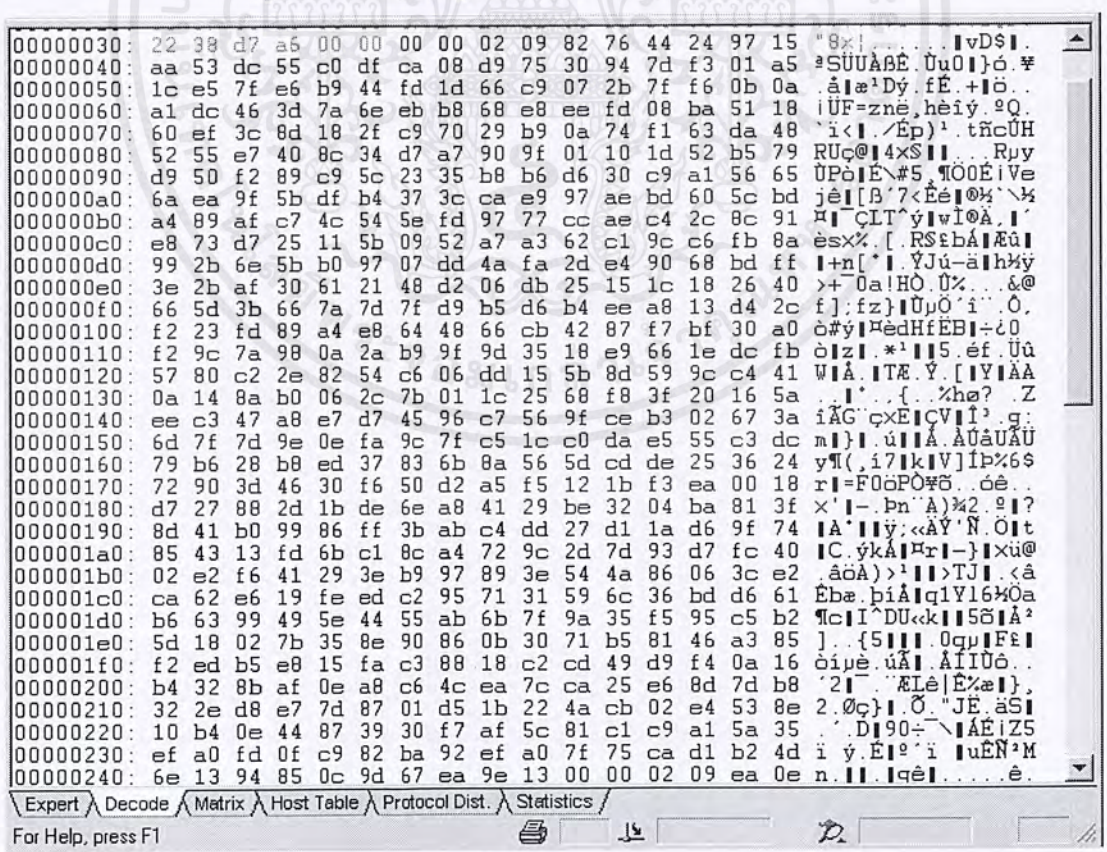
รูปที่ 7.5 แสดงการเปรียบเทียบข้อมูลที่ดักจับได้บน NetXray ระหว่างเทลเน็ตกับ Secure Shell

7.3.3 ช่วงการแลกเปลี่ยนข้อมูลแบบอินเตอร์แอ็กทีฟ

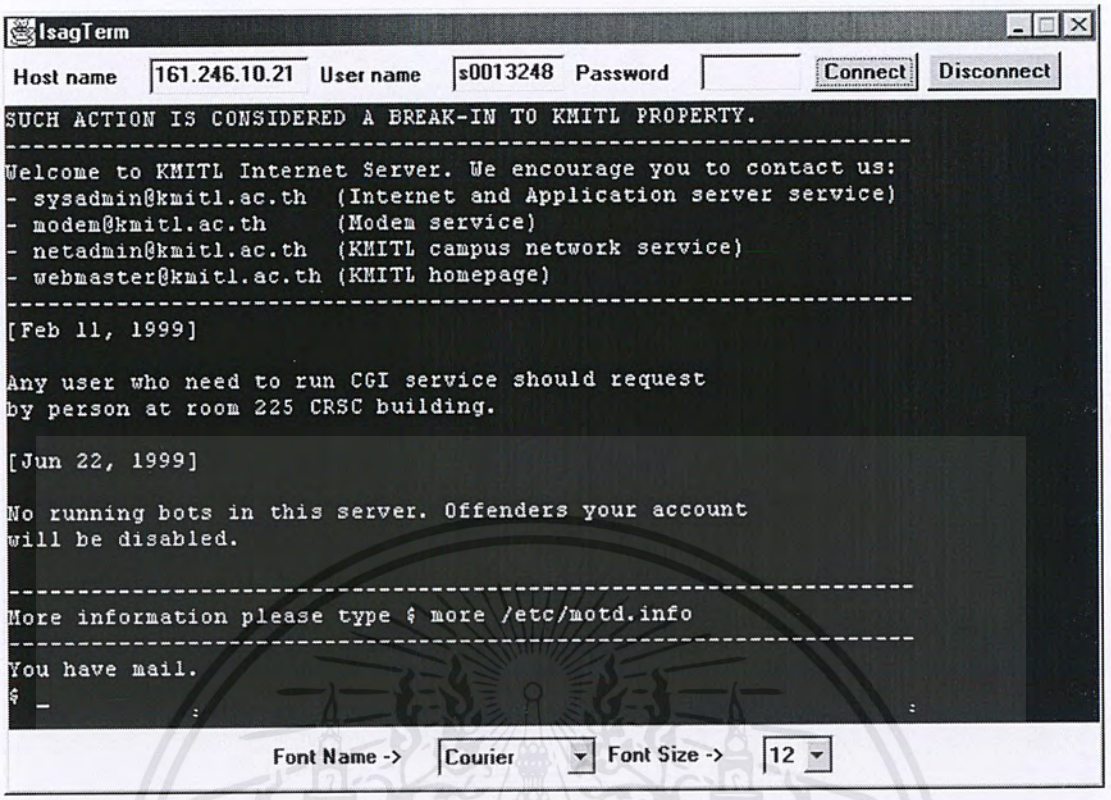
เมื่อติดต่อแลกเปลี่ยนข้อมูลและตรวจสอบสิทธิ์ผู้ใช้เรียบร้อยแล้ว จะเข้าสู่ขั้นตอนการติดต่อสื่อสารข้อมูลกันแบบอินเตอร์แอ็กทีฟ เมื่อมีการกดคีย์ใดๆ ไคลเอ็นต์จะส่งข้อมูลไปยังเซิร์ฟเวอร์แปลความหมายแล้วส่งข้อมูลดังกล่าวกลับมา การสื่อสารข้อมูลระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้นจะต้องเข้ารหัสเพื่อไม่ให้ดักจับข้อมูลไปอ่านทำความเข้าใจได้ทันทีดังเช่นโปรแกรมเทลเน็ต การเข้ารหัสและถอดรหัสในส่วนของการแลกเปลี่ยนข้อมูลแบบอินเตอร์แอ็กทีฟนี้จะใช้การเข้าและถอดรหัสชนิด 3DES ที่มีความน่าเชื่อถือได้ และมีความเร็วพอสมควรในการเข้ารหัสข้อมูล



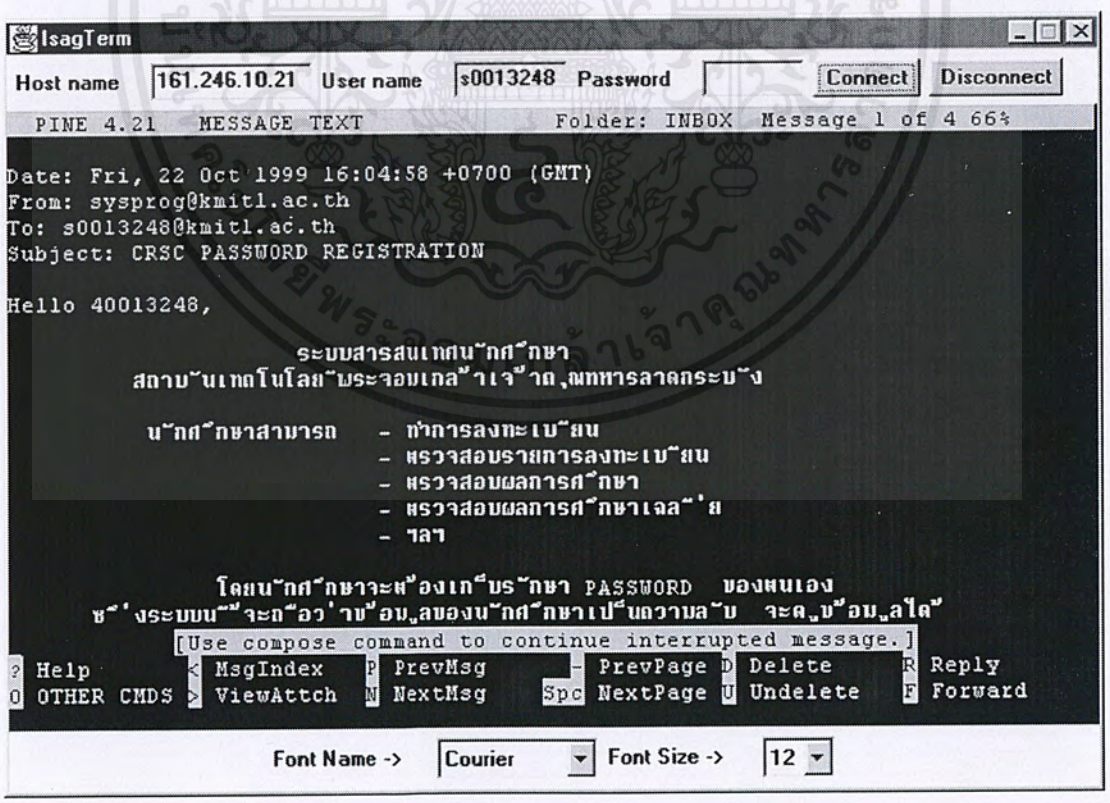
รูปที่ 7.6 แสดงข้อมูลที่ดักจับได้จากเทอร์มินอลเทลเน็ต



รูปที่ 7.7 แสดงข้อมูลที่ดักจับได้จากเทอร์มินอล Secure Shell



รูปที่ 7.8 แสดงหน้าจอและการทำงานของโปรแกรมเทอร์มินอล Secure Shell ด้วยจาวา



รูปที่ 7.9 แสดงการทำงานของโปรแกรมเทอร์มินอล Secure Shell ด้วยจาวากับภาษาไทย

7.3.4 การสร้างการแสดงผลและการทำงานร่วมกับภาษาไทย

ส่วนสุดท้ายที่พัฒนาและทดสอบคือส่วนของการแสดงผลและการทำงานร่วมกับภาษาไทย เนื่องจากเทอร์มินอลจำลองจะทำงานกับระบบยูนิกซ์เซิร์ฟเวอร์ที่ไม่รองรับการทำงานภาษาไทย และจาวาเองไม่รองรับการแสดงผลภาษาไทยโดยตรงเนื่องจากรหัสที่ใช้เป็น Uniceode ไม่ใช่ ASCII แต่มีไฟล์ที่กำหนดค่าให้ทำงานได้กับภาษาไทยชื่อ font.properties.th ที่จะแมปตัวอักษรของจาวาให้ใช้กับตัวอักษรของระบบปฏิบัติการที่มีภาษาไทย ไฟล์ font.properties นี้จะมีนามสกุลตามแต่ภาษาของระบบปฏิบัติการนั้นๆ

เพื่อให้การแสดงผลภาษาไทยไม่ขึ้นกับระบบปฏิบัติการ จึงต้องสร้างชุดตัวอักษรภาษาไทยเฉพาะ ส่วนการพิมพ์หรือรับข้อมูลภาษาไทยนั้นยังต้องขึ้นกับระบบปฏิบัติการ เนื่องจากการแมปตัวอักษรจากแป้นพิมพ์ยังเป็นหน้าที่ของระบบปฏิบัติการ ในส่วนของการแสดงผลภาษาไทยบนเทอร์มินอล Secure Shell แสดงดังรูปที่ 7.9

บทที่ 8
บทวิจารณ์และสรุป

ในการสร้างและพัฒนาโปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวานั้น เป็นการพัฒนาโปรแกรมให้ทำงานได้ตามโพรโตคอล Secure Shell ที่มีมาตรฐานความปลอดภัยที่สูงในการติดต่อระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ ทำให้ผู้เข้าใช้งานเกิดความมั่นใจในการใช้งานยูนิกซ์เซิร์ฟเวอร์มากยิ่งขึ้น และตลอดเวลาการพัฒนาโปรแกรมจำลองเทอร์มินอลแบบปลอดภัยนี้ ได้พัฒนาให้ตรงไปตามรูปแบบมาตรฐานของการทำงานในแต่ละส่วนที่ศึกษามาดังบทสรุปและวิจารณ์ต่อไปนี้

8.1 บทสรุปและวิเคราะห์มาตรฐานของ Secure Shell

การติดต่อสื่อสารผ่านระบบเครือข่ายในปัจจุบัน ต้องการความปลอดภัยในการส่งข้อมูลเป็นอย่างมาก นอกจากนั้นยังต้องการความรวดเร็วในการติดต่อสื่อสารอีกด้วย ซึ่งระบบที่เป็นยูนิกซ์เซิร์ฟเวอร์นั้น การรีโมตเข้าใช้งานระยะไกลนั้น โพรโตคอล Secure Shell มีความเหมาะสมมากเนื่องจากการเข้ารหัสที่หลากหลายตามความต้องการของไคลเอ็นต์ที่ต้องการเข้าใช้บริการ ความปลอดภัยในการส่งข้อมูลที่สูงกว่าโพรโตคอลของเทลเน็ตและ rlogin ที่ไม่มีการเข้ารหัสข้อมูลที่ส่งทำให้ไม่มีความปลอดภัยในการติดต่อสื่อสาร โดยข้อดีของโพรโตคอล Secure Shell มีดังนี้

- การปลอมแปลง IP ไม่สามารถกระทำได้ เนื่องจากมีการตรวจสอบการเข้าใช้งานของผู้ใช้มีความปลอดภัยสูง โดยการเข้ารหัส Host key และ cookie ที่สุ่มขึ้นป้องกันการปลอมแปลง IP
- ข้อมูลผู้ใช้และรหัสผ่านของผู้ใช้นั้นจะมีความปลอดภัยในการส่งที่สูงเนื่องจากมีการเข้ารหัสที่มีประสิทธิภาพสูงด้วยกรรมวิธีของ RSA ทำให้ยากต่อการถอดรหัส
- ข้อมูลที่ติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้นมีการเข้ารหัสที่หลากหลายตามความต้องการของผู้ใช้ฝั่งไคลเอ็นต์ ทำให้ยากต่อการถอดรหัสข้อมูลที่ดักจับมาได้เป็นอย่างมาก
- Man in the middle attack ไม่สามารถทำได้เนื่องจากมีการใช้ Host key และ Session key

การพัฒนาโปรแกรมจึงเป็นไปตามมาตรฐานการเข้ารหัสข้อมูลของโพรโตคอล Secure Shell ที่มีการเข้ารหัสที่แข็งแกร่ง ซึ่งได้พัฒนาในส่วนต่างๆดังต่อไปนี้

- ส่วนของการแลกเปลี่ยนคีย์ Host key และ Session key ระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะใช้การเข้ารหัสแบบ RSA
- การตรวจสอบสิทธิ์ผู้ใช้และการส่ง User และ Password จะใช้การเข้ารหัสแบบ RSA
- การแลกเปลี่ยนข้อมูลระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะใช้การเข้ารหัสชนิด DES และ 3DES เท่านั้น

การเข้ารหัสที่หลากหลายรูปแบบด้วยอัลกอริทึมที่มีความเชื่อถือในปัจจุบัน ทำให้การถอดรหัสด้วยการสุ่มคีย์ทำได้ยาก ซึ่งทำให้ข้อมูลมีความปลอดภัยสูงด้วยเช่นกัน

8.2 บทสรุปและวิเคราะห์ของโปรแกรมที่พัฒนาขึ้น

สำหรับโปรแกรมที่ได้พัฒนาขึ้น เป็นส่วนของไคลเอ็นต์ที่ติดต่อไปยังเซิร์ฟเวอร์ที่ให้บริการ Secure Shell เวอร์ชันไม่เกิน 2.0 โดยยังมีข้อจำกัดหลายประการในการพัฒนาได้คือ

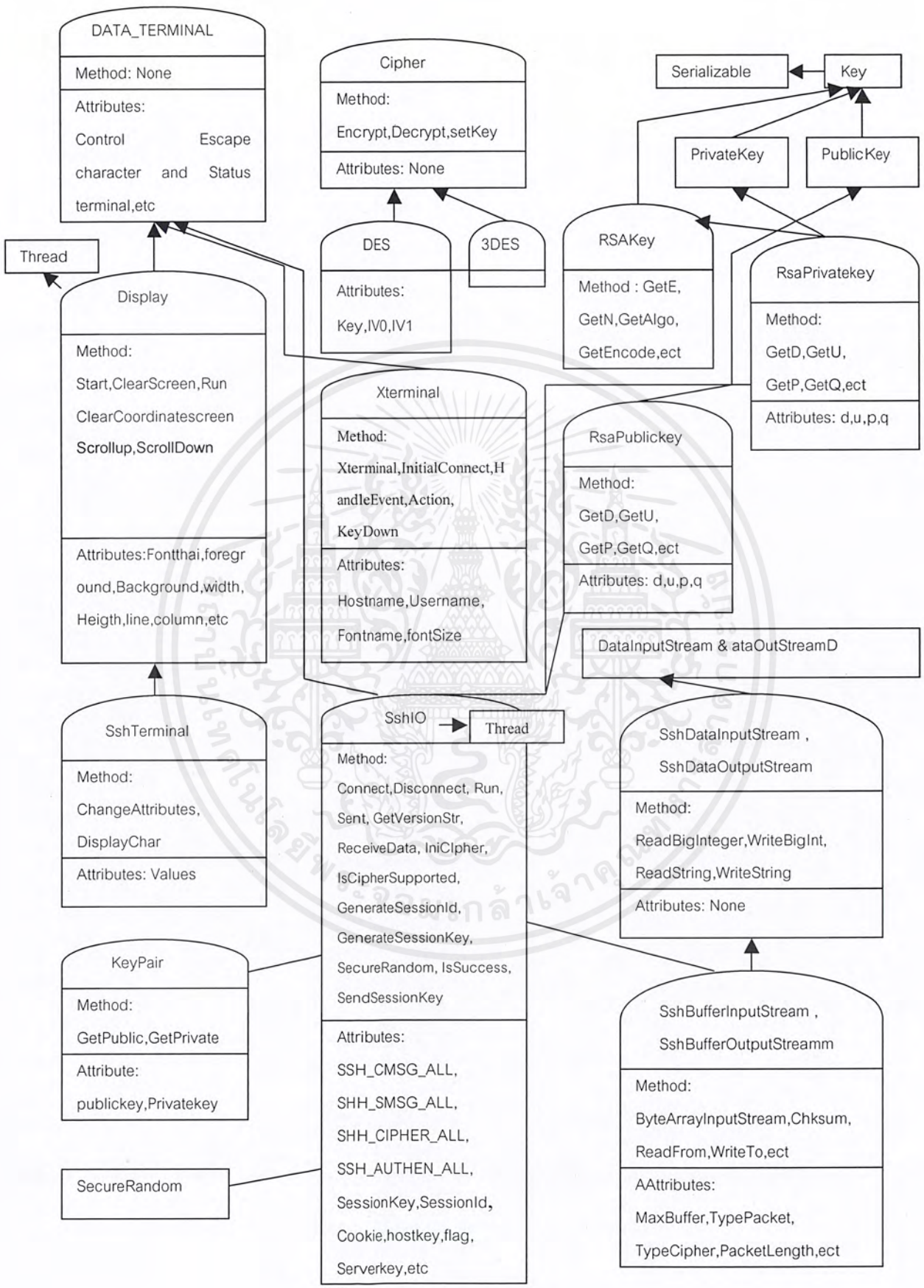
- การรองรับการเข้ารหัสในโปรแกรมยังไม่ครอบคลุมอัลกอริทึมการเข้ารหัสทั้งหมดของเซิร์ฟเวอร์ Secure Shell ที่มีให้บริการ
- อัลกอริทึมที่ใช้ในการเข้ารหัสคีย์มีความซับซ้อนมากทำให้เสียเวลาในการทำงานค่อนข้างนาน และตัวจาวาเองก็มีการทำงานค่อนข้างช้า ทำให้มีโปรแกรมทำการติดต่อสื่อสารได้ช้า
- การแสดงผลภาษาไทยสามารถทำได้ดี แต่ไม่มีการชดเชยสระที่แสดงเนื่องจากการชดเชยสระทำให้ยากต่อการควบคุมการแสดงผลที่หน้าจอเทอร์มินอล และตัวอักษรที่แสดงจะไม่ขยายขนาดตามการเปลี่ยนขนาดตัวอักษร เพราะว่าตัวอักษรที่แสดงนั้นเป็นชนิดรูปภาพ
- การใช้งานแป้นพิมพ์ภาษาไทยบนระบบปฏิบัติการอื่นๆยังมีปัญหา เนื่องจากการแมปแป้นพิมพ์ในแต่ละระบบปฏิบัติการจะขึ้นอยู่กับไฟล์ Properties ของจาวาเวอร์ชวลแมชชีน ทำให้บางครั้งเทอร์มินอลมีอาการผิดปกติได้

8.3 แนวทางการพัฒนาต่อในอนาคต

แนวทางในการพัฒนาโปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวาภาษาไทยนั้น มีแนวทางในการพัฒนาและประยุกต์ดังต่อไปนี้

- พัฒนาการรองรับมาตรฐาน Secure Shell ให้ครบถ้วน โดยเฉพาะเรื่องของอัลกอริทึมการเข้ารหัสขณะที่ติดต่อสื่อสารข้อมูลที่ยังไม่ครบถ้วน เพื่อให้ยากแก่การดักจับข้อมูลไปถอดรหัส
- การพัฒนาให้มีรูปแบบการใช้งานเครือข่ายหลายแบบ ยกตัวอย่างเช่นให้รองรับกับโพรโตคอลของ FTP เพื่อใช้รับและส่งไฟล์
- ปรับปรุงรูปแบบการเข้ารหัส เนื่องจากการสร้างคีย์และรับส่งคีย์ใช้เวลาค่อนข้างนานทำให้ไม่เหมาะแก่การใช้งานกับเครื่องรุ่นเก่า
- พัฒนาระบบการรับข้อความภาษาไทยจากแป้นพิมพ์ ให้สามารถใช้ร่วมกันได้ทุกระบบปฏิบัติการ
- ปรับปรุงการแสดงผลภาษาไทยให้มีความสามารถในการชดเชยสระ ปรับขนาดของตัวอักษรและเปลี่ยนรูปแบบของอักษรภาษาไทยได้
- ปรับปรุงการทำงานให้รองรับโพรโตคอล Secure Shell เวอร์ชัน 2 ด้วย

ภาคผนวก ก
คลาสไดอะแกรม



ภาคผนวก ข

รายละเอียดของแพ็กเก็ต Secure Shell

SSH Packet Description

ในการติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ในแต่ละแพ็กเก็ตที่ติดต่อกันจะมีชนิดที่แตกต่างกัน ขึ้นอยู่กับข้อมูลที่จะส่ง ซึ่งแพ็กเก็ตแต่ละชนิดจะมีเบอร์เฉพาะ เพื่อให้ทั้งเซิร์ฟเวอร์และไคลเอนต์เข้าใจตรงกัน

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

0 SSH_MSG_NONE

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

1 SSH_MSG_DISCONNECT

string Cause of disconnection

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

2 SSH_SMSG_PUBLIC_KEY

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

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

3 SSH_CMSG_SESSION_KEY

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

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

4 SSH_CMSG_USER

string	user login name on server

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

5 SSH_CMSG_AUTH_RHOSTS

string	client-side user name

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

6 SSH_CMSG_AUTH_RSA

mp-int	identity_public_modulus

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

7 SSH_SMSG_AUTH_RSA_CHALLENGE

mp-int encrypted challenge

8 SSH_CMSG_AUTH_RSA_RESPONSE

16 bytes MD5 of decrypted challenge

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

9 SSH_CMSG_AUTH_PASSWORD

string plain text password

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

10 SSH_CMSG_REQUEST_PTY

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

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

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

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

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

n bytes tty modes encoded in binary

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

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

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

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

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

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

11 SSH_CMSG_WINDOW_SIZE

32-bit int terminal height, rows

32-bit int terminal width, columns

32-bit int terminal width, pixels

32-bit int terminal height, pixels

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

12 SSH_CMSG_EXEC_SHELL

(no arguments)

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

13 SSH_CMSG_EXEC_CMD

string command to execute

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

14 SSH_SMSG_SUCCESS

(no arguments)

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

15 SSH_SMSG_FAILURE

(no arguments)

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

16 SSH_CMSG_STDIN_DATA

string data

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

17 SSH_SMSG_STDOUT_DATA

string data

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

18 SSH_SMSG_STDERR_DATA

string data

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

19 SSH_CMSG_EOF

(no arguments)

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

20 SSH_SMSG_EXITSTATUS

32-bit int exit status of the command

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

21 SSH_MSG_CHANNEL_OPEN_CONFIRMATION

32-bit int remote_channel

32-bit int local_channel

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

22 SSH_MSG_CHANNEL_OPEN_FAILURE

32-bit int remote_channel

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

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

23 SSH_MSG_CHANNEL_DATA

32-bit int remote_channel

string data

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

24 SSH_MSG_CHANNEL_CLOSE

32-bit int remote_channel

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

25 SSH_MSG_CHANNEL_CLOSE_CONFIRMATION

32-bit int remote_channel

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

26 (OBSOLETED; was unix-domain X11 forwarding)

27 SSH_SMSG_X11_OPEN

32-bit int local_channel

string originator_string (see below)

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

28 SSH_CMSG_PORT_FORWARD_REQUEST

32-bit int server_port

string host_to_connect

32-bit int port_to_connect

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

---

29 SSH_MSG_PORT_OPEN

32-bit int local_channel

string host_name

32-bit int port

string originator_string (see below)

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

---

30 SSH_CMSG_AGENT_REQUEST_FORWARDING

(no arguments)

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

---

31 SSH_SMSG_AGENT_OPEN

32-bit int local_channel.

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

32 SSH_MSG_IGNORE

string data

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

33 SSH_CMSG_EXIT_CONFIRMATION

(no arguments)

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

34 SSH_CMSG_X11_REQUEST_FORWARDING

string x11_authentication_protocol

string x11_authentication_data

32-bit int screen number (if SSH_PROTOFLAG_SCREEN_NUMBER)

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

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

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

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

35 SSH_CMSG_AUTH_RHOSTS_RSA

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

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

36 SSH_MSG_DEBUG

string debugging message sent to the other side

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

37 SSH_CMSG_REQUEST_COMPRESSION

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

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

38 SSH_CMSG_MAX_PACKET_SIZE

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

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

39 SSH_CMSG_AUTH_TIS

(no arguments)

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

40 SSH_SMSG_AUTH_TIS_CHALLENGE

string tis challenge

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

41 SSH_CMSG_AUTH_TIS_RESPONSE

string user response to tis challenge

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

42 SSH_CMSG_AUTH_KERBEROS

string authentication info

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

43 SSH_SMSG_AUTH_KERBEROS_RESPONSE

string response data

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

44 SSH_CMSG_HAVE_KERBEROS_TGT

string kerberos credentials

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

ภาคผนวก ค

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

0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

Background Colors

0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

ภาคผนวก ง
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

หนังสืออ้างอิง

[1] Patrick Chan, Rosanna Lee (1996) : "*The Java Class Libraries An Annotated Reference*", An imprint of Addison Wesley Longman, Inc. 1996.

[2] Steve Potts (1999) : "*Java 1.2 How-To*", Techmedia 1999.

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

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

[5] Merlin and Conrad Hughes, Michael Shoffner, Maria Winslow :"*Java Network Programming*" , Manning Publications Co. 1997

[6] Elliotte Rustly Harold : "*Java Network Programming*", O'Reilly & Associates, Inc. United State of America 1997.

[7] กิตติ ภักดีวัฒนะกุล : "*Java ฉบับโปรแกรมเมอร์*", บริษัท เคทีพี คอมพ์ แอนด์ คอนซัลท์ จำกัด 1999.

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

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

เว็บไซต์อ้างอิง

[1] http://www.uni-karlsruhe.de/~ig25/ssh-faq/

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

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

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

[5] http://www.cl.cam.ac.uk/~fapp2/software/

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