ชุดโปรแกรมสื่อสารอย่างปลอดภัยด้วยโพรโตคอลเอสเอสเอช

Secure Communicatiom Client Suite Using SSH Protocol

โดย

นายเอกวุฒิ ว่องสุวรรณเลิศ รหัส 45015401

นายเอกสมัตถ์ พึ่งพลาวัฒน์ รหัส 45015402

อาจารย์ที่ปรึกษา

อ.อัครเดช วัชระภูพงษ์

อ.ธนา หงษ์สุวรรณ

อ.ธนัญชัย ตรีภาค

รายงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ชุดโปรแกรมสื่อสารอย่างปลอดภัยด้วยโพรโตคอลเอสเอสเอช

Secure Communicatiom Client Suite Using SSH Protocol

ผู้จัดทำ

1. นายเอกวุฒิ ว่องสุวรรณเลิศ รหัสประจำตัว 45015401
2. นายเอกสมัตถ์ พึ่งพลาวัฒน์ รหัสประจำตัว 45015402

อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชรภูพงษ์)

อาจารย์ที่ปรึกษา

(อาจารย์ธนา หงษ์สุวรรณ)

อาจารย์ที่ปรึกษา

(อาจารย์ธนัญชัย ตรีภาค)

โปรแกรมจำลองเทอร์มินอลสำหรับเข้าใช้ยูนิกซ์เซิร์ฟเวอร์แบบปลอดภัยด้วยจาวา

นายเอกวุฒิ ว่องสุวรรณเลิศ
นายเอกสมัตถ์ พึ่งพลาวัฒน์
อ.อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา
อ.ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อ.ธนัญชัย ตรีภาค อาจารย์ที่ปรึกษา
ปีการศึกษา 2547

บทคัดย่อ

ในปัจจุบันเมื่อผู้ใช้ต้องการเข้าใช้งานเครื่องคอมพิวเตอร์ใดๆในระบบยูนิกซ์ และในระบบเครือข่ายที่ต้องการความปลอดภัยในการส่งข้อมูล แอพพลิเคชันที่เราใช้งานทั่วๆไปเช่น telnet และ ftp ไม่มีกรรมวิธีทำให้ข้อมูลที่ใช้เหล่านั้นปลอดภัยเนื่องจากจะส่งข้อมูลในลักษณะที่ไม่มีการเข้ารหัส ดังนั้นโครงงานนี้ จึงเสนอวิธีการพัฒนาแอพพลิเคชันในรูปแบบเดียวกับ telnet และ ftp แต่จะมีกลยุทธ์ที่ทำให้ข้อมูลเหล่านั้นปลอดภัยมากที่สุดโดยใช้ โพรโตคอล SSH (Secure Shell) ที่รองรับการเข้าและถอดรหัสหลากหลายประเภท เช่น RSA, DES, 3DES, IDEA และ MD5 ทำให้มีเสถียรภาพในการใช้งานเป็นอย่างมาก โพรโตคอล Secure Shell ที่นำมาใช้คือเวอร์ชัน 2.0 แต่ในโครงงานเดิมคือ IsagTerm 2542 และ IsagFTP นั้นได้นำ โพรโตคอล Serure Shell เวอร์ชัน 1.0 มาใช้ซึ่งในปัจจุบันนี้ไม่นิยมนำมาใช้เพราะในโพรโตคอล Secure Shell เวอร์ชัน 2.0 นั้นมีความยืดหยุ่นกว่าในการทำงาน แต่ทั้งโครงงานเก่าและโครงงานที่พัฒนาขึ้นมาใหม่คือ IsagTerm 2547 และ IsagSFTP จะใช้ภาษาจาวาในการพัฒนา ซึ่งมีโครงสร้างเป็นออบเจ็กต์ – โอเรียนเต็ดทำให้พัฒนาแอพพลิเคชันได้ง่าย อีกทั้งแอพพลิเคชันที่พัฒนาด้วยภาษาจาวานั้นสามารถนำไปใช้กับระบบปฏิบัติการที่มีจาวาเวอร์ชวนแมชชีน โดยไม่ต้องคอมไพล์หรือเขียนโค้ดขึ้นมาใหม่

Secure Communicatiom Client Suite Using SSH Protocol

Mr.Ekkawut	Wongsuwanloet	
Mr. Eksamat	Phungphalawat	
Mr.Akkradach	Watcharapupong	Advisor
Mr.Thana	Hongsuwan	Advisor
Mr.Thananchai	Tripak	Advisor

ABSTRACT

Nowadays, when users want to use the applications in UNIX systems and such as Telnet or FTP 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 safty. The SSH protocol supports many kinds of encryption-decryption technology such as RSA, DES, 3DES, IDEA, and MD5. We use SSH version 2.0 and develop the client application by using JAVA 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 3 ท่านคือ อาจารย์อัครเดช วัชระภูพงษ์ อาจารย์ธนา หงษ์สุวรรณ และอาจารย์ ธนัญชัย ตรีภาคที่คอยแนะนำ เป็นที่ปรึกษา และเอาใจใส่กับการทำโครงงานนี้เป็นอย่างดี ซึ่งทางคณะผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษาทั้งสามท่านเป็นอย่างสูง

นอกเหนือจากนี้ก็ต้องขอขอบพระคุณคณะอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เป็นอย่างยิ่งที่ได้ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ อีกทั้งภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้เอื้อเฟื้ออุปกรณ์ทรัพยากร สถานที่และอำนวยความสะดวกต่างๆในการทำโครงงานนี้ด้วย

ขอขอบใจเพื่อนๆชาว ISAG ทุกคนที่ช่วยเหลือในการทำงานและแก้ไขอุปสรรคต่างๆในการทำงานให้ผ่านพ้นไปด้วยดี เป็นที่ปรึกษาและกำลังใจที่ดีเสมอมา

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

เอกวุฒิ ว่องสุวรรณเลิศ
เอกสมัตถ์ พึ่งพลาวัฒน์

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1

บทนำ

ความสำคัญและที่มา

ปัจจุบันเครือข่ายคอมพิวเตอร์ได้มีบทบาทสำคัญต่อการใช้งานในชีวิตประจำวัน เป็นอย่างมาก ดังนั้นประเด็นเรื่องความปลอดภัยเป็นสิ่งที่ต้องคำนึงถึง โปรแกรมและโพรโตคอลที่ใช้งานในปัจจุบันมักผ่านการออกแบบมาตั้งแต่ช่วงที่ระบบเครือข่ายไม่ได้ใช้งานแพร่หลายและมีความสำคัญเหมือนเช่นในปัจจุบัน ซึ่งในขณะออกแบบไม่ได้คำนึงถึงเรื่องความปลอดภัยมากนัก ซึ่งส่งผลให้การใช้งานที่ต้องพึงพาโปรแกรมเหล่านั้น มีความเสี่ยงในประเด็นเรื่องความปลอดภัยได้

telnet และ ftp เป็นโปรแกรมหนึ่งที่ใช้งานกันแพร่หลาย แต่มีปัญหาในเรื่องความปลอดภัย เพราะข้อมูลที่ส่งเป็นข้อความตัวอักษรธรรมดา การดักจับ หรือแก้ไขสามารถทำได้ง่าย หรือแม้แต่โปรแกรมพวกส่งข้อความด่วน (Instant Messaging) ก็มีปัญหาเช่นเดียวกัน

ซีเคียวเชล (Secure Shell : SSH) เป็นโพรโตคอลหนึ่งที่ออกแบบมาเพื่อใช้แทนที่หรือใช้ร่วมกับโปรแกรมที่มีปัญหาในด้านความปลอดภัย สนับสนุนการทำอินเทอร์แอกทีฟเซสชัน (interactive session) ทีซีพีพอร์ตฟอร์เวิร์ด (TCP port forwarding) และการแลกเปลี่ยนไฟล์อย่างปลอดภัย (SFTP) ซึ่งตอนนี้ได้พัฒนามาเป็นเวอร์ชัน 2.0

IsagTerm 2542 และ IsagFTP นั้นเป็นโครงงานเก่าซึ่งพัฒนาขึ้นมาโดยใช้โพรโตคอล SSH-1 ซึ่งตอนนี้ไม่ได้รับความนิยมแล้วเพราะในเวอร์ชัน 2.0 นั้นมีการทำงานที่ยืดหยุ่นกว่าทั้งในเรื่องการเรียกใช้อัลกอริทึมและการพัฒนาอัลกอริทึมขึ้นมาใช้งานเอง

หลักการและเหตุผล

เนื่องจากในปัจจุบันการสื่อสารผ่านระบบเครือข่ายไม่ว่าจะเป็นการควบคุมระยะไกล (Remote) และการส่งไฟล์ข้อมูล (File Transfer) สามารถทำได้อย่างรวดเร็ว และกว้างขวางมากขึ้น โดยการเชื่อมต่อผ่าน TELNET และ FTP แต่การสื่อสารข้อมูลในรูปแบบนี้ยังมีปัญหาในเรื่องความปลอดภัยของข้อมูลเพราะการส่งข้อมูลที่เป็นตัวอักษรธรรมดา ซึ่งหากมีการดักจับข้อมูล ก็จะสามารถทำให้ผู้ที่ได้รับข้อมูล อ่าน หรือแก้ไขข้อมูลนี้ได้ อาจมีทั้งข้อมูลที่เป็นชื่อผู้ใช้ หรือ รหัสผ่านได้ จึงจำเป็นที่จะต้องมีการเข้ารหัสข้อมูล(Encryption) ที่ฝั่งผู้ส่ง และ ถอดรหัสข้อมูล(Decryption)ที่ฝั่งผู้รับ ซึ่งการใช้ Secure Shell(SSH) ก็มีความสามารถตรงจุดนี้ โดยอาศัยวิธีคิด(Algorithm) การเข้ารหัสข้อมูลแบบต่างๆมาใช้ ทำให้การลักลอบอ่านหรือแก้ไขข้อมูลทำได้ยากขึ้น

วัตถุประสงค์

1. เพื่อพัฒนาโครงงานเดิมคือ IsagTerm 2542 และ IsagFTP ให้ใช้งานกับ SSH-2 ได้
2. เพื่อพัฒนาโปรแกรมให้สามารถใช้บน OS อื่นๆได้
3. เพื่อศึกษาแนวทางในการรับส่งข้อมูลแบบปลอดภัย
4. เพื่อเพิ่มความปลอดภัยในการติดต่อและส่งข้อมูลในระบบเครือข่าย

ปัญหาหรือประโยชน์ที่เป็นเหตุผลให้ควรพัฒนาโปรแกรม

การสื่อสารผ่านเครือข่ายอินเตอร์เนตยังไม่มีความปลอดภัย เนื่องจากข้อมูลที่ส่งไปบนเครือข่ายเป็นข้อมูลจริงที่ไม่มีการเข้ารหัส จึงต้องใช้โพรโตคอล SSH ซึ่งมีการเข้ารหัสข้อมูลมาปกป้องข้อมูลจากผู้ไม่ประสงค์ดี โดยโปรแกรมจะทำการเข้ารหัสข้อมูลที่ต้องการส่งไปบนเครือข่ายตามหลักการของโพรโตคอล SSH

เป้าหมายและขอบเขตของโครงการ

1. โปรแกรมด้วยภาษา JAVA เพื่อให้สามารถทำงานได้ทุกระบบปฏิบัติการบน JAVA Platform
2. รองรับทั้ง SSH Version 1 และ SSH Version 2
3. ตัวโปรแกรมมี GUI ที่สามารถใช้งานได้สะดวก
4. โปรแกรมให้สามารถทำงานพร้อมกันได้หลายๆ การเชื่อมต่อ
5. นำส่ง File อย่างปลอดภัยระหว่างเครื่อง Client กับ Server โดยฝั่งทาง Server จำเป็นต้องเปิดใช้ SSH ด้วย
6. สามารถเข้าถึงจากระยะไกลใช้งานเครื่องเซิร์ฟเวอร์โดยมีความปลอดภัยในการรับส่งข้อมูลโดยใช้อัลกอริทึมในการเข้ารหัสข้อมูลแบบต่างๆของ SSH เพื่อให้มีความปลอดภัยในการส่งรหัสหรือในการส่งข้อมูลมากยิ่งขึ้น

บทที่ 2

การเข้ารหัสและการคำนวณ

2.1 ระบบของเข้ารหัสข้อมูล (Cryptography System)

การเข้ารหัสข้อมูล คือ การทำให้ข้อมูลที่ต้องการ เป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูลปัจจุบัน โดยอาศัยหลักการของการเข้ารหัส (Encryption) และการถอดรหัส (Decryption)

การเข้ารหัสเป็นการเปลี่ยนรูปข้อมูล โดยผ่านรูปแบบและกระบวนการแปรรูปข้อมูล ทำให้ข้อมูลที่ส่งมีรูปแบบที่ไม่เหมือนเดิม เพื่อทำให้ข้อมูลเป็นความลับ

การถอดรหัสเป็นการแปลงข้อมูลที่ได้เข้ารหัสให้กลับมาเป็นข้อมูลเดิม ซึ่งการเข้ารหัสและการถอดรหัสจะถูกควบคุมโดยคีย์ (Key)

2.1.1 ระบบการเข้ารหัสโดยใช้คีย์เดียว (Symmetric Key)

การเข้ารหัสข้อมูลระบบนี้ทั้งผู้รับและผู้ส่งจะต้องมีคีย์ที่เป็นความลับ ที่เหมือนกันในการเข้าและถอดรหัสข้อมูล ซึ่งหากคีย์ที่มีต่างกัน ก็จะทำให้ข้อมูลที่สื่อสารกันผิดพลาด ตัวอย่างการเข้ารหัสระบบนี้ได้แก่ การเข้ารหัสแบบ DES, 3DES, IDEA เป็นต้น

รูปที่ 2-1 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์เดียว

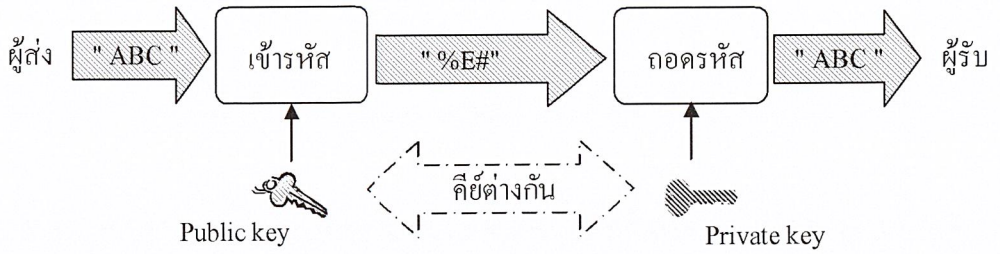
2.1.2 ระบบการเข้ารหัสแบบคีย์สาธารณะ

การเข้ารหัสข้อมูลระบบนี้จะประกอบด้วยคีย์ 2 อัน คือ

1. คีย์ส่วนตัว (private key) เป็นคีย์ที่จะต้องเก็บเป็นความลับ

2. คีย์สาธารณะ (public key) เป็นคีย์ที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งคีย์ทั้งสองนี้จะเป็นคีย์ที่ต่างกัน การมีวิธีในการเข้าและถอดรหัส ดังรูป



รูปที่ 2-2 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์สาธารณะ

2.2 รูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพี

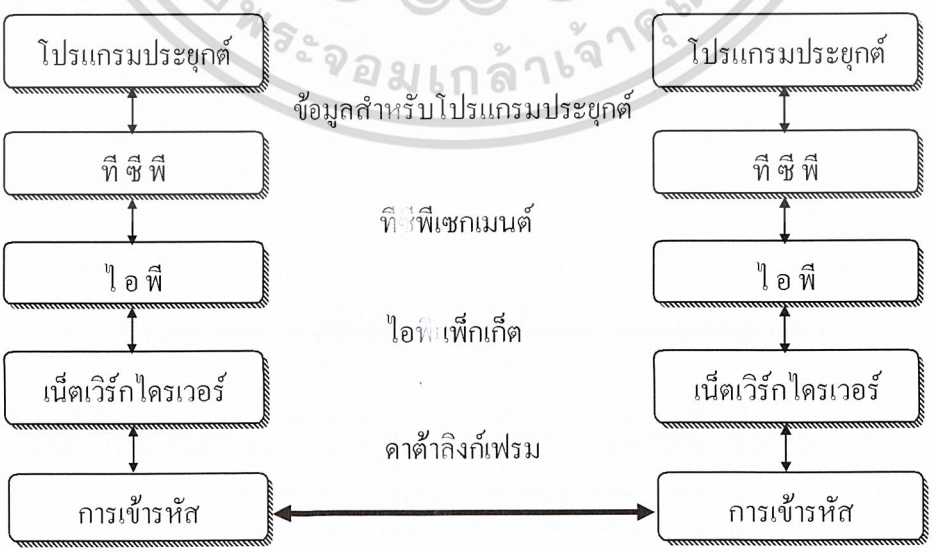
สำหรับรูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพี แบ่งออกเป็น 4 ระดับ ได้แก่

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

2.2.1 การเข้ารหัสระดับการเชื่อมโยงข้อมูล (Link Level Encryption)

เป็นการเข้ารหัสในระดับดาต้าลิงก์ โดยทั่วไปมักจะเป็นการใช้อุปกรณ์พิเศษที่เรียกว่า กล่องเข้ารหัส (encryption box) ซึ่งวิธีการนี้ถูกใช้ในระบบไซเฟอร์หรือการใช้ดีไวซ์ไดร์เวอร์

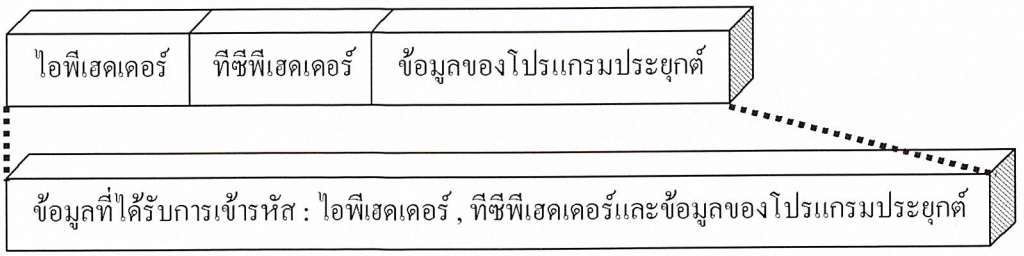
วิธีการนี้เป็นวิธีการที่ปลอดภัยที่สุด เนื่องจากข้อมูลถูกเข้ารหัสทั้งหมด นั่นคือทั้งข้อมูลที่ใช้งานและข้อมูลที่เป็นโครงสร้างของโพรโตคอล แต่ด้วยวิธีการนี้เป็นวิธีการที่ค่อนข้างมีค่าใช้จ่ายที่สูงเนื่องจากอุปกรณ์สำหรับการสื่อสารข้อมูลทั้งหมด จะต้องสนับสนุนการเข้ารหัสนี้ด้วย



รูปที่ 2-3 การเข้ารหัสในระดับการเชื่อมโยงข้อมูล

2.2.2 การเข้ารหัสระดับเครือข่าย (Network Level Encryption)

เป็นการเข้ารหัสในไอพีแพ็กเก็ต (IP packet) มีลักษณะดังภาพที่ 4-4

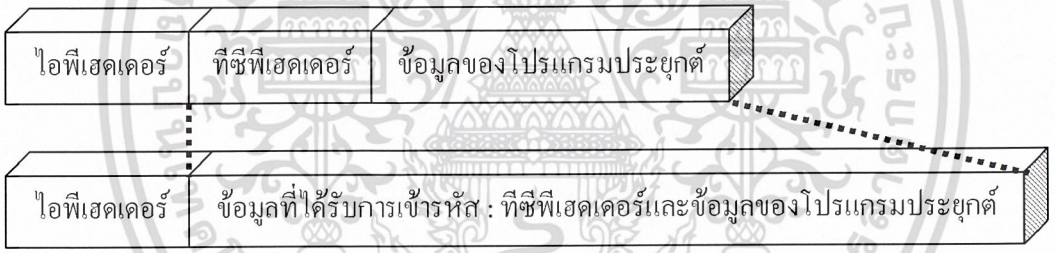


รูปที่ 2-4 การเข้ารหัสระดับเครือข่าย

2.2.3 การเข้ารหัสระดับทรานสปอร์ต (Transport Level Encryption)

การเข้ารหัสระดับทรานสปอร์ตเป็นการเข้ารหัสในส่วนของทีซีพีเซกเมนต์ ซึ่งได้แก่ ทีซีพีเฮดเดอร์และ ข้อมูลของโปรแกรมประยุกต์

การเข้ารหัสในรูปแบบนี้ทำให้การส่งข้อมูลไปตามเครือข่ายสามารถทำได้โดยไม่จำเป็นต้องเปลี่ยนแปลงอุปกรณ์ใด เนื่องจากโครงสร้างในส่วนของไอพีไม่มีการเปลี่ยนแปลง

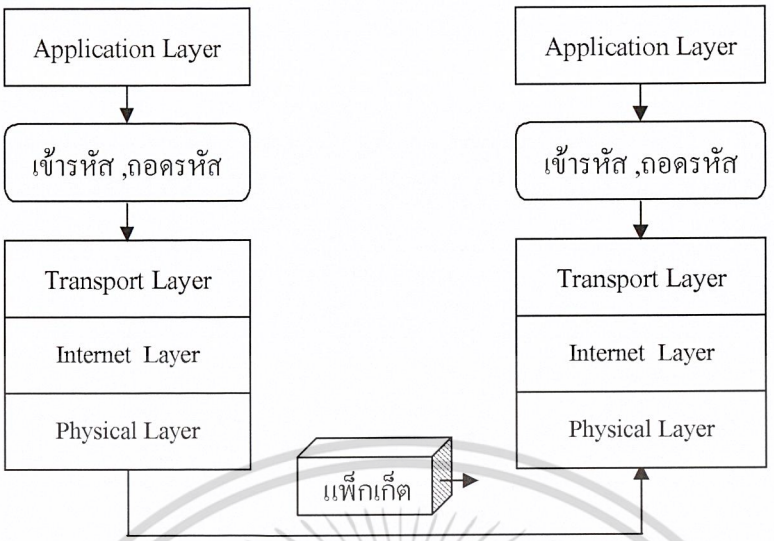


รูปที่ 2-5 การเข้ารหัสระดับทรานสปอร์ต

2.2.4 การเข้ารหัสระดับโปรแกรมประยุกต์ (Application Level Encryption)

การเข้ารหัสระดับโปรแกรมประยุกต์ เป็นการเข้ารหัสในลักษณะของการเข้ารหัสแบบจุดต่อจุด (end-to-end encryption) นั่นคือ โปรแกรมประยุกต์จะเข้ารหัสในส่วนข้อมูลของตัวเอง ก่อนที่จะส่งผ่านไปยังระดับล่าง ทำให้กระบวนการในระดับล่างไม่จำเป็นต้องมีการเปลี่ยนแปลงอะไร ซึ่งวิธีการนี้ถ้าต้องการนำมาใช้กับโปรแกรมประยุกต์เดิม เช่น เทลเนตหรือเอฟทีพี จะต้องแก้ไขโปรแกรมใหม่ ทั้งโปรแกรมขอรับบริการและโปรแกรมให้บริการ

สำหรับโครงงานนี้ จะทำการเข้ารหัสข้อมูลในช่วงระหว่างชั้นแอปพลิเคชั่นและชั้นทรานสปอร์ตในระบบโอเอสไอโมเดลของโพรโตคอลทีซีพี/ไอพี ดังภาพที่ 4-6



รูปที่ 2-6 แสดงการเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดลของทีซีพี/ไอพี

สำหรับการเข้ารหัสข้อมูลที่ใช้ในโครงงานนี้ จะใช้วิธีการเข้ารหัสข้อมูล คือ DES , 3DES (ระบบคีย์เดียว) และ RSA (ระบบคีย์สาธารณะ) เป็นหลักซึ่งรายละเอียดของการเข้ารหัสข้อมูลดังกล่าวจะมีดังต่อไปนี้

2.3 การเข้ารหัสแบบ DES (Data Encryption Standard)

2.3.1 ประวัติและที่มาของ DES

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

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

2.3.2 รายละเอียดของ DES

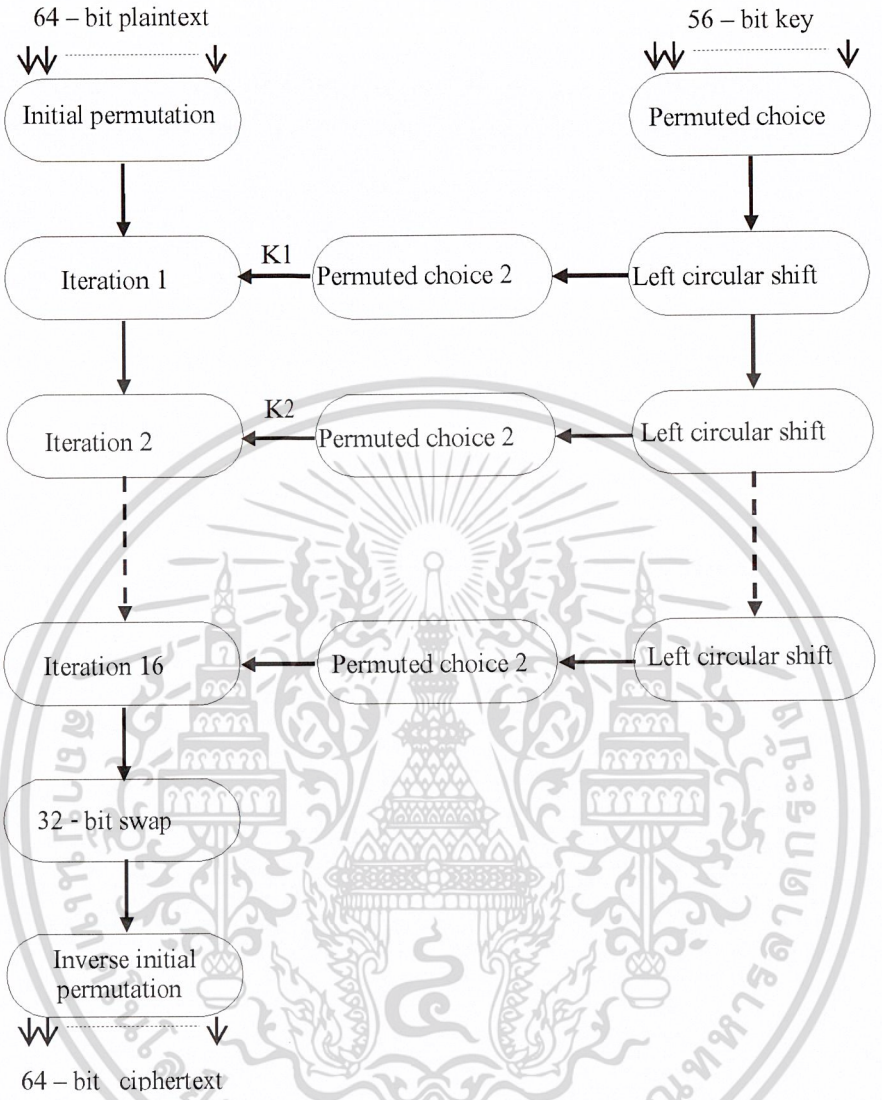
เป็นวิธีการเข้ารหัสที่ใช้กันอย่างแพร่หลายที่เป็นพื้นฐานบน Data Encryption Standard (DES) ที่ได้พัฒนาขึ้นในปี 1977 โดย National Bureau of Standards ซึ่งปัจจุบันคือ Federal Information Processing Standard 46 (FIPS PUB46) สำหรับ DES ข้อมูลจะถูกเข้ารหัสเป็นบล็อกขนาด 64 บิตซึ่งใช้คีย์ขนาด 56

บิต โดยวิธีการจัดการกับข้อมูล 64 บิตที่เข้ามาและแปลงเป็นข้อมูล 64 บิตออกไป จะใช้คีย์ตัวเดียวกันในการถอดรหัส

แม้ว่า DES ถูกนำมาใช้ตั้งแต่ช่วงทศวรรษที่ 70 (ค.ศ. 1960 – 1970) และได้รับการตอบรับเป็นอย่างดีจากเหล่านักวิเคราะห์รหัส (Cryptanalysis) แต่ก็เป็นข้อถกเถียงกันอย่างมากถึงเรื่อง DES นั้นจะปลอดภัยหรือไม่และมีความปลอดภัยมากน้อยแค่ไหน แต่จนถึงปัจจุบันเราก็ยังไม่พบช่องโหว่ของ DES ตามเอกสารที่ตีพิมพ์เป็นสาธารณะ แม้ว่าจะใช้คีย์เพียงไม่กี่บิตก็ตาม ในทางตรงกันข้ามแนวความคิดแบบ IDEA กลับใช้คีย์ขนาด 128 บิต (ซึ่งขนาดประมาณ 2 เท่าของ DES) และได้รับการตอบรับจากสาธารณะตั้งแต่ทศวรรษ 90 (ค.ศ. 1980 – 1990) (แต่ดีไม่เท่าตอนประกาศใช้ DES) IDEA มีความปลอดภัยมากกว่า DES และสามารถประมวลผลได้เร็วกว่า DES อย่างไรก็ตาม IDEA ยังต้องรอการตรวจสอบจากผู้เชี่ยวชาญอีกมากถึงเรื่องช่องโหว่ของความปลอดภัย

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

- เฟส 1 จะทำการจัดการกับข้อมูลที่ไม่ได้ผ่านการเข้ารหัสที่มีขนาด 64 บิตผ่านเข้าไปยังส่วนที่เรียกว่า Initial Permutation (IP) ซึ่งจะทำการเรียบเรียงบิตใหม่เพื่อผลิต “permuted input” ข้อมูลที่มีการสลับตำแหน่ง
- เฟสที่ 2 จะทำฟังก์ชันเดียวกัน 16 ครั้ง ซึ่งฟังก์ชันนี้รวมการทำ permutation และ substitution ซึ่งผลลัพธ์ที่ได้จากการทำทั้งหมด 16 ครั้งนี้จะได้ข้อมูลขนาด 64 บิตโดยใช้ทั้งข้อมูลที่ไม่ได้ผ่านการเข้ารหัสและคีย์ในการทำ ซึ่งข้อมูลที่มีขนาด 64 บิตที่ได้นี้แบ่งเป็น 2 ด้านคือซ้ายและขวา ทั้งหมดจะถูก สว็อพ เพื่อผลิต 64 บิตที่เป็น preoutput
- เฟสที่ 3 จะนำ preoutput ผ่านเข้าไปยังส่วนที่เรียกว่า “Inverse initial permutation” :IP^{-1} ซึ่งทำหน้าที่ inverse ตัว initial permutation function ซึ่งทั้งหมดจะผลิต 64 บิตที่เรียกว่า ข้อมูลที่ผ่านการเข้ารหัสแล้ว (Ciphertext)



รูปที่ 2-7 แสดงการขั้นตอนการทำงานของ DES

2.3.3 การสลับตำแหน่งข้อมูลเริ่มต้น (Initial Permutation)

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

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}	M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}

M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}	M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

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

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

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

2.3.4 รายละเอียดของการทำฟังก์ชันในแต่ละรอบ

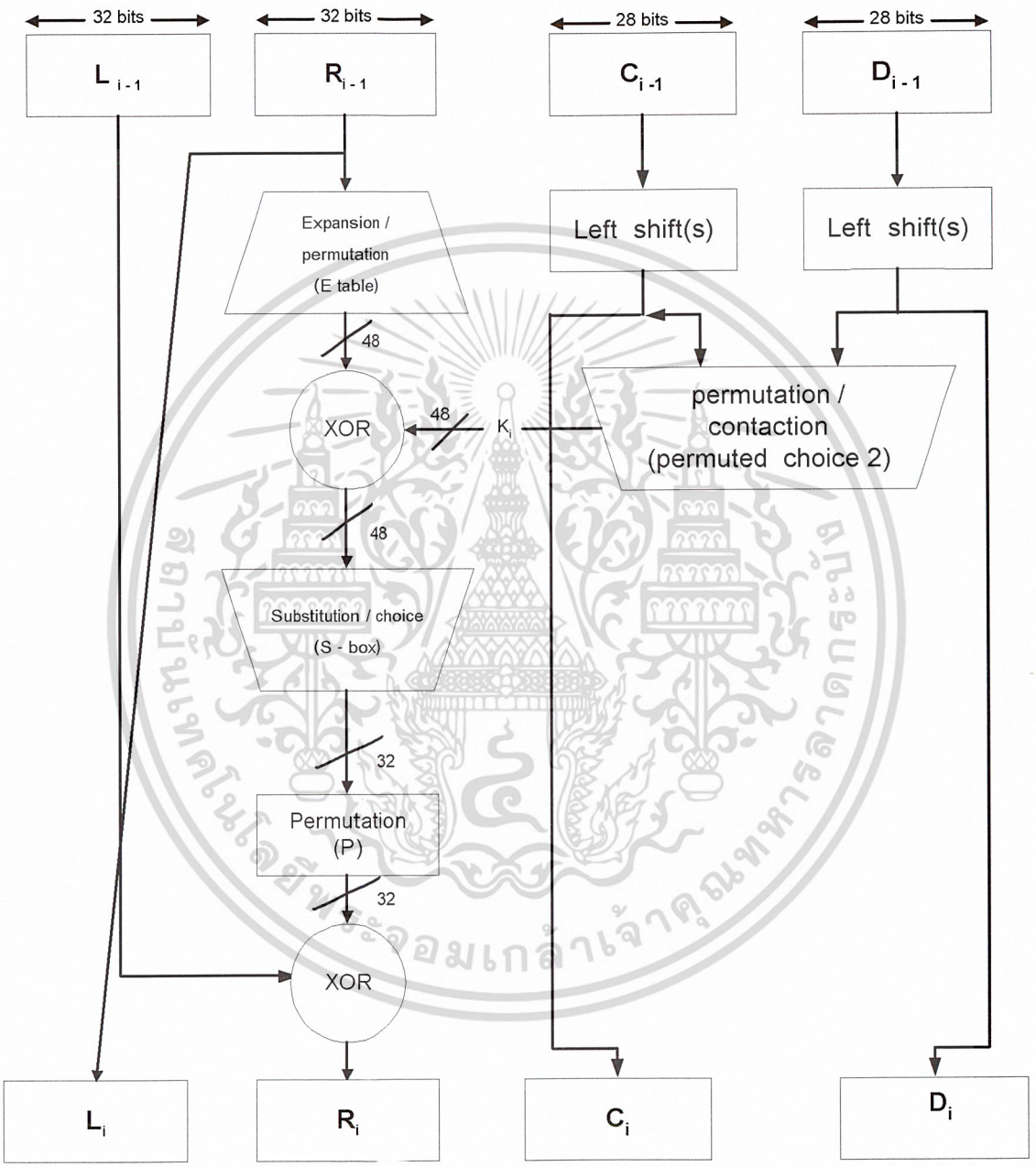
ข้อมูลที่เข้ามามีขนาด 64 บิต โดยจะทำการแบ่งข้อมูลเป็น 2 ส่วนด้วยกันขนาดละ 32 บิตเท่ากัน (แบ่งเป็นซ้ายกับขวา) ซึ่งกระบวนการทำในแต่ละครั้งสามารถสรุปเป็นสูตรได้ดังนี้

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

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

เมื่อ $\oplus$ หมายถึง การทำ XOR function

จากสูตรจะเห็นได้ว่า 32 บิตด้านซ้ายมือ(L_I) จะเท่ากับด้านขวาของ (R_{I-1}) รอบที่ผ่านมา โดย R_I จะเท่ากับการนำ L_{I-1} มา XOR กับ $f(R_{I-1}, K_I)$ ซึ่งฟังก์ชัน f จะแสดงดังรูปที่ 2.8



รูปที่ 2-8 แสดงการเข้ารหัส DES ในแต่ละครั้ง(ทั้งหมดทำ 16 ครั้ง)

(a) Initial Permutation (IP)

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

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

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

(c) Expansion Permutation(E)

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

(d) Permutation Function (P)

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

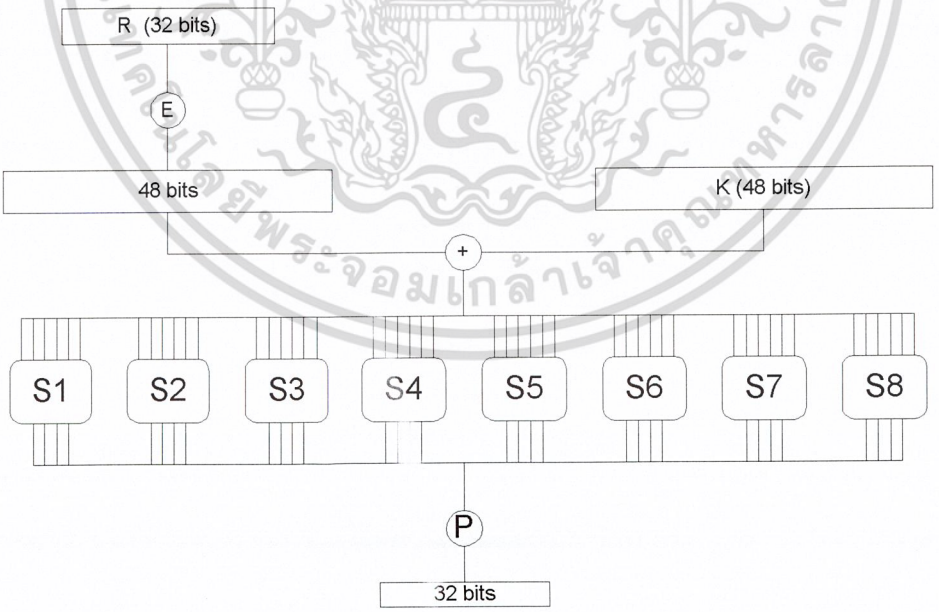
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Form input bit	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

ตารางที่ 2-1 แสดง Permutation ของ DES

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

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

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



รูปที่ 2-9 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	S1
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	S2
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	

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	S3
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	S5
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	11	14	2	13	6	15	0	9	10	4	5	3	

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

S6

Column Number

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

Box

S7

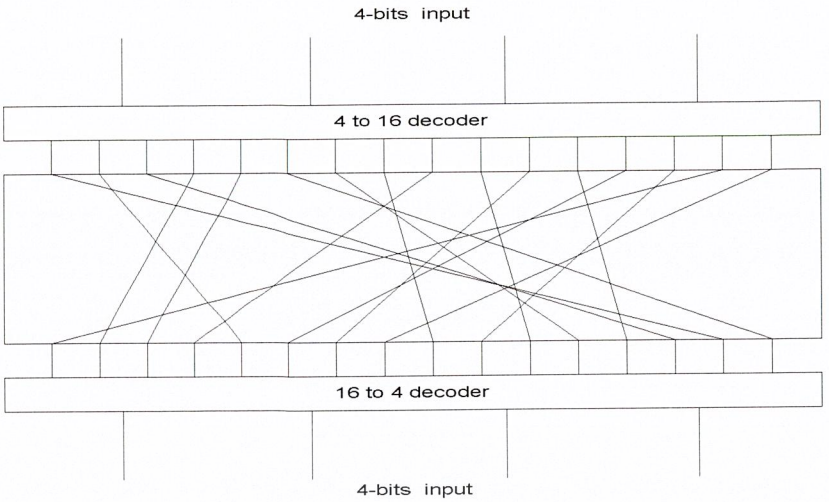
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S8

ตารางที่ 2-2 แสดงขั้นตอนใน S- box ทั้ง 8 ชุด

2.3.5 การสร้างคีย์ (Key Generation)

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



รูปที่ 2-10 แสดงการทำ Permutation Choice

(a) Permuted Choice One(PC-1)

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

(b) Permuted Choice Two (PC-2)

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

(c) Schedule of Left Shifts

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

ตารางที่ 2-3 แสดงการสร้างคีย์ในแต่ละรอบฟังก์ชัน

2.3.6 การถอดรหัสข้อมูล DES

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

เราจะแสดงถึงผลลัพธ์ของขั้นตอนแรกในการกระบวนการถอดรหัส ซึ่งจะเท่ากับ 32 บิตที่ถูกสว็อพจากอินพุตของการทำทั้งหมด 16 รอบของการเข้ารหัส เริ่มจาก

$$L16 = R15$$

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

ในด้านการถอดรหัส

$$Ld1 = Rd0 = L16 = R15$$

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

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

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

ซึ่งคุณสมบัติของ XOR ที่สำคัญคือ

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

$$D \oplus D = 0$$

$$E \oplus 0 = E$$

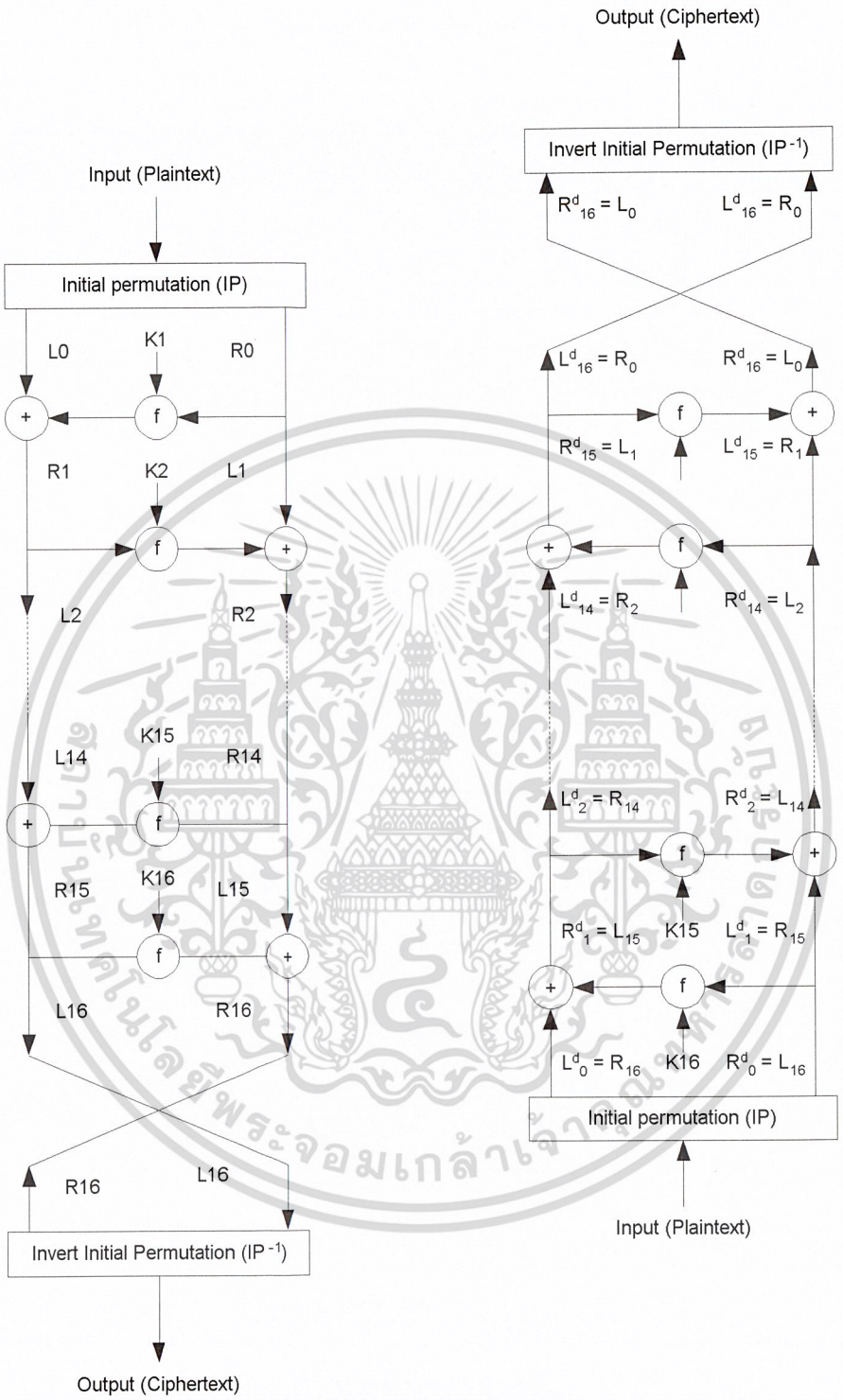
ดังนั้นเรามี Ld1 = R15 และ Rd1 = L15 จะได้เอาท์พุตของขั้นตอนแรกในการถอดรหัสคือ L15||R15 เราสามารถเขียนเป็นสมการในการถอดรหัสได้ดังนี้คือ

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

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

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

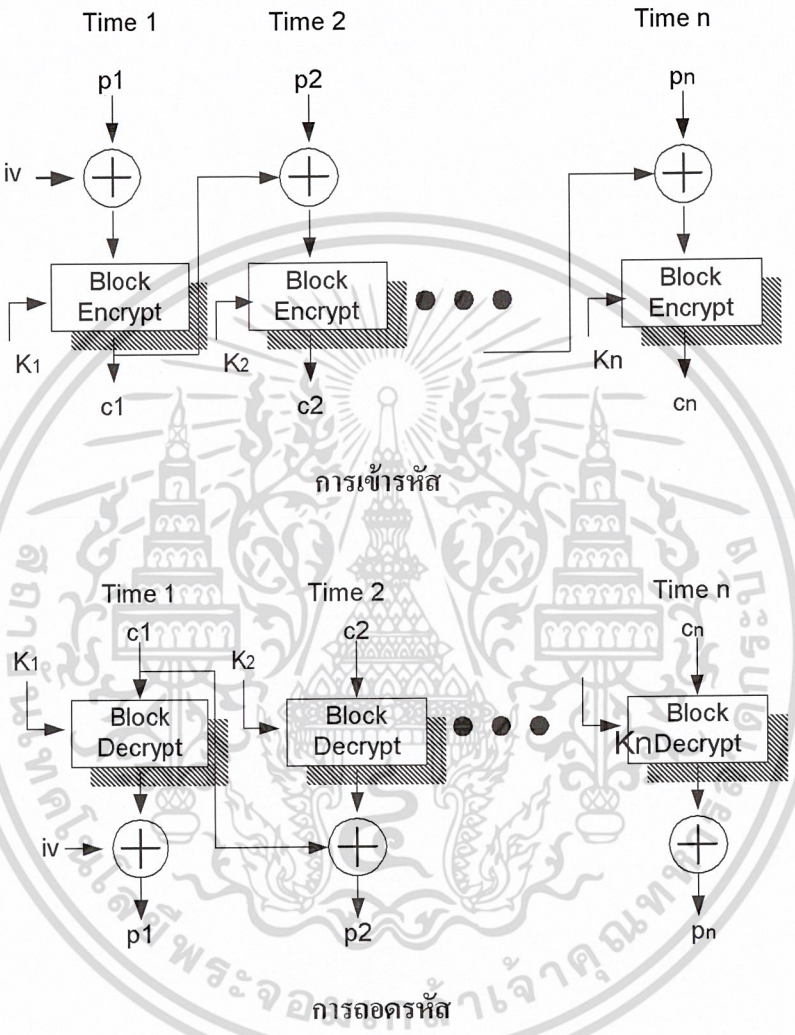
$$IP^{-1}(L0||R0) = IP^{-1}(IP(plaintext)) = plaintext$$



รูปที่ 2-11 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES

2.3.7 โหมด CBC (Cipher Block Chaining)

เป็นวิธีการเข้ารหัสที่พัฒนามาจาก DES ช่วยทำให้ข้อมูลที่ส่งยิ่งมีความปลอดภัยมากยิ่งขึ้น โดยมีวิธีการคือ จะนำข้อมูลที่ผ่านการเข้ารหัสของข้อมูลตัวก่อนมา XOR กับข้อมูลที่ยังไม่ได้เข้ารหัสของตัวถัดไปก่อนจะทำการเข้ารหัสแบบ DES ตามปกติดังรูป



รูปที่ 2-12 แสดงการเข้ารหัสและถอดรหัสของ DES CBC

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

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

สมการในการถอดรหัส

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

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

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

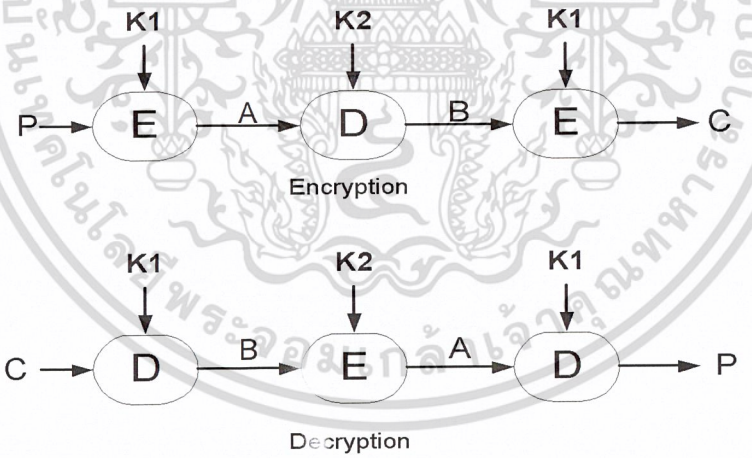
จะสังเกตเห็นได้ว่าบล็อกแรกของข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) จะมีการนำ iv (Initialization Vector) มาทำการ XOR กับข้อมูลที่ไม่ได้เข้ารหัส (plaintext) ก่อนจะมีการเข้ารหัส DES ตามปกติ และในส่วนของการถอดรหัสข้อมูลก็จะใช้ iv ในการถอดรหัสข้อมูลเช่นเดียวกัน ดังนั้นจึงจำเป็นต้องมีข้อตกลงกันระหว่างผู้ส่งกับผู้รับก่อนว่าจะใช้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

2.4 การเข้ารหัสแบบ 3DES (Triple DES)

เป็นที่ทราบกันว่ายิ่งคีย์มีความยาวมากขึ้น การถอดรหัสยิ่งทำได้ยากยิ่งขึ้น นอกจากนี้เราก็สามารถเพิ่มสมรรถนะของการเข้ารหัสให้มากขึ้นได้โดย การเข้ารหัสหลายๆ ครั้ง

อย่างไรก็ตามการเข้ารหัสครั้งที่สองโดยใช้คีย์ที่ต่างจากครั้งแรกนั้น มิได้หมายความว่าข้อมูลจะมีความปลอดภัยเพิ่มขึ้นเป็น 2 เท่า ถ้าการเข้ารหัสดังกล่าวเป็นการเข้ารหัสโดยเทคนิคทางคณิตศาสตร์

DES มิได้อยู่ในกลุ่มดังกล่าว แต่การที่จะทำให้ DES มีความปลอดภัยมากขึ้นเป็น 2 เท่านั้นจำเป็นที่เราจะต้องเข้ารหัสถึง 3 ครั้ง



รูปที่ 2-13 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES

ขั้นตอนของการเข้ารหัสแบบ 3 ครั้งนั้นมีดังนี้

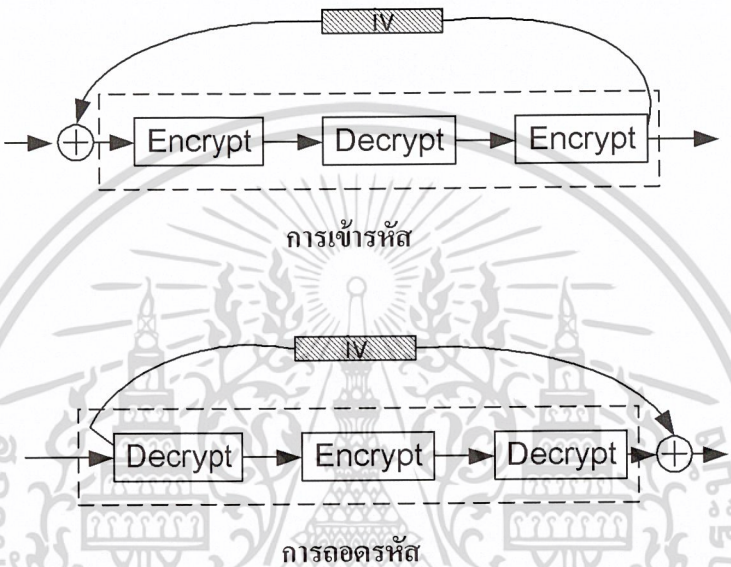
1. เข้ารหัสโดยใช้คีย์ตัวแรก
2. ถอดรหัสโดยคีย์ตัวที่สอง
3. เข้ารหัสโดยใช้คีย์ตัวที่สาม

(อาจใช้คีย์ตัวที่ 1 และ 3 เป็นตัวเดียวกันก็ได้ แต่ประสิทธิภาพของการเข้ารหัสก็จะลดลง)

2.4.1 Triple DES โหมด CBC (3DES CBC Mode)

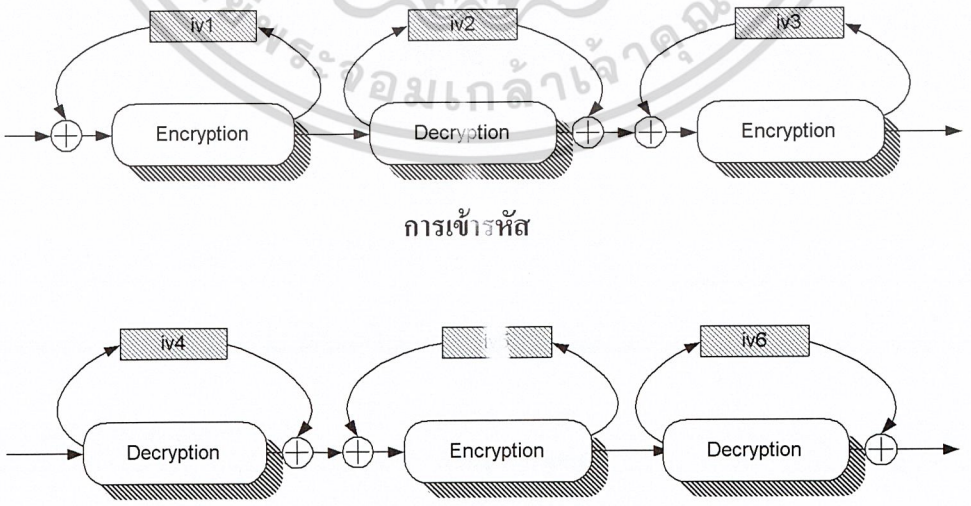
เนื่องจากการเข้ารหัสแบบ DES นั้นมีโหมดในการเข้ารหัสแบบ CBC ซึ่งการเข้ารหัส แบบ 3DES นี้ได้มีการประยุกต์มาจาก DES จึงได้มีการพัฒนา 3DES นี้ให้มีโหมด CBC ด้วย ซึ่งในการเข้ารหัสแบบ 3DES นี้ได้พัฒนาโหมดนี้ออกเป็น 2 แบบ

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



รูปที่ 2-14 แสดงการเข้าและถอดรหัสแบบ Triple DES โหมด inner CBC

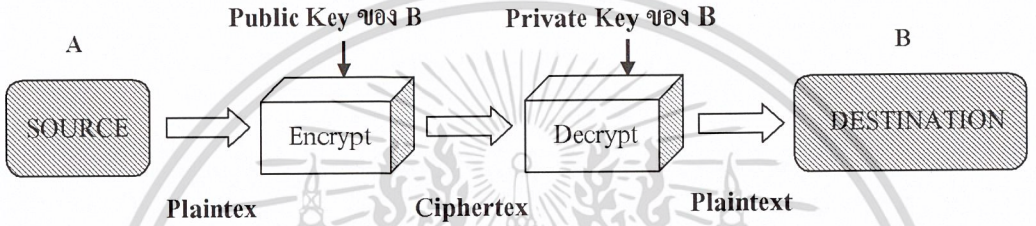
- **แบบ outer CBC** การเข้ารหัสแบบนี้แต่ละโมดูลในการเข้าหรือถอดรหัสจะมี iv เฉพาะของแต่ละโมดูล ดังรูป 4-15



การถอดรหัสรูปที่ 2-15 แสดงการเข้าและถอดรหัสแบบ *Triple DES โหมด outer CBC*

2.5 การเข้ารหัสแบบ RSA

RSA ถูกคิดค้นขึ้นในปี 1977 โดยที่ชื่อ RSA ได้มาจากอักษรตัวแรกของนามสกุลของผู้ร่วมกันคิดค้นคือ Ron River, Adi Shamir และ Leonard Adleman เป็นวิธีการเข้ารหัสแบบคีย์สาธารณะที่เรียกว่า พับลิกคีย์ (Public–Key Crytosystem) เพื่อแก้ไขปัญหาการทำให้คีย์เป็นความลับ (Secret–Key Cryptosystem) โดยสมาชิกแต่ละคนจะต้องมีคีย์ 2 ชนิดคือ คีย์ส่วนตัวหรือไพรเวตคีย์ (Private Key) และคีย์สาธารณะหรือพับลิกคีย์ (Public Key) โดยคีย์ส่วนตัวจะถูกเก็บไว้เป็นความลับ ส่วนคีย์สาธารณะนั้นจะเปิดเผยให้ใครก็ได้ที่ต้องการส่งเอกสารให้แก่ตน หลักการทำงานของการเข้ารหัสแบบ RSA คือ ข้อมูลที่ถูกเข้ารหัสด้วยคีย์สาธารณะของผู้ใด จะถูกถอดรหัสได้ด้วยคีย์ส่วนตัวของผู้นั้นเท่านั้น การทำงานของระบบพับลิกคีย์สามารถอธิบายได้ดังต่อไปนี้



รูปที่ 2-16 แสดงขั้นตอนการทำ Public – Key Cryptosystem

2.5.1 หลักการทำงานของ RSA

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

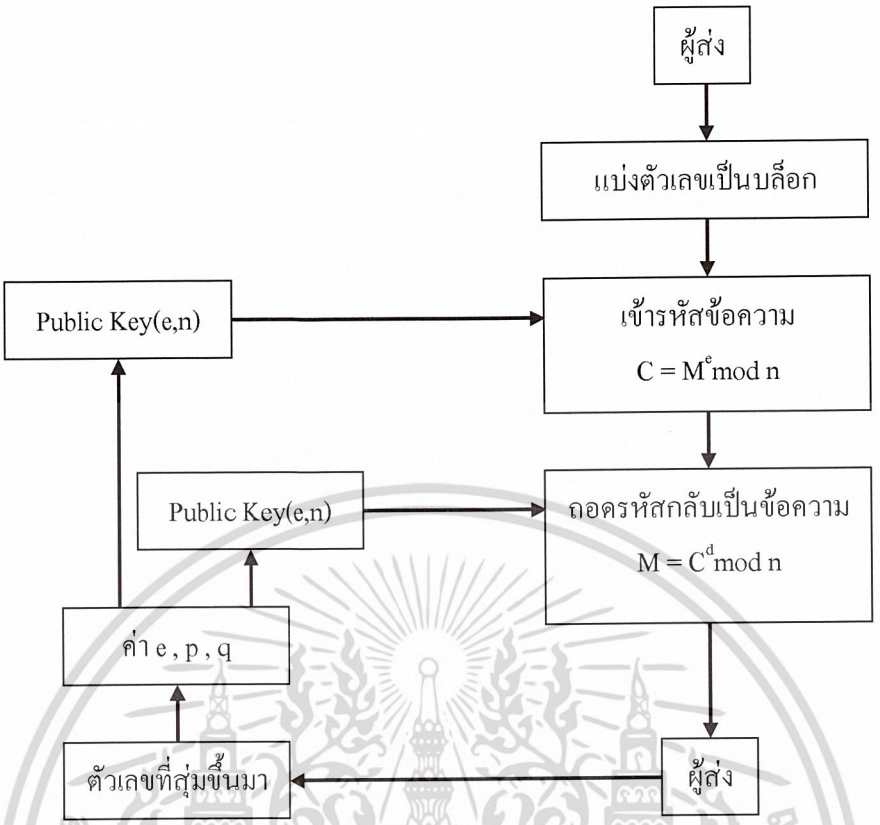
$$e*d \bmod (p-1)(q-1) = 1 \quad \text{.....(1)}$$

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

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

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

และเพราะมีแต่บุญมีเท่านั้นที่รู้ค่า d บุญมีเท่านั้นจึงจะถอดรหัสได้ คุณสมบัติสำคัญประการหนึ่งของ RSA คือจากสมการ (2) ในทางกลับกันถ้าเราเข้ารหัสด้วยไพรเวตคีย์ เราก็สามารถถอดรหัสด้วยพับลิกคีย์ได้ด้วยเช่นกัน คุณสมบัติข้อนี้เองที่ทำให้ RSA มีประโยชน์มาก เพราะใช้ในการเซ็นระบบดิจิตอลได้อีกด้วย



รูปที่ 2-17 แสดงการเข้ารหัสแบบ RSA

ตัวอย่างขั้นตอนการเข้ารหัสแบบ RSA ซึ่งมีขั้นตอนในการหาคีย์ ดังนี้

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

   77 * 5 = 385 = 4 * 96 +1

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

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

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

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

2.6 การคำนวณแบบ MD5

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

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

ขั้นที่ 2 : เติมความยาว ใส่ขนาดของข้อความโดยจะมีค่าไม่เกิน 2^{64}

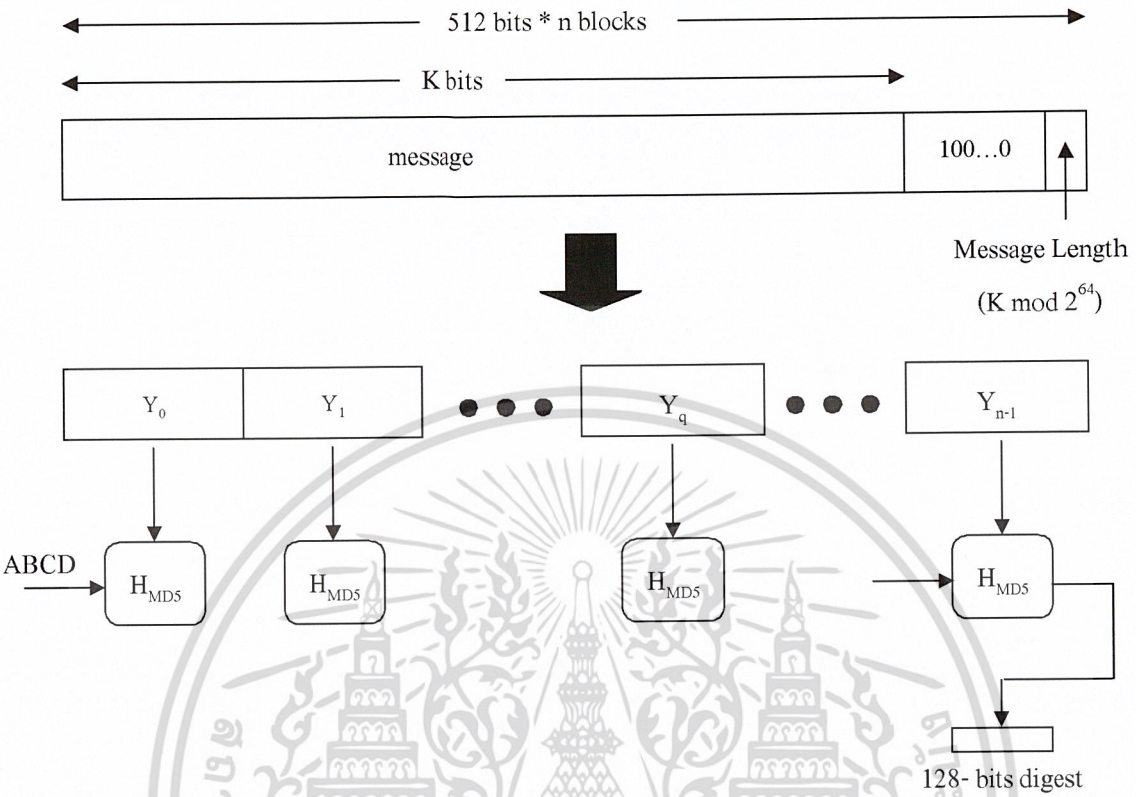
ขั้นที่ 3 : กำหนดค่าเริ่มของ MD บัฟเฟอร์ บัฟเฟอร์จะมีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการ hash โดยบัฟเฟอร์เริ่มต้นจะประกอบด้วย 32 บิตไบต์ 4 ตัว (A,B,C,D) ซึ่งมีค่าต่อไปนี้

A = 01234567

B = 89ABCDEF

C = FEDCBA98

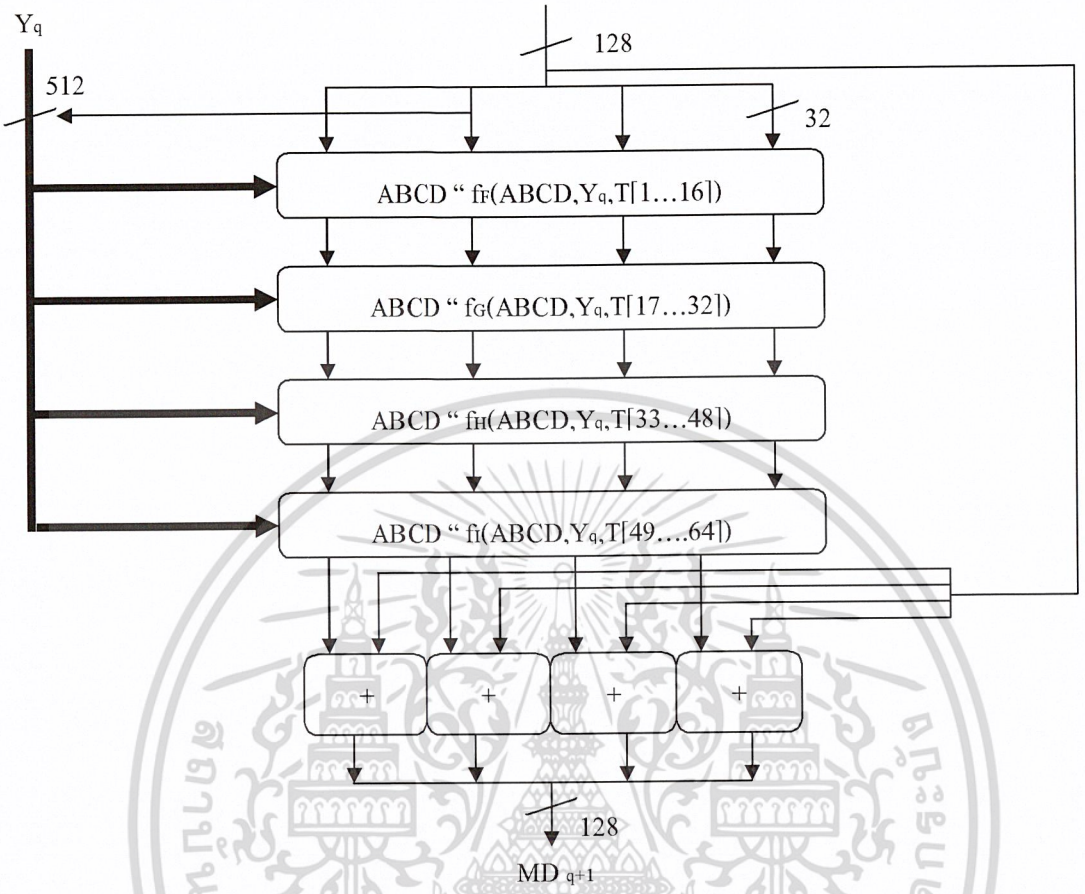
D = 76543210



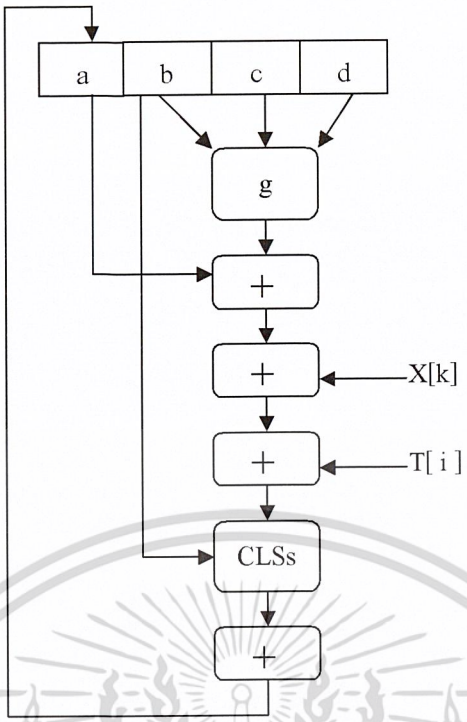
รูปที่ 2-18 แสดงการคำนวณแบบ MD5

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

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



รูปที่ 2-19 กระบวนการทำในหนึ่งบล็อกของ MD5



รูปที่ 2-20 แสดงการทำหนึ่ง operation ของ MD5 [abcd k s i]

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

และ

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

ขั้นที่ 5: Output

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

บทที่ 3

โพรโตคอลเอสเอสเอช 1

โพรโตคอลเอสเอสเอช 1 เป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่ผู้ใช้สามารถล็อคอินเข้าใช้บริการจากเครื่องให้บริการที่อยู่ห่างไกลบนระบบเครือข่าย และสามารถทำการประมวลผล คำสั่งในเครื่องให้บริการที่อยู่ห่างไกลได้ (remote machine) พร้อมทั้งสามารถแลกเปลี่ยนข้อมูลบนระบบเครือข่ายอินเตอร์เน็ตได้อย่างปลอดภัย เนื่องจากมีวิธีการพิสูจน์ความเป็นเจ้าของที่มีประสิทธิภาพ (Strong Authentication) และระบบการติดต่อที่ปลอดภัย (Secure Communication) บนระบบเครือข่าย

3.1 ภาพรวมของโพรโตคอลเอสเอสเอช 1

โพรโตคอลเอสเอสเอช 1ประกอบด้วยโปรแกรมในส่วนของเครื่องที่ให้บริการ (เซิร์ฟเวอร์) และอีกส่วนหนึ่งคือส่วนของเครื่องใช้บริการ (ไคลเอ็นต์) โดยการติดต่อระหว่างเครื่องให้บริการและเครื่องใช้บริการจะติดต่อกันโดยใช้หมายเลขไอพี (IP Address) และ ทีซีพี/ไอพีซ็อกเก็ต (TCP/IP socket) ซึ่งเป็นการติดต่อสองทิศทาง (bi-directional communication)

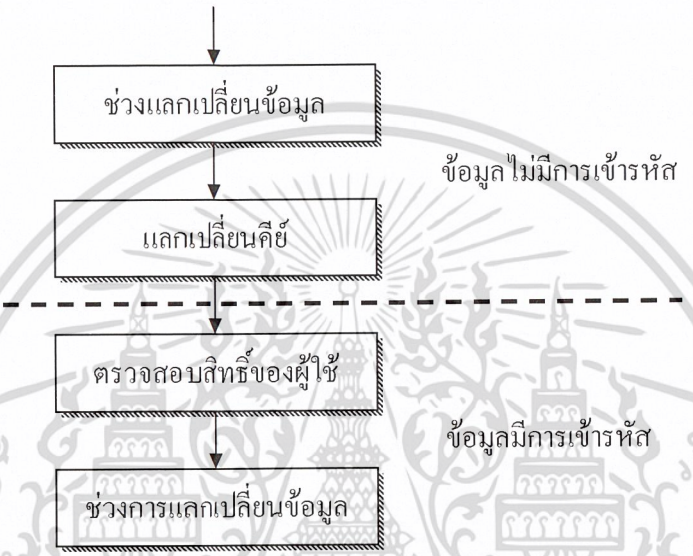
การติดต่อกันจะเริ่มการติดต่อจากเครื่องใช้บริการ โดยเครื่องให้บริการจะรอฟังจากพอร์ตที่กำหนดไว้เฉพาะ ว่ามีเครื่องที่จะใช้บริการติดต่อเข้ามายังพอร์ตนั้นหรือไม่ ซึ่งเครื่องให้บริการสามารถติดต่อกับเครื่องใช้บริการได้หลายเครื่องพร้อมๆ กัน

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

หลังจากทำการตรวจสอบเวอร์ชันของกันและกันและกันแล้ว ข้อมูลในการติดต่อจะเปลี่ยนมาเป็นรูปแบบของแพ็กเก็ตซึ่งเรียกโพรโตคอลในการติดต่อนี้ว่า *Binary packet protocol* โดยเครื่องผู้ให้บริการจะเริ่มส่งการติดต่อโดยส่งคีย์สาธารณะที่เรียกว่า Host Public Key, Server Key (สำหรับรายละเอียดเกี่ยวกับคีย์ต่างๆ จะอยู่ในบทที่ 4) และข้อมูลอื่นๆไปยังเครื่องผู้ใช้บริการ ตัวผู้ใช้บริการจะทำการผลิตคีย์เซสชัน (Session Key) ซึ่งใช้ในการ ้ารหัสข้อมูลที่มีขนาด 236 บิต ซึ่งจะถูกเข้ารหัสแบบ RSA 2 ครั้ง และทำการส่งไปให้เครื่องผู้ให้บริการพร้อมทั้งชนิดของการเข้ารหัสข้อมูล (cipher) ที่ใช้ หลังจากนั้นทั้ง 2 ฝั่งจะเปิดการติดต่อกันโดยข้อมูลที่ติดต่อกันนี้จะมีการเข้ารหัสโดยใช้คีย์และวิธีการที่ได้กำหนดไว้ข้างต้น

ต่อมาเครื่องผู้ใช้บริการจะทำการตรวจสอบสิทธิ์ของตน โดยทำการส่งหมายเลขเพื่อบ่งบอกวิธีในการพิสูจน์ตนและทำการพิสูจน์ตนตามแบบที่ได้ส่งไปเซิร์ฟเวอร์

เมื่อทำการตรวจสอบและพิสูจน์ตนแล้วว่าถูกต้อง เครื่องผู้ใช้บริการจะทำการส่งตัวเลขเพื่อทำการเตรียมตัวสำหรับการเรียกใช้เชลล์ (Shell) และทำการเรียกใช้เชลล์เพื่อที่จะเข้าไปทำการติดต่อในโหมดเซสชันแบบอินเตอร์แอ็กทีฟ (Interactive Session Mode) ซึ่งการทำงานในโหมดนี้เป็นโหมดในการแลกเปลี่ยนข้อมูลกันและกันระหว่างผู้ที่ใช้บริการกับเครื่องผู้ให้บริการและจะสิ้นสุดการติดต่อเมื่อเครื่องผู้ให้บริการส่งคำสั่งเลิกบริการ (Exit Status) ของโปรแกรมไปยังเครื่องผู้ใช้บริการ



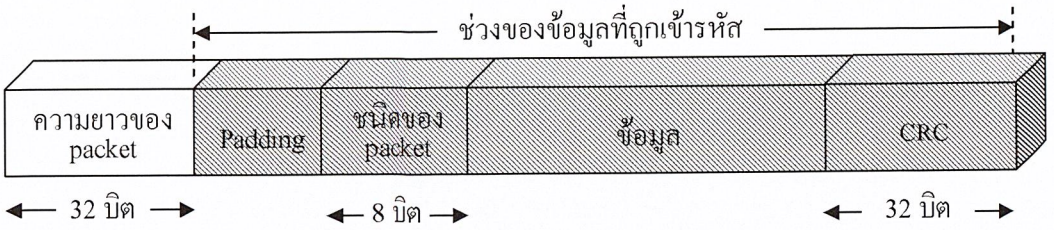
รูปที่ 3-1 แสดงขั้นตอนของโพรโตคอลเอสเอสเอช 1

3.2 ลักษณะและโพรโตคอลของข้อมูล

จากภาพรวมของโพรโตคอลที่ได้กล่าวมาข้างต้น ทำให้เราทราบวิธีการทำงาน ขั้นตอนการติดต่อและรูปแบบข้อความที่ใช้ติดต่อกัน แต่ก่อนที่เราจะศึกษาขั้นตอนการทำงานอย่างละเอียด เราจะมาทำความรู้จักกับคำที่ได้กล่าวมาข้างต้นก่อนว่าแต่ละคำคืออะไร ทำหน้าที่อย่างไร และข้อความที่ใช้มีอะไรบ้าง

3.2.1 ไบนารีแพ็กเก็ตโพรโตคอล (Binary Packet Protocol)

ภายหลังจากการทำการตรวจสอบเวอร์ชันของกันและกัน ทั้งเครื่องให้บริการและเครื่องผู้ใช้บริการจะทำการส่งแพ็กเก็ตที่มีรูปแบบเฉพาะซึ่งเรียกว่า Binary Packet Protocol รูปแบบของแพ็กเก็ตเป็นดังนี้



รูปที่ 3-2 แสดงฟิลด์ต่างๆในแพ็กเก็ตของระบบ Binary Packet Protocol

ความยาวของแพ็กเก็ต มีขนาด 32 บิตซึ่งเป็นจำนวนเต็มบวก แบ่งออกเป็น 8 บิต/ไบต์ บิตแรกเป็นบิตที่มีค่าสูงสุด ความยาวของแพ็กเก็ต จะไม่นับรวมฟิลด์ของความยาวของแพ็กเก็ตและ padding ความยาวสูงสุดของแพ็กเก็ต คือ 26,244 ไบต์

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

ชนิดของแพ็กเก็ต จะมีขนาด 8 บิต ค่า 233 กันไว้สำหรับใช้ในอนาคต

ข้อมูลที่จะส่ง ข้อมูล (มีลักษณะเป็นแบบเลขฐานสอง) ที่จะส่ง โดยขึ้นกับชนิดของแพ็กเก็ต จำนวนของข้อมูลมีขนาดเท่ากับความยาวของแพ็กเก็ตลบ 3

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

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

3.2.2 การเข้ารหัสแพ็กเก็ต (Packet Encryption)

แพ็กเก็ตของโพรโตคอลนี้รองรับการเข้ารหัสได้หลายชนิด โดยตั้งแต่เริ่มต้นการติดต่อ เครื่องผู้ให้บริการจะส่งบิตมาร์ค (bitmask) ของวิธีการเข้ารหัสแบบต่างๆที่เครื่องผู้ให้บริการรองรับ ตัวเครื่องผู้ใช้บริการจะเลือกวิธีการในการเข้ารหัสข้อมูลและสร้างเซสชันคีย์ ขนาด 236 บิต จากนั้นทำการส่งไปให้เครื่องผู้ให้บริการ

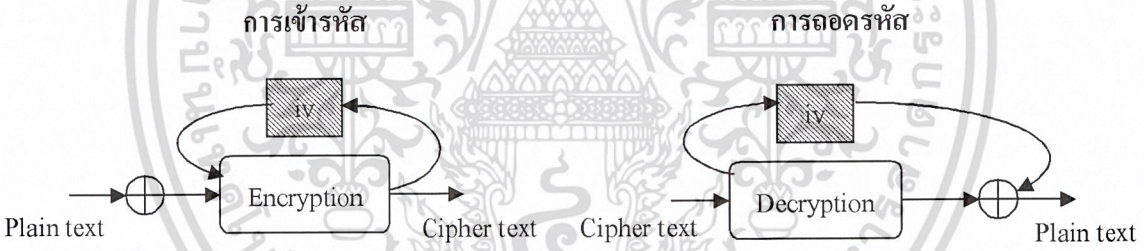
วิธีการเข้ารหัสที่รองรับและโค้ดของวิธีการเข้ารหัสแต่ละแบบคือ

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 3-1 แสดงวิธีการเข้ารหัสที่โพรโตคอลเอสเอสเอช 1รองรับ

SSH_CIPHER_DES

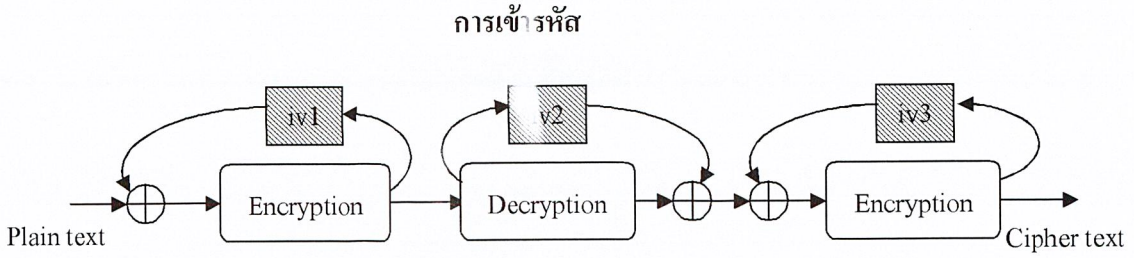
กุญแจในการเข้ารหัสนำมาจาก 8 บิตแรกของเซสชันคีย์ และบิตที่มีค่าน้อยสุดของแต่ละไบต์จะถูกตัดทิ้งไป กุญแจที่ได้จะมีขนาด 36 บิต การเข้ารหัสแบบ DES ที่ใช้นี้จะเป็นโหมด CBC ซึ่ง iv (initialization vector) ในการเริ่มต้นถูกกำหนดให้เป็น 0 ทั้งหมด (ซึ่งได้อธิบายละเอียดในบทการเข้ารหัสข้อมูล)



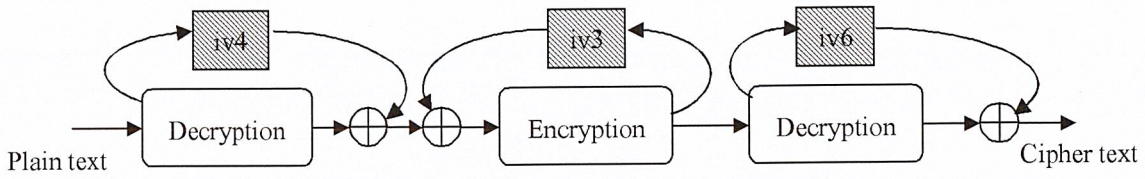
รูปที่ 3-3 แสดงการเข้ารหัสและถอดรหัสแบบ DES โหมด CBC

SSH_CIPHER_3DES

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



การถอดรหัส



รูปที่ 3-4 แสดงวิธีการเข้ารหัสและถอดรหัสแบบ 3DES โหมด CBC

สำหรับการเข้ารหัส ข้อมูลจะถูกเข้ารหัสด้วยไซเฟอร์แรกและถอดรหัสด้วยไซเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกเข้ารหัสอีกครั้งด้วยไซเฟอร์ 3 ส่วนการถอดรหัส ข้อมูลจะถูกถอดรหัสด้วยไซเฟอร์แรกและเข้ารหัสด้วยไซเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกถอดรหัสอีกครั้งด้วยไซเฟอร์ 3 โดยคีย์ที่ใช้สำหรับไซเฟอร์แรกจะถูกนำมาจาก 8 ไบต์แรกของคีย์เซสชัน และคีย์สำหรับไซเฟอร์ที่ 2 มาจาก 8 ไบต์ถัดไปของคีย์เซสชัน และคีย์สำหรับไซเฟอร์ที่ 3 จะเป็น 8 ไบต์ถัดไปของคีย์เซสชันเช่นกัน (ค่า iv จะไม่เกี่ยวข้องกัน)

3.2.3 ชนิดของข้อมูลที่ถูกเข้ารหัส (Data Type Encoding)

ฟิลด์ข้อมูลของแต่ละแพ็กเก็ตจะเก็บข้อมูลที่เข้ารหัสตามวิธีที่ได้อธิบายไป ซึ่งจะประกอบรายการของข้อมูลหลายๆตัว แต่ละรายการจะมีรหัสในแต่ละแบบและถูกนำมาใช้โดยการนำแต่ละรายการมาต่อกัน

ข้อมูลแต่ละชนิดจะถูกเก็บตามนี้คือ

- **8 bit byte** เป็นข้อมูลที่บรรจุในไบต์ๆ เดี่ยว (มีขนาด 8 บิต)
- **32 bit unsigned integer** เก็บข้อมูลโดยแบ่งเป็น 4 ไบต์ เริ่มต้นด้วยบิตที่มีค่าสูงสุด (MSB first)
- **Arbitrary length binary string** ข้อมูล 4 ไบต์แรกจะเป็นความยาวของข้อความ (String) เริ่มต้นด้วยบิตที่มีค่าสูงสุด (ไม่นับรวมความยาวของตัวมันเอง) ตามด้วยข้อความ (String) ซึ่งเป็นค่าของตัวอักษรเรียงกัน (ไม่นับรวมตัวอักษรปิดข้อความ)
- **Multiple – precision integer** ข้อมูล 2 ไบต์แรกเป็นจำนวนของบิตในรูปแบบของตัวเลข โดยเริ่มต้นด้วยบิตที่มีค่าสูงสุด (ตัวอย่างเช่น ค่า 0x00012343 จะมี 17 บิต) ค่า 0 จะมีจำนวน 0 บิต ซึ่งเป็นการอนุญาตให้จำนวนของบิตสามารถมากกว่าจำนวนบิตจริงของเลข ข้อมูลบอกจำนวนของบิตจะตามด้วยข้อมูลค่าของเลขจำนวนเต็มซึ่งมีการเก็บข้อมูลขนาด (จำนวนบิต + 7)/ 8 ไบต์ โดยเริ่มต้นด้วยบิตที่มีค่าสูงสุด

3.2.4 หมายเลขทีซีพี/ไอพีพอร์ต (TCP/IP Port Number and Other Option)

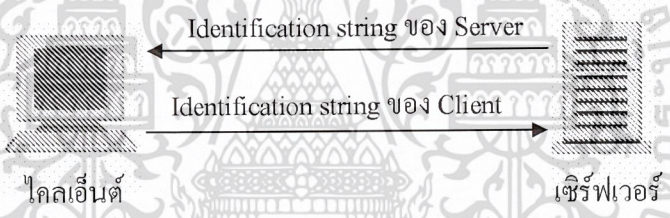
ตัวผู้ให้บริการจะรอติดต่อบนโพรโตคอลทีซีพี/ไอพีที่พอร์ต 22 ตัวผู้ใช้บริการจะใช้พอร์ตใดของตนก็ได้ในการติดต่อเข้ากับเครื่องผู้ให้บริการ แต่ถ้าตัวเครื่องใช้บริการต้องการจะใช้การติดต่อแบบ .rhosts หรือ /etc/hosts.equiv จะต้องทำการติดต่อจากพอร์ตที่ถูกสงวนไว้ (หมายเลขพอร์ตที่ต่ำกว่าเบอร์ 1024)

3.3 รายละเอียดขั้นตอนการติดต่อ

3.3.1 ช่วงตรวจสอบเวอร์ชัน (Protocol Version Identification)

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

ส่วนผู้ใช้บริการเมื่อได้รับข้อความมาก็จะทำการแปล ถ้าไม่รองรับเวอร์ชันของโพรโตคอลก็จะทำการปิดการติดต่อ แต่ถ้ารองรับเครื่องผู้ใช้บริการก็จะส่งข้อความที่เป็นข้อมูลของตัวเองลักษณะเดียวกับที่เครื่องผู้ให้บริการส่งมากลับไป เมื่อเครื่องผู้ให้บริการได้รับก็จะตีความหมายว่าเป็นเวอร์ชันของโพรโตคอลที่รองรับหรือไม่ ถ้าใช่ก็จะส่งข้อความแรกกลับไป ซึ่งข้อความจะอยู่ในรูปของ binary packet protocol



รูปที่ 3-5 แสดงการตรวจสอบเวอร์ชัน Identification

3.3.2 ช่วยการแลกเปลี่ยนคีย์ (Key Exchange)

หลังจากที่มีการตรวจสอบเวอร์ชันว่ารองรับกันได้แล้ว ก็จะเข้าสู่การแลกเปลี่ยนคีย์ เนื่องจากการเข้ารหัสของข้อมูลต่างๆ ที่ใช้ติดต่อกันทั้งสองฝ่ายจะต้องมีคีย์เดียวกันในการเข้ารหัสและถอดรหัสของข้อมูลนั้นๆ ซึ่งจะต้องมีวิธีการทำให้ทั้งคู่มีคีย์เดียวกัน โดยที่ผู้อื่นไม่ทราบว่าคีย์นี้ คืออะไร ซึ่งมีขั้นตอนคือ

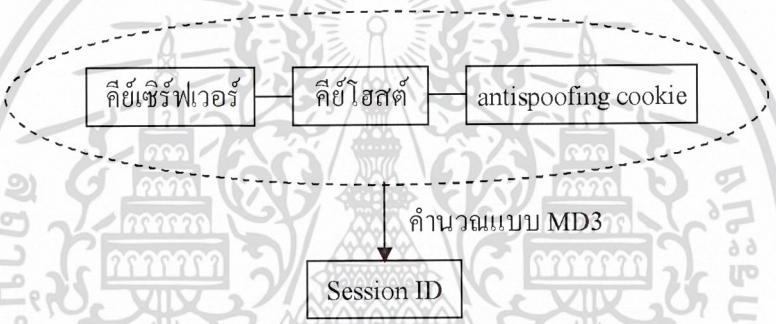
หลังจากที่เซิร์ฟเวอร์ได้รับ Identification String จากไคลเอ็นต์และตรวจสอบว่ารองรับแล้ว เซิร์ฟเวอร์จะส่งข้อความในรูปแบบ Binary Packet Protocol ที่เป็นชนิดของ SSH_SMSG_PUBLIC_KEY ซึ่งจะประกอบด้วย

- **ค่าที่สุ่มขึ้นมาขนาด 64 บิต (คุ้กกี้)** เพื่อให้ไคลเอ็นต์ส่งข้อมูลนี้กลับมา ทำให้ยากต่อการทำการปลอมแปลง IP
- **โฮสต์คีย์ (host key)** เป็นคีย์สาธารณะของเซิร์ฟเวอร์ มีขนาด 1024 บิต
- **คีย์เซิร์ฟเวอร์ (server key)** เป็นคีย์สาธารณะของเซิร์ฟเวอร์ที่มีการสร้างใหม่ทุกๆชั่วโมง มีขนาด 768 บิต

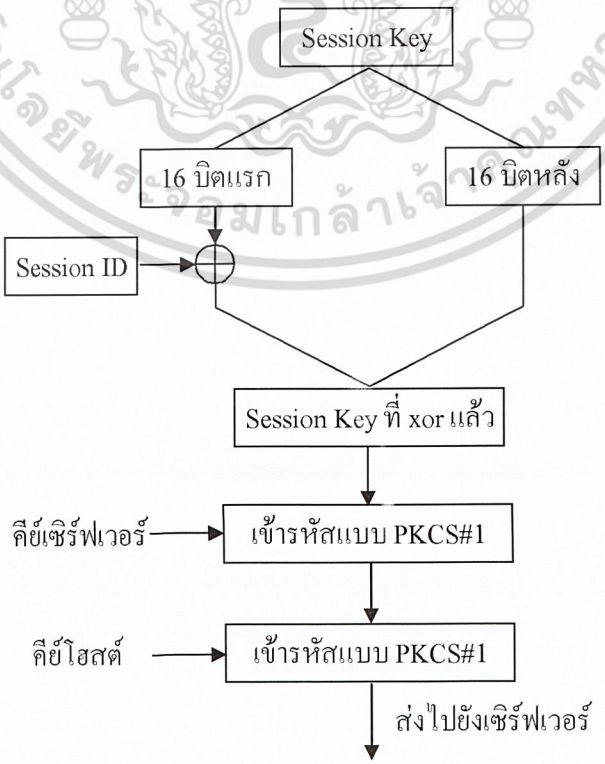
- **การเข้ารหัสที่รองรับ (supported ciphers)** เป็นชนิดของการเข้ารหัสที่เซิร์ฟเวอร์รองรับ
- **การพิสูจน์สิทธิที่รองรับ (supported authentication method)** เป็นวิธีของการพิสูจน์ตนที่เซิร์ฟเวอร์รองรับ

เมื่อไคลเอ็นต์ได้รับข้อความ SSH_SMSG_PUBLIC_KEY ไคลเอ็นต์จะสร้างคีย์ (เรียกว่า คีย์เซสชัน) ที่ใช้สำหรับเข้ารหัสแพ็กเก็ตต่างๆขึ้นมาโดยการสุ่มค่าขึ้น (คีย์มีขนาด 32 ไบต์) เพื่อจะทำการส่งไปให้เซิร์ฟเวอร์ โดยก่อนที่จะทำการส่งจะนำเอาคีย์เซสชัน 16 บิตแรก มาเอ็กซ์คลูซีฟออร์กับไอดีของเซสชัน (มีขนาด 16 ไบต์) แล้วนำมาต่อรวมกับ 16 บิตหลังของคีย์เซสชัน จากนั้นจึงทำการเข้ารหัสแบบ PKCS#1 (ซึ่งเป็นการเข้ารหัสแบบ RSA) 2 ครั้ง ด้วยคีย์เซิร์ฟเวอร์และคีย์โฮสต์ โดยใช้กุญแจที่สั้นกว่าก่อน (นั่นคือคีย์เซิร์ฟเวอร์)

สำหรับไอดีของเซสชันเป็นค่าที่ทั้งสองฝั่งคำนวณหาได้ โดยการนำเอาคีย์โฮสต์ต่อด้วยคีย์เซิร์ฟเวอร์และต่อด้วยคุกกี้ แล้วนำมาคำนวณด้วยอัลกอริทึมแบบ MD3 ซึ่งทั้งสองฝั่งจะได้ค่านี้เช่นเดียวกัน



รูปที่ 3-6 แสดงการคำนวณหาไอดีของเซสชัน



รูปที่ 3-7 แสดงการเข้ารหัสคีย์เซสชันก่อนส่งให้เซิร์ฟเวอร์

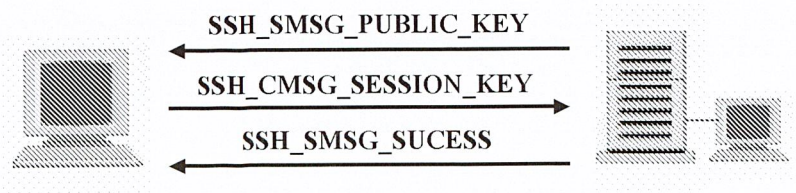
สำหรับการเข้ารหัสแบบ PKCS#1 นั้นใช้มาตรฐานของ RSA มีลักษณะของการเข้ารหัสเหมือน RSA โดยข้อมูลที่จะทำการเข้ารหัสนั้น ให้ไบต์แรกมีค่าเป็น 0 ไบต์ถัดมามีค่าเป็น 2 (ตามมาตรฐาน) แล้วตามด้วยค่าที่สุ่มขึ้นและไม่เท่ากับ 0 ในตำแหน่งถัดมาที่ไม่ได้ใช้ ตามด้วยไบต์ 0 และตามด้วยข้อมูลที่ต้องการจะถูกเข้ารหัส แล้วนำข้อมูลนี้มาทำการเข้ารหัสแบบ RSA ($C = p^e \bmod n$ โดย C คือผลการเข้ารหัส, p คือ ข้อมูลที่จะทำการเข้ารหัส, e คือ เอ็กซ์โปเนนท์ของคีย์สาธารณะ และ n คือ โมดูลัสของคีย์สาธารณะ)

รูปที่ 3-8 แสดงตัวอย่างของข้อมูลที่จะทำการเข้ารหัส PKCS#1

หลังจากที่ได้ข้อมูลที่เป็นคีย์ (ที่ทำการเข้ารหัสข้างต้น) ก็จะส่งแพ็กเก็ตชนิด SSH_CMSG_SESSION_KEY โดยมีข้อมูลต่างๆดังต่อไปนี้ ตามลำดับ

- ชนิดของการเข้ารหัส
- คุ้กกี้ที่ได้จากเซิร์ฟเวอร์
- เซสชันคีย์ที่ถูกเข้ารหัสข้างต้น
- โพรโตคอลเวอร์ชันที่ใช้

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเก็ต SSH_CMSG_SESSION_KEY ก็จะถอดรหัสและคำนวณหาเซสชันคีย์ออกมา หากลักษณะของข้อมูลถูกต้องก็จะส่งแพ็กเก็ตกลับไปว่าเรียบร้อยแล้ว โดยมีชนิดว่า SSH_SMSG_SUCCESS ซึ่งข้อมูลนี้จะถูกทำการเข้ารหัสด้วยวิธีการตามที่ไคลเอ็นต์เลือกมา



รูปที่ 3-9 ขั้นตอนการส่งของแพ็กเก็ตต่างๆ ในช่วงการแลกเปลี่ยนคีย์

3.4 ช่วงตรวจสอบชื่อและพิสูจน์ตน (Declare User Name and Authentication Phase)

ไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_CMSG_USER ไปยังเซิร์ฟเวอร์ ซึ่งจะมีชื่อของผู้ใช้ในแพ็กเก็ตนี้ เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบว่ามีผู้ใช้คนนั้นในรายการหรือไม่ และตรวจสอบการตรวจสอบสิทธิ์ที่ต้องใช้ แล้วจะตอบกลับด้วยแพ็กเก็ตชนิด SSH_SMSG_SUCCESS ถ้าผู้ใช้คนนี้ถูกกำหนดไว้ว่าไม่ต้องการให้มีการพิสูจน์ตน (ไม่ต้องใช้รหัสผ่าน) หรือ SSH_SMSG_FAILURE ถ้าผู้ใช้จำเป็นต้องมีการพิสูจน์ตน

สำหรับในกรณีที่ไม่มีชื่อของผู้ใช้คนนี้ เซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_FAILURE มาเช่นกัน ทำให้ผู้ใช้ไม่ทราบว่าผู้ใช้คนนี้มีอยู่หรือเปล่า

เมื่อไคลเอ็นต์ได้รับแพ็กเก็ต SSH_SMSG_FAILURE ไคลเอ็นต์จะทำการส่งวิธีการในการพิสูจน์ตนให้กับเซิร์ฟเวอร์ ซึ่งวิธีการพิสูจน์ตนที่รองรับ มีวิธีต่างๆดังนี้

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

ถ้าเซิร์ฟเวอร์ได้รับวิธีการพิสูจน์ตนจากไคลเอ็นต์ หากรองรับวิธีการก็จะส่งแพ็กเก็ตชนิด SSH_SMSG_SUCCESSS แต่หากไม่รองรับก็จะส่งแพ็กเก็ตชนิด SSH_SMSG_FAILURE

เวลาสำหรับการพิสูจน์ตนจะถูกกำหนดไว้ไม่ให้เกิน 3 นาที (ถ้าหากเกิน เซิร์ฟเวอร์ก็จะทำการยกเลิกการติดต่อ) สำหรับวิธีการตรวจสอบสิทธิ์วิธีต่างๆมีรายละเอียดดังนี้

3.4.1 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv

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

การวิธีการพิสูจน์ตนด้วยระบบที่เชื่อถือฝั่งรีโมต (trusts the remote host) สามารถที่จะถูกอ่านของข้อมูลได้โดยวิธีการปลอมแปลงเลขไอพี แต่จะถูกป้องกันไว้จากโพรโตคอล (เพราะทุกๆแพ็กเก็ตจะถูกเข้ารหัสอยู่)

3.4.2 ตรวจสอบสิทธิ์ด้วยวิธี RSA

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RSA ตามด้วยโมดูลัสของคีย์สาธารณะของไคลเอ็นต์ (ต่างจากวิธี .rhosts + RSA ที่ส่งค่าคีย์สาธารณะทั้งหมดไป แต่วิธีนี้ส่งเฉพาะค่า n ที่เป็นโมดูลัสของคีย์สาธารณะ)

ถ้าเซิร์ฟเวอร์ไม่พบคีย์การพิสูจน์ตนของไคลเอ็นต์นี้ในตัวมัน (ค่าคีย์สาธารณะทั้งหมดของไคลเอ็นต์) เซิร์ฟเวอร์จะตอบกลับทันทีด้วย SSH_SMSG_FAILURE ในทางตรงกันข้าม ถ้าเซิร์ฟเวอร์พบ มันจะสร้าง challenge ซึ่งทำการเข้ารหัสด้วยคีย์สาธารณะของไคลเอ็นต์

เมื่อไคลเอ็นต์ได้รับ จะทำการถอดรหัสโดยคีย์ส่วนตัว แล้วนำมาต่อกับไอดีของเซสชันที่มีอยู่ ทำการคำนวณแบบ MD3 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge +เซสชันไอดี => ทำ MD3) ถ้าถูกต้องก็จะส่ง SSH_SMSG_SUCCESS แต่ในทางตรงกันข้ามก็จะส่ง SSH_SMSG_FAILURE และปฏิเสธการพิสูจน์ตนนี้

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

เป็นวิธีที่เพิ่มจากวิธีการตรวจสอบสิทธ์จากไฟล์ .rhosts หรือ /etc/host.equiv โดยมีการพิสูจน์ตนแบบ RSA ด้วย โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_RHOSTS_RSA ตามด้วยชื่อของผู้ใช้และกุญแจสาธารณะของไคลเอ็นต์

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

ถ้าเซิร์ฟเวอร์รู้โฮสต์คีย์ของเครื่องไคลเอ็นต์ มันจะทำการตรวจสอบจากโฮสต์คีย์ที่ได้มาว่าตรงกันหรือไม่ ถ้าไม่ ก็จะส่ง SSH_SMSG_FAILURE เพื่อยกเลิกการติดต่อเช่นกัน

ถ้าหากถูกต้องเซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_AUTH_RSA_CHALLENGE ที่บรรจุด้วย challenge ที่ถูกเข้ารหัสสำหรับไคลเอ็นต์ โดย challenge จะมีขนาดเป็น 32 บิตที่ถูกสุ่มขึ้นมาแล้วทำการจัดข้อความและเข้ารหัสด้วยกุญแจสาธารณะของไคลเอ็นต์ (วิธีการเช่นเดียวกับที่เข้ารหัสเซสชันคีย์)

ไคลเอ็นต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับเซสชันไอดีที่มีอยู่ มาทำการคำนวณแบบ MD3 ได้ผลลัพธ์ออกมามีขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge +เซสชันไอดี => ทำ MD3) ถ้าถูกต้องก็จะส่ง SSH_SMSG_SUCCESS ตรงกันข้ามก็จะส่ง SSH_SMSG_ FAILURE และปฏิเสธการพิสูจน์ตนนี้

3.4.4 ตรวจสอบสิทธิ์จากรหัสผ่าน

โดยไคลเอ็นต์จะส่ง SSH_CMSG_AUTH_PASSWORD ตามด้วยข้อความที่เป็นรหัสผ่าน (ข้อความนี้จะยังไม่มีการทำอะไร แต่จะถูกทำการเข้ารหัสด้วยกลไกของแพ็กเก็ตโดยอัตโนมัติ)

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบรหัสผ่าน ถ้าการพิสูจน์ตนถูกต้อง เซิร์ฟเวอร์จะส่ง SSH_SMSG_SUCCESS แต่ถ้าผิดพลาดเซิร์ฟเวอร์จะส่ง SSH_SMSG_FAILURE

3.5 ช่วงการเตรียมการ (Preparatory Operation)

หลังจากที่ทำการพิสูจน์ตนแล้ว เซิร์ฟเวอร์จะรอคำร้องขอจากไคลเอ็นต์และทำการจัดการกับคำสั่งที่เข้ามา เซิร์ฟเวอร์จะตอบกลับด้วย SSH_SMSG_SUCCESS เมื่อคำร้องขอที่เข้าถูกจัดการเสร็จสิ้น แต่ในกรณีตรงกันข้าม เมื่อคำร้องขอที่เข้ามาไม่สามารถจัดการได้หรือจัดการไม่สำเร็จก็จะส่ง SSH_SMSG_ FAILURE ซึ่งข้อความช่วงนี้เป็นการเตรียมการสำหรับในช่วงต่อไป

3.6 เซสชันแบบอินเตอร์แอ็กทีฟและการแลกเปลี่ยนข้อมูล (Interactive Session and Exchange of Data)

ในเซสชันแบบอินเตอร์แอ็กทีฟ ข้อมูลทุกตัวจะถูกเขียนโดยเชลล์หรือคำสั่งที่ทำงานอยู่บนเครื่องเซิร์ฟเวอร์เพื่อส่งออกไปยังส่วนแสดงผลหรือส่วนแสดงข้อผิดพลาดบนเครื่องไคลเอ็นต์ ส่วนอินพุตจะมาจากส่วนรับข้อมูลบนเครื่องไคลเอ็นต์ ซึ่งจะถูกส่งไปยังโปรแกรมบนเครื่องเซิร์ฟเวอร์

การแลกเปลี่ยนข้อมูลทั้งหมดจะเป็นอะซิงโครนัส (asynchronous) โดยการส่งสามารถเกิดขึ้นได้ทุกเวลาและไม่ต้องมีการตอบสนอง (ปกติแล้ว ทีซีพี/ไอพีจัดการสร้างความน่าเชื่อถือได้อยู่แล้ว และโพรโตคอลแพ็กเก็ตจะทำการป้องกันการเข้ามายุ่งหรือทำการปลอมแปลงไอพี)

เมื่อไคลเอ็นต์ได้รับ EOF จากส่วนรับข้อมูล มันจะส่ง SSH_CMSG_EOF การแลกเปลี่ยนข้อมูลและโหมดอินเตอร์แอ็กทีฟจะสิ้นสุดลงเมื่อเซิร์ฟเวอร์ส่ง SSH_SMSG_EXITSTATUS เพื่อที่จะแสดงว่าการติดต่อกับตัวไคลเอ็นต์ได้สิ้นสุดลง ส่วนไคลเอ็นต์หรือเซิร์ฟเวอร์จะสามารถยกเลิกการติดต่อเมื่อใดก็ได้ โดยการส่งข้อความ SSH_MSG_DISCONNECT หรือทำการปิดการเชื่อมต่อ (close the connection)

3.7 การยกเลิกการติดต่อ (Termination of the Connection)

โดยปกติการยกเลิกการติดต่อจะเริ่มโดยเซิร์ฟเวอร์ซึ่งจะทำการส่ง SSH_SMSG_EXITSTATUS หลังจากที่โปรแกรมจบลง ตัวไคลเอ็นต์จะทำการตอบสนองกับข้อความดังกล่าว โดยการส่ง SSH_ CMSG_EXIT_CONFIRMATION และทำการปิดซ็อกเก็ตของตัวเองลง ส่วนเซิร์ฟเวอร์เมื่อได้รับข้อความจึงค่อยทำการปิดซ็อกเก็ตลง เป้าหมายสำคัญสำหรับสำหรับการยืนยันในการยกเลิกคือ ในบางระบบอาจจะสูญเสียข้อมูลที่ส่งไปก่อนหน้านี้เมื่อซ็อกเก็ตถูกปิด และการยกเลิกในฝั่งไคลเอ็นต์ก่อนทำให้เกิด

TCP/IP TIME_WAIT [RFC 0793] ซึ่งทำให้เซิร์ฟเวอร์รอรับคำตอบจากฝั่งไคลเอ็นต์ที่ไม่ได้ใช้ ทำให้เซิร์ฟเวอร์ต้องสูญเสียรีซอร์ส

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

3.8 การใช้งานการฟอร์เวิร์ดพอร์ตของโพรโตคอลเอสเอสเอช 1

การฟอร์เวิร์ดพอร์ตความสามารถที่เป็นประโยชน์มากอีกอย่างของโพรโตคอลเอสเอสเอช 1 ทำให้เราสามารถใช้โปรแกรมประยุกต์อื่นๆ ที่ไม่มีความปลอดภัย คือไม่มีการเข้ารหัสข้อมูล เป็นการสื่อสารที่มีความปลอดภัยด้วยการเข้ารหัสของโพรโตคอลเอสเอสเอช 1 แบ่งออกเป็น 2 ชนิดคือ

- การฟอร์เวิร์ดเอ็กซ์ (X Forwarding)
- การฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี (TCP/IP Port Forwarding)

3.8.1 การฟอร์เวิร์ดเอ็กซ์ (X Forwarding)

สำหรับในระบบยูนิกซ์นั้น ผู้ใช้สามารถใช้การติดต่อแบบกราฟิกได้โดยการเรียกใช้เอ็กซ์วินโดว์ (X window) ซึ่งการเรียกใช้เอ็กซ์วินโดว์จะใช้โพรโตคอลเอ็กซ์ 11 โพรโตคอลเอสเอสเอช 1 นั้นสามารถช่วยเข้ารหัสข้อมูลที่ถูกรับส่งโดยโพรโตคอลเอ็กซ์ 11 นี้ได้โดยใช้หลักการเดียวกับการฟอร์เวิร์ดพอร์ตทีซีพี/ไอพีซึ่งจะอธิบายต่อไป

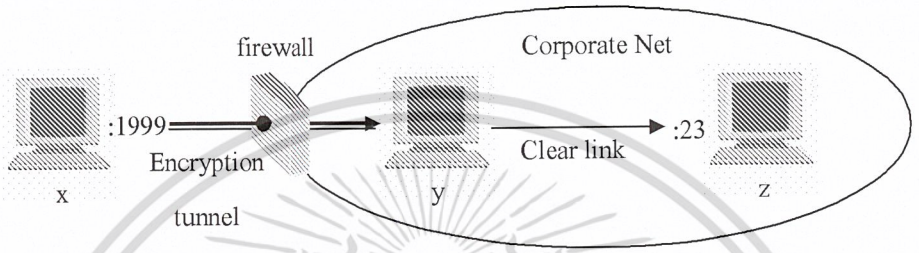
3.8.2 การฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี (TCP/IP Port Forwarding)

ปกติแล้วของโพรโตคอลที่ทำงานอยู่บนโพรโตคอลทีซีพี/ไอพี เช่น เอชทีทีพี, เอฟทีพี, เทลเน็ต, เอสเอ็มทีพี นั้นจะถูกแบ่งแยกโดยหมายเลขพอร์ต โพรโตคอลเอสเอสเอช 1 ก็เช่นกัน โดยโพรโตคอลเอสเอสเอช 1 ทำงานที่พอร์ตหมายเลข 22 เมื่อเราเรียกใช้งานโพรโตคอลเอสเอสเอช 1 แล้ว การส่งข้อมูลระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะถูกเข้ารหัสเฉพาะข้อมูลของโพรโตคอลเอสเอสเอช 1 เท่านั้น โดยข้อมูลของโพรโตคอลอื่นๆ นั้นไม่มีการเข้ารหัสเลย โพรโตคอลเอสเอสเอช 1 สามารถช่วยให้โพรโตคอลต่างๆ ที่ทำงานอยู่บนโพรโตคอลทีซีพี/ไอพีมีการเข้ารหัสข้อมูลได้โดยการเรียกใช้บริการฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี ซึ่งแบ่งออกได้เป็น 2 ลักษณะ คือ การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ หรือ มีการสร้างพอร์ตที่ฝั่งของเซิร์ฟเวอร์โพรโตคอลเอสเอสเอช 1

- การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ (Local) สามารถอธิบายง่ายๆได้ด้วยตัวอย่างดังรูปที่ 3-10 เป็นการฟอร์เวิร์ดพอร์ตจากโฮสต์ฝั่งโลคอลไปยังเครื่องรีโมต คือ พิจารณาโฮสต์ 3 แห่ง คือ x y และ z โฮสต์ x นั้นเป็นเครื่องไคลเอ็นต์และสร้างการเชื่อมต่อกับโฮสต์ y ซึ่งเป็นเครื่องเซิร์ฟเวอร์ผ่านโพรโตคอลเอสเอสเอช 1 พอร์ตที่ 1999 บนเครื่อง x ถูกสร้างส่งต่อไปยังเครื่องอีกเครื่องนั้นคือ โฮสต์ z ที่พอร์ต

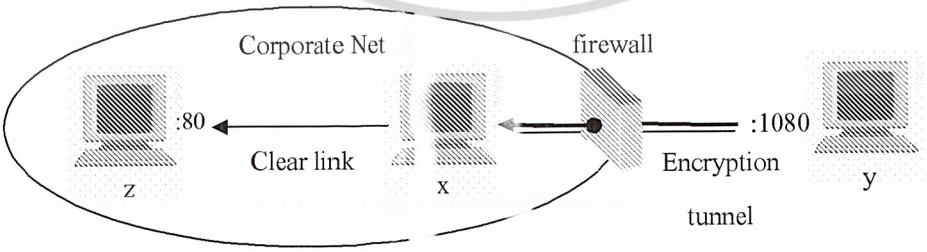
23 พอร์ต 1999 นี้จะส่งผ่านช่องทางที่มีการเข้ารหัสไปยังเครื่อง y ซึ่งจะส่งต่อไปที่เครื่อง z ที่พอร์ต 23 โดยที่ไม่มีการเข้ารหัสในการเชื่อมต่อส่วนนี้

ด้วยเหตุนี้เอง ถ้าใช้คำสั่ง telnet localhost 1999 จะมีผลเหมือนกับการเชื่อมต่อเพื่อสร้างการติดต่อกับ z โดยผ่านช่องทางที่มีความปลอดภัยของเครื่อง y



รูปที่ 3-10 แสดงการทำงานของการฟอร์เวิร์ดพอร์ตแบบโลคอล

- การสร้างพอร์ตที่ฝั่งเซิร์ฟเวอร์โพรโตคอลเอสเอสเอช 1 (remote) เป็นการฟอร์เวิร์ดพอร์ตของโฮสต์รีโมตไปยังเครื่องโลคอล คือ จะฟอร์เวิร์ดพอร์ตของเครื่องรีโมต ซึ่งคือ y พอร์ตที่ 1080 ไปยังโฮสต์ z พอร์ต 80 บนช่องทางที่มีการเข้ารหัสของ x ดังรูปที่ 3-11 ซึ่งสามารถถูกมองเป็น pseudo proxy capability หรือ tunneling IP และมีผลกระทบในแง่ของจุดบกพร่องทางด้านความปลอดภัยในไฟล์วอลล์ ซึ่งจะอนุญาตให้โพรโตคอลเอสเอสเอช 1 ผ่านไปได้ ดังนั้นไฟล์วอลล์หรืออุปกรณ์ที่ทำหน้าที่ใกล้เคียงกัน จะคิดว่ามันกำลังอนุญาตให้การติดต่อของโพรโตคอลเอสเอสเอช 1 ผ่านเข้าไป แต่จะมีผลที่จริงแล้วเหมือนกับการอนุญาตให้มีการติดต่อกันอย่างอิสระระหว่างระบบภายในกับระบบภายนอก



รูปที่ 3-11 แสดงการทำงานของการส่งต่อพอร์ตแบบรีโมต

3.9 ข้อเสียของ SSH-1

- การทำงานจะทำงานอยู่เพียงโพรโตคอลเดียว ทำให้การทำงานไม่มีความยืดหยุ่นในการทำงาน
- ใช้ CRC – 32 ในการตรวจสอบความถูกต้องทำให้ง่ายต่อการถูกโจมตี
- สามารถมีหนึ่งเซสชั่นเท่านั้นต่อหนึ่งการเชื่อมต่อ
- ไม่สามารถเลือกอัลกอริทึมการบีบอัดข้อมูล , MAC ได้โดยเลือกได้เฉพาะอัลกอริทึมการเข้ารหัสลับเท่านั้น
- ไม่สามารถที่จะพัฒนาอัลกอริทึมขึ้นมาเองได้เนื่องจากไม่สนับสนุนการเรียกชื่ออัลกอริทึมด้วยสตริง

บทที่ 4

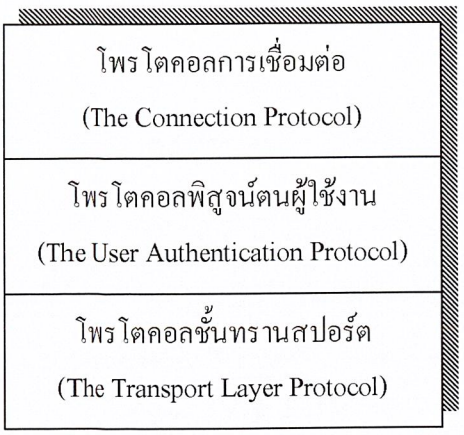
โพรโตคอลเอสเอสเอช 2

4.1 แนะนำ

โพรโตคอลเอสเอสเอช 2 เป็นโพรโตคอลสำหรับการล็อกอินระยะไกลแบบปลอดภัย และการใช้บริการอื่นๆผ่านเครือข่ายแบบปลอดภัย บนเครือข่ายที่ไม่ปลอดภัย ประกอบด้วยสามคอมโพเนนท์หลัก คือ

- **โพรโตคอลชั้นทรานสปอร์ต (The Transport Layer Protocol)** ทำให้มีการพิสูจน์เซิร์ฟเวอร์ (server authentication), การสร้างความมั่นใจในความปลอดภัย (confidentiality) และความสมบูรณ์ (integrity) และสามารถเลือกให้มีการบีบอัด (compression) ได้ด้วย โดยปกติชั้น ทรานสปอร์ตจะทำงานบนการเชื่อมต่อแบบทีซีพี/ไอพี (TCP/IP Connection) แต่ก็อาจถูกนำไปใช้ทำงานอยู่บนสตรีมข้อมูลที่ไว้ใจได้ (reliable data stream) อื่นๆอีกด้วย
- **โพรโตคอลพิสูจน์ตนผู้ใช้ (The User Authentication Protocol)** ทำหน้าที่พิสูจน์ตนของผู้ใช้งานฝั่งไคลเอ็นต์ (client-side) ให้แก่เซิร์ฟเวอร์ โพรโตคอลนี้ทำงานบนโพรโตคอลชั้นทรานสปอร์ต
- **โพรโตคอลการเชื่อมต่อ (The Connection Protocol)** ทำหน้าที่มัลติเพล็กซ์ (multiplex) ทันเนลที่เข้ารหัส (encrypted tunnel) เป็นหลายๆแชนเนล (logical channel) โพรโตคอลนี้จะทำงานอยู่บนโพรโตคอลพิสูจน์ตนผู้ใช้งาน

โดยการจัดวางตัวของชั้นโพรโตคอลหลักของโพรโตคอลเอสเอสเอช 2 เป็นดังรูปที่ 4-1



รูปที่ 4-1 การจัดวางตัวของโพรโตคอลหลักภายในโพรโตคอลเอสเอสเอช 2

ไคลเอ็นต์จะส่งคำร้องขอบริการ (service request) เมื่อการเชื่อมต่อชั้นทรานสปอร์ตที่ปลอดภัยได้ถูกก่อตั้งขึ้นแล้ว คำร้องขอบริการครั้งที่สองจะถูกส่งหลังจากการพิสูจน์ตนของผู้ใช้ได้ทำสำเร็จลง

โพรโตคอลการเชื่อมต่อให้แชนเนลที่สามารถนำเอาไปใช้ได้หลากหลายจุดประสงค์ วิธีการมาตรฐานถูกใช้ในการตั้งค่าเซสชันของเชลล์แบบอินเทอร์แอกทีฟที่ปลอดภัย และสำหรับการฟอร์เวิร์ด (forwarding) หรือการทันเนล พอร์ตทีซีพี/ไอพีและการเชื่อมต่อ X11

4.2 สถาปัตยกรรม

4.2.1 คีย์โฮสต์ (Host Keys)

แต่ละโฮสต์เซิร์ฟเวอร์ควรมีคีย์โฮสต์ 1 คีย์ โฮสต์อาจจะมีคีย์โฮสต์ได้หลายคีย์สำหรับอัลกอริทึมต่างกัน หลายๆโฮสต์อาจใช้คีย์โฮสต์เดียวกันร่วมกัน โฮสต์ต้องมีคีย์อย่างน้อย 1 คีย์สำหรับอัลกอริทึมคีย์สาธารณะซึ่งบังคับต้องมี (ในขณะนี้ใช้ DSS [FIPS-186])

คีย์โฮสต์ของเซิร์ฟเวอร์ถูกใช้ในระหว่างการแลกเปลี่ยนคีย์ เพื่อใช้ในการตรวจสอบว่าไคลเอ็นต์ได้ติดต่อกับเซิร์ฟเวอร์ที่ถูกต้องจริง เพื่อให้ใช้วิธีการนี้ได้ ไคลเอ็นต์ต้องรู้คีย์โฮสต์สาธารณะของเซิร์ฟเวอร์ก่อน

มีโมเดลการไว้วางใจ (trust modes) 2 โมเดลที่แตกต่างกันซึ่งสามารถนำมาใช้

- ไคลเอ็นต์มีฐานข้อมูลโลคอล ซึ่งเก็บแต่ละชื่อโฮสต์ (ถูกพิมพ์โดยผู้ใช้) กับคีย์โฮสต์สาธารณะที่สอดคล้องกันไว้ วิธีการนี้ไม่ต้องการโครงสร้างการบริหารแบบศูนย์กลางและการประสานงานกับกลุ่ม (party) ที่สาม ข้อเสียคือฐานข้อมูลของ ชื่อ-คีย์ กลับกลายเป็นภาระในการบำรุงรักษา
- ข้อมูล ชื่อ-คีย์ ของโฮสต์ถูกรับรองโดย CA (certification authority) ที่ไว้วางใจ ไคลเอ็นต์รู้เพียงคีย์รูต (root key) ของ CA ก็สามารถตรวจสอบความถูกต้องของทุกคีย์โฮสต์ที่รับรองโดย CA ที่ยอมรับ

ทางเลือกที่สองช่วยบรรเทาปัญหาการบำรุงรักษา เนื่องจากตามอุดมคติ คีย์ของ CA คีย์เพียงเดียวที่จำเป็นต้องถูกเก็บอย่างปลอดภัยลงในไคลเอ็นต์ แต่คีย์โฮสต์แต่ละคีย์ต้องถูกรับรองโดย CA ก่อนที่จะสามารถทำการพิสูจน์ตนได้

โพรโตคอลมีทางเลือกที่จะไม่ตรวจสอบความสอดคล้องกันของ ชื่อเซิร์ฟเวอร์-คีย์โฮสต์ ในขณะทำการเชื่อมต่อครั้งแรกไปยังโฮสต์ ทำให้การสื่อสารไม่มีการสื่อสารกันก่อนของคีย์โฮสต์หรือใบรับรอง การเชื่อมต่อยังคงป้องกันการดักฟัง (passive listening) แต่อย่างไรก็ตาม กลับกลายเป็นอ่อนแอต่อการโจมตีโดยบุคคลกลาง (active man-in-the-middle attacks) การอิมพลีเมนต์ไม่ควรให้การเชื่อมต่อรูปแบบนี้เป็นแบบดีฟอลต์ แต่อย่างไรก็ดี ทางเลือกนี้ยังคงให้ความปลอดภัยมากกว่าที่ได้รับจากใช้วิธีการแบบเดิม (เช่น telnet [RFC-854] และ rlogin [RFC-1282])

การอิมพลีเมนต์อาจให้วิธีการเพิ่มเติมสำหรับตรวจสอบความถูกต้องของคีย์โฮสต์ เช่น ฟิงเกอร์พริ้นท์แบบเลขฐานสิบหก ที่ได้จากอัลกอริทึมแฮช SHA-1 ของคีย์สาธารณะ ฟิงเกอร์พริ้นท์ถูกตรวจสอบได้ง่ายโดยใช้โทรศัพท์หรือแชนเนลการสื่อสารภายนอก

ทุกการอิมพลีเมนต์ควรให้ทางเลือกที่ไม่ทำการยอมรับคีย์โฮสต์ที่ไม่สามารถตรวจสอบได้

4.2.2 ความสามารถในการขยาย (Extensibility)

เชื่อว่าโพรโตคอลจะมีวิวัฒนาการตลอดเวลา และบางองค์กรมีความต้องการที่ใช้การเข้ารหัส, การพิสูจน์ตน และหรือ วิธีการแลกเปลี่ยนคีย์ ของตนเอง ในทางกลับกัน การไม่มีการลงทะเบียนศูนย์กลางนำไปสู่ความขัดแย้งในไอเดนติฟายเออร์ของวิธีการ ทำให้ยากต่อการทำงานร่วมระหว่างระบบ (interoperability)

เราเลือกที่จะบ่งชี้อัลกอริทึม, วิธีการ, รูปแบบ และโพรโตคอลขยายเพิ่ม ด้วยชื่อในรูปแบบข้อความซึ่งอยู่ในรูปแบบที่กำหนดไว้ ชื่อ DNS ถูกใช้ในการตั้งโลคอลเนมสเปซ (local namespace) ทำให้การทดลองหรือการขยายที่เป็นหมวดหมู่สามารถกำหนดได้ โดยไม่ต้องกลัวว่าจะเกิดความขัดแย้งกับการอิมพลีเมนต์อื่น

เป้าหมายหนึ่งของการออกแบบคือพยายามให้โพรโตคอลพื้นฐานง่ายที่สุดเท่าที่จะเป็นไปได้ และต้องการอัลกอริทึมน้อยที่สุดเท่าที่จะเป็นไปได้ อย่างไรก็ตาม ทุกการอิมพลีเมนต์ต้องรองรับกลุ่มอัลกอริทึมเล็กที่สุดที่ทำให้มีความสามารถทำงานร่วมกันระหว่างระบบได้อย่างมั่นใจ (ไม่ได้บอกว่านโยบาย (policy) โลคอลบนทุกโฮสต์จำเป็นต้องใช้อัลกอริทึมเหล่านี้) อัลกอริทึมขาดไม่ได้ (mandatory) ถูกกำหนดในเอกสารของโพรโตคอล

อัลกอริทึมเพิ่มเติม, วิธีการ, รูปแบบ และโพรโตคอลขยายเพิ่ม สามารถถูกกำหนดในฉบับร่าง (draft) แยกกันไป ข้อมูลเพิ่มเติมดูในหัวข้อการตั้งชื่ออัลกอริทึม

4.2.3 การกำหนดนโยบาย (Policy Issues)

โพรโตคอลมีการเจรจาการเข้ารหัส, การตรวจสอบความสมบูรณ์, การแลกเปลี่ยนคีย์, การบีบอัด และอัลกอริทึมและรูปแบบคีย์สาธารณะ อัลกอริทึมการเข้ารหัส, อัลกอริทึมการตรวจสอบความสมบูรณ์, อัลกอริทึมคีย์สาธารณะ และอัลกอริทึมการบีบอัด สามารถแตกต่างกันสำหรับแต่ละทิศทาง

การกำหนดนโยบายควรกำหนดลงในกลไกการปรับแต่งค่าของแต่ละการอิมพลีเมนต์ดังต่อไปนี้

- อัลกอริทึมการเข้ารหัส, อัลกอริทึมการตรวจสอบความสมบูรณ์ และอัลกอริทึมการบีบอัด จะแยกกันสำหรับแต่ละทิศทาง นโยบายต้องกำหนดอัลกอริทึมเสนอ (prefered algorithm) เช่น ลิสต์อัลกอริทึมแรกในแต่ละหมวด
- อัลกอริทึมคีย์สาธารณะ และวิธีการแลกเปลี่ยนคีย์ ถูกใช้ในการพิสูจน์โฮสต์ การมีอยู่ของคีย์โฮสต์ที่ไว้วางใจได้สำหรับอัลกอริทึมคีย์สาธารณะที่แตกต่างกัน จะมีผลต่อตัวเลือกนี้
- วิธีการพิสูจน์ตนที่เซิร์ฟเวอร์ต้องการสำหรับแต่ละผู้ใช้ นโยบายของเซิร์ฟเวอร์อาจต้องการพิสูจน์ตนหลายครั้ง สำหรับบางผู้ใช้หรือกับทุกผู้ใช้ อัลกอริทึมที่ต้องการขึ้นอยู่กับที่ ๆ ผู้ใช้กำลังพยายามล็อกอินมาจากที่นั้น
- การดำเนินการที่ผู้ใช้ได้รับอนุญาตให้กระทำโดยใช้โพรโตคอลการเชื่อมต่อ บางประเด็นมีความเกี่ยวข้องกับความปลอดภัย ตัวอย่างเช่น นโยบายไม่ควรอนุญาตให้เซิร์ฟเวอร์สตาร์ตเซสชันหรือรันคอมมานด์บนเครื่องไคลเอ็นต์ และต้องไม่อนุญาตให้มีการเชื่อมต่อไปยังเอเจนต์พิสูจน์ตน (authentication agent) เว้นแต่การฟอร์เวิร์ดที่การเชื่อมต่อถูกร้องขอมา ประเด็นอื่น ๆ เช่น พอร์ตทีซีพี/ไอพีใด ๆ ที่สามารถถูกฟอร์เวิร์ด และโดยใครได้บ้าง เป็นประเด็นของนโยบายบนโลคอล หลายประเด็นนี้เกี่ยวข้องกับการทราเวิร์ส (traversing) และการบายพาส (bypassing) ผ่านไฟร์วอลล์ (firewalls) และมีความเกี่ยวข้องกันกับนโยบายความปลอดภัยบนโลคอล

4.2.4 คุณสมบัติความปลอดภัย (Security Properties)

เป้าหมายพื้นฐานของโพรโตคอลเอสเอสเอช 2 คือความปลอดภัยบนอินเทอร์เน็ตที่ได้รับการปรับปรุง ต่อไปนี้เป็นแนวทางเพื่อให้ง่ายต่อการใช้งาน

- ทุกการอัลกอริทึมการเข้ารหัส, อัลกอริทึมการตรวจสอบความสมบรูณ์ และอัลกอริทึมคีย์สาธารณะที่ใช้ เป็นอัลกอริทึมที่เป็นที่รู้จักกันดี (well-known, well-established)
- ทุกอัลกอริทึมจะถูกใช้กับขนาดคีย์ที่อาศัยเหตุผลในการเข้ารหัสเป็นสำคัญ ซึ่งเชื่อว่าจะให้การป้องกันได้แม้จะเป็นการโจมตีเพื่อวิเคราะห์การเข้ารหัสที่แข็งแกร่งที่สุด (strongest crypanalytic attack) นานนับทศวรรษ
- ทุกอัลกอริทึมจะถูกเจรจา และในกรณีที่บางอัลกอริทึมถูกโจมตีสำเร็จ มันง่ายที่จะสลับไปใช้อัลกอริทึมอื่น โดยปราศจากการปรับเปลี่ยนโพรโตคอลพื้นฐาน

การยินยอมอ่อนข้อถูกทำเพื่อสร้างความง่ายเพื่อให้การใช้งานแพร่หลาย กรณีพิเศษที่เกิดขึ้นคือการตรวจสอบว่าคีย์โฮสต์ของเซิร์ฟเวอร์ เป็นของโฮสต์ที่ต้องการจริง โพรโตคอลอนุญาตให้การตรวจสอบถูกละเว้นได้ (แต่ไม่แนะนำ) เชื่อว่าจะปรับปรุงให้การใช้งานในระยะสั้นๆสะดวกขึ้นอย่างเห็นได้ชัดเจน จนกว่าจะมีโครงสร้างคีย์สาธารณะสำหรับอินเทอร์เน็ตที่กว้างขวางเกิดขึ้น

4.2.5 ขนาดแพ็กเก็ตและโสหุ้ย (Packet Size and Overhead)

ผู้อ่านอาจกังวลเกี่ยวกับการเพิ่มขึ้นของขนาดแพ็กเก็ตเพื่อเฮดเดอร์ใหม่ แพดดิ้ง (padding) ใหม่ และ MAC ใหม่ ขนาดแพ็กเก็ตที่เล็กที่สุดอยู่ในลำดับของ 28 ไบต์ (ขึ้นอยู่กับอัลกอริทึมที่เจรจา) การเพิ่มขึ้นเป็นเรื่องเล็กน้อยสำหรับแพ็กเก็ตขนาดใหญ่ แต่จะเป็นเรื่องสำคัญสำหรับแพ็กเก็ตขนาดไบต์เดียว (เซลชั่น telnet-type) อย่างไรก็ตาม มีหลายปัจจัยที่ทำให้เรื่องนี้ไม่เป็นประเด็นสำคัญในแทบทุกกรณี

- ขนาดที่เล็กที่สุดของเฮดเดอร์ทีซีพี/ไอพี คือ 32 ไบต์ ดังนั้นการเพิ่มขึ้นจริงจะอยู่ในช่วง 33 ถึง 51 ไบต์ (โดยคร่าวๆ)
- ขนาดที่เล็กที่สุดของฟิลด์ข้อมูลในแพ็กเก็ตอีเทอร์เน็ต คือ 46 ไบต์ [RFC-894] ดังนั้นการเพิ่มขึ้นจะไม่มากกว่า 5 ไบต์ เมื่อพิจารณาเฮดเดอร์อีเทอร์เน็ต พบว่าการเพิ่มขึ้นนั้นน้อยกว่า 10 เปอร์เซ็นต์
- เศษรวมของข้อมูล telnet-type ในอินเทอร์เน็ตเป็นเรื่องเล็กน้อยเมื่อขนาดแพ็กเก็ตถูกเพิ่มแล้ว

มีเพียง PPP [RFC-1134] ที่ดูเหมือนจะมีผลกระทบอย่างชัดเจนเมื่อขนาดแพ็กเก็ตถูกเพิ่มขึ้น เมื่อถูกใช้บนการเชื่อมต่อผ่านโมเด็มที่ช้า (PPP บีบอัดเฮดเดอร์ทีซีพี/ไอพี เมื่อมีการเพิ่มขึ้นของแพ็กเก็ต) อย่างไรก็ตาม ด้วยโมเด็มสมัยใหม่ เวลาที่ใช้ในการถ่ายโอนจะอยู่ในลำดับของ 2 มิลลิวินาที ซึ่งเร็วกว่าที่คนจะสามารถพิมพ์มาก

ประเด็นที่เกี่ยวข้องกับขนาดแพ็กเก็ตที่ใหญ่ที่สุด สำหรับเซสชันอินเตอร์แอกทีฟแล้ว เราไม่ต้องการแพ็กเก็ตที่มีขนาดใหญ่เกินความจำเป็น เพื่อลดดีเลย์ในการอัปเดทหน้าจอให้น้อยที่สุด ขนาดแพ็กเก็ตใหญ่สุดจะถูกเจรจาแยกกันสำหรับแต่ละแชนเนล

4.3 การแทนชนิดข้อมูล (Data Type Representation) ที่ใช้ในโพรโตคอลเอสเอสเอช 2

- **byte** *byte* ใช้แทนค่า8 บิตใด ๆ (octet) [RFC-1700] บางครั้งข้อมูลที่มีความยาวคงที่ถูกแทนด้วยอาร์เรย์ของ *byte* เขียนด้วย *byte[n]* โดยที่ *n* เป็นจำนวนของ *byte* ในอาร์เรย์
- **boolean** ค่า *boolean* ถูกเก็บด้วยไบต์เดี่ยว ค่า 0 แทนค่า FALSE และค่า 1 แทนค่า TRUE ค่าที่ไม่เป็นศูนย์ต้องถูกตีความเป็น TRUE อย่างไรก็ตาม โปรแกรมประยุกต์ต้องไม่เก็บค่าอื่นนอกเหนือจาก 0 และ 1
- **uint** 32 ใช้แทนค่าจำนวนเต็มที่ไม่เป็นลบขนาด 32 บิต เก็บลงแบบข้อมูลชนิด *byte* 4 ไบต์ โดยมีลำดับนัยสำคัญเรียงจากมากไปน้อย (เรียงแบบไบต์เครือข่าย) ตัวอย่างเช่น ค่า 699921578 (0x29b7f4aa) ถูกเก็บเรียง 29 b7 f4 aa
- **uint64** ใช้แทนค่าจำนวนเต็มที่ไม่เป็นลบขนาด 64 บิต เก็บลงแบบข้อมูลชนิด *byte* 8 ไบต์ โดยมีลำดับนัยสำคัญเรียงจากมากไปน้อยเช่นเดียวกับ *uint32*
- **string** ใช้แทนสตริงไบนารีความยาวใดๆ สตริงสามารถบรรจุข้อมูลไบนารีใด ๆ ก็ได้ รวมถึงตัวอักขระ null และตัวอักขระ 8 บิต มันถูกเก็บแบบ *uint32* ที่บรรจุความยาวของมันเอง (จำนวนไบต์ที่ตามมา) และ 0 (เป็นสตริงว่าง) หรือหลายไบต์ที่เก็บค่าของสตริง และไม่ใช้ตัวอักขระ null ปิดท้าย

สตริงถูกใช้ในการเก็บข้อความอักษร (text) ในกรณีเช่นนี้ US-ASCII จะถูกใช้สำหรับชื่อภายใน (internal name) และ ISO-10646 UTF-8 สำหรับข้อความอักษรที่อาจถูกแสดงไปยังผู้ใช้ และไม่ควรมีอักขระ null ปิดท้ายเก็บในสตริง

ตัวอย่างเช่น สตริงแบบ US-ASCII "testing" ถูกแทนด้วย 00 00 00 07 t e s t i n g การแมปแบบ UTF8 ไม่ได้ทำการเปลี่ยนแปลงการเข้ารูปแบบของอักขระ US-ASCII

- **mpint** ใช้แทนจำนวนเต็มที่มีความเที่ยงตรงมาก (multiple precision integer) ในรูปแบบทูคอมพลีเมนต์ (two's complement) ถูกจัดเก็บแบบสตริง เริ่มจากไบต์ที่มีนัยสำคัญมากก่อน เลขที่ติดลบจะมีค่าที่บิตนัยสำคัญสูงสุดของไบต์แรกสุดของส่วนข้อมูลเป็น 1 สำหรับบิตนัยสำคัญสูงสุดของจำนวนบวก หากมีค่าเป็น 1 แล้ว จะต้องเพิ่มไบต์ 0 นำหน้า ค่า 0 ต้องถูกเก็บเป็นสตริงที่ไม่มีไบต์ของข้อมูล

ค่า (ฐานแปด)	การแทนค่า (ฐานแปด)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

ตารางที่ 4.1 แสดงตัวอย่างการแทนค่าด้วย mpint

- **name-list** เป็นสตริงที่บรรจุลิสต์ของชื่อที่ใช้อักขระจุลภาค (,) แบ่งคั่นภายใน ลิสต์ชื่อจะถูกแทนด้วย *uint32* ที่บรรจุความยาวของมัน (จำนวนไบต์ที่ตามมา) ตามด้วยลิสต์ของชื่อศูนย์ชื่อหรือมากกว่าที่ใช้อักขระจุลภาคแบ่งคั่นภายใน ชื่อจะต้องมีความยาวไม่เป็นศูนย์และต้องไม่ประกอบด้วยอักขระจุลภาค และอาจมีข้อกำหนดเพิ่มเติมกับชื่อ เช่น จะต้องเป็นไอเดนติฟายเออร์อัลกอริทึมที่ถูกต้อง (ดูในหัวข้อการตั้งชื่ออัลกอริทึม) หรือเป็นแท็กภาษา [RFC-1766] ลำดับของชื่อในลิสต์อาจเป็นหรือไม่เป็นนัยสำคัญก็ได้ ขึ้นอยู่กับคอนเทกซ์ที่ลิสต์นั้นใช้ และทั้งแต่ละชื่อและทั้งลิสต์นั้นไม่ต้องปิดท้ายด้วยอักขระ NUL

ค่า (ฐานแปด)	การแทนค่า (ฐานแปด)
(), ลิสต์ว่าง	00 00 00 00
("zlib")	00 00 00 04 7a 6c 69 62
("zlib", "none")	00 00 00 09 7a 6c 69 62 2c 6e 6f 6e 65

ตารางที่ 4.2 แสดงตัวอย่างการแทนค่าด้วย name-list

4.4 การตั้งชื่ออัลกอริทึม (Algorithm Naming)

โพรโตคอลเอสเอสเอชอ้างถึงอัลกอริทึมแฮช อัลกอริทึมการเข้ารหัส อัลกอริทึมตรวจความสมบรูณ์ อัลกอริทึมการบีบอัด อัลกอริทึมการแลกเปลี่ยนคีย์ หรือโพรโตคอลด้วยชื่อ มีบางอัลกอริทึมมาตรฐานที่ทุกการอิมพลีเมนต์ต้องรองรับ มีบางอัลกอริทึมที่ถูกนิยามไว้ในการกำหนดลักษณะเฉพาะของโพรโตคอลแต่เป็นทางเลือก (optional) ยิ่งไปกว่านี้ บางองค์กรต้องการที่จะใช้อัลกอริทึมของตน

ในโพรโตคอลนี้ ทุกไอเดนติฟายเออร์สำหรับอัลกอริทึมต้องเป็นสตริง US-ASCII ที่สามารถพิมพ์เห็นได้โดยไม่เป็นสตริงว่าง และไม่ยาวเกิน 64 อักขระ ชื่อต้องเป็นแบบมีความแตกต่างในอักษรเล็ก-ใหญ่

ชื่ออัลกอริทึมมี 2 รูปแบบ

- ชื่อที่ไม่ประกอบด้วยอักขระ @ ถูกสงวนไว้เพื่อถูกกำหนดโดยมติของ IETF (RFC) ตัวอย่างเช่น *`3des-cbc'*, *`sha-1'*, *`hmac-sha1'*, และ *`zlib'* (อักขระ ‘ ไม่ได้เป็นส่วนหนึ่งของชื่อ) ชื่อในรูปแบบนี้ต้องไม่ถูกนำมาใช้โดยไม่ผ่านการลงทะเบียนก่อน ชื่อที่ลงทะเบียนแล้วต้องไม่ประกอบด้วยอักขระ @ หรือ ,
- ทุกคนสามารถนิยามอัลกอริทึมเพิ่มเติม โดยใช้ชื่อในรูปแบบ *name@domainname* เช่น "*ourcipher-cbc@ssh.com*" รูปแบบของส่วนที่นำหน้าอักขระ @ ไม่ถูกกำหนดไว้ มันต้องประกอบขึ้นจากอักขระ US-ASCII ยกเว้น @ และ , ส่วนที่ตามหลังเครื่องหมาย @ ต้องเป็นโดเมนสำหรับอินเทอร์เน็ตแบบระบุเต็ม (fully qualified) ที่ถูกต้อง [RFC-1034] และถูกควบคุมโดยบุคคลหรือองค์กรที่นิยามชื่อแล้วแต่ละโดเมนว่าจะจัดการโลคอลเนมสเปซของตนอย่างไร

4.5 หมายเลขแมสเซจ (Message Number)

แพ็กเก็ตเอสเอสเอชมีหมายเลขแมสเซจอยู่ในช่วง 1 ถึง 255 หมายเลขถูกกำหนดไว้ดังนี้

โพรโตคอลชั้นทรานสปอร์ต :

1 ถึง 19	ส่วนหลัก (generic)ของโพรโตคอลชั้นทรานสปอร์ต (เช่น disconnect, ignore, debug เป็นต้น)
20 ถึง 29	การเจรจาอัลกอริทึม
30 ถึง 49	ระบุวิธีการแลกเปลี่ยนคีย์ (หมายเลขสามารถถูกนำมาใช้ใหม่สำหรับวิธีการพิสูจน์ตนที่ต่างกัน)

โพรโตคอลการพิสูจน์ตนผู้ใช้ :

50 ถึง 59 ส่วนหลัก (generic) ของโพรโตคอลพิสูจน์ตนผู้ใช้

60 ถึง 79 ระบุวิธีการพิสูจน์ตนผู้ใช้ (หมายเลขสามารถถูกนำมาใช้ใหม่สำหรับวิธีการพิสูจน์ตนที่ต่างกัน)

โพรโตคอลการเชื่อมต่อ :

80 ถึง 89 ส่วนหลัก (generic) ของโพรโตคอลการเชื่อมต่อ

90 ถึง 127 แมสเซจที่เกี่ยวข้องกับแชนเนล (Channel related messages)

สงวนไว้สำหรับโพรโตคอลบนไคลเอ็นต์ :

128 ถึง 191 สงวนไว้

ส่วนขยายบนโลคอล :

192 ถึง 255 ส่วนขยายบนโลคอล (Local extensions)

4.6 การพิจารณาของ IANA

การจัดสรรชื่อของชนิดต่อไปนี้ในโพรโตคอลเอสเอสเอช ถูกกำหนดโดยมติของ IETF

- ชื่ออัลกอริทึมการเข้ารหัส
- ชื่ออัลกอริทึม MAC
- ชื่ออัลกอริทึมสาธารณะ (อัลกอริทึมคีย์สาธารณะบ่งชี้เป็นนัยในความสามารถของการเข้ารูปแบบและความสามารถของการลงลายเซ็น/การเข้ารหัส)
- ชื่อวิธีการแลกเปลี่ยนคีย์
- ชื่อ (บริการ) โพรโตคอล

ชื่อเหล่านี้ต้องเป็นสตริง US-ASCII ที่สามารถพิมพ์เห็นได้ และต้องไม่ประกอบด้วยอักขระ @ หรือ , หรืออักขระพิมพ์ไม่เห็น (whitespace) หรืออักขระควบคุม (ASCII รหัส 32 หรือน้อยกว่า) ชื่อมีความแตกต่างระหว่างตัวเล็ก-ใหญ่ และต้องไม่ยาวกว่า 64 อักขระ

ชื่อที่มีอักขระ @ ภายใน ถูกจัดสรรโดยเจ้าของชื่อ DNS ที่อยู่หลังอักขระ @ (การจัดสรรแบบลำดับชั้น อธิบายใน [RFC-2343])

แต่ละหมวดหมู่ชื่อที่แสดงดังที่ผ่านมามีเนมสเปซแยกกัน อย่างไรก็ตาม ควรหลีกเลี่ยงการใช้ชื่อเดียวกันในหลายหมวดหมู่

หมายเลขแมสเซจในช่วง 0 ถึง 191 ควรถูกจัดสรรผ่านมติของ IETF หมายเลขแมสเซจในช่วง 192 ถึง 255 ชุดส่วนขยายบนโลคอลถูกสงวนไว้ใช้ส่วนตัว

4.7 พิจารณาความปลอดภัย

ควรใส่ใจเป็นพิเศษ เพื่อให้มั่นใจว่าหมายเลขสุ่มทั้งหมดมีคุณภาพดี หมายเลขสุ่มควรถูกสร้างด้วยกลไกที่ปลอดภัย อธิบายไว้ใน [RFC-1750]

เมื่อมีการแสดงข้อความ เช่นข้อความบอกความผิดพลาด หรือข้อความดีบักแก่ผู้ใช้ ซอร์ฟแวร์ไคลเอ็นต์ควรแทนที่อักขระควบคุมใด ๆ (ยกเว้น tab, carriage return และ newline) ด้วยลำดับที่ปลอดภัย เพื่อหลีกเลี่ยงการโจมตีโดยการส่งอักขระควบคุมเทอร์มินอล

ควรหลีกเลี่ยงการไม่ใช้ MAC หรือการเข้ารหัส ถ้าการเข้ารหัสถูกระงับการใช้งาน โพรโตคอลพิสูจน์ตนผู้ใช้จะเป็นเป้าหมายของการโจมตีแบบแมนอินเดอะมิดเดิล โพรโตคอลเอสเอสเอชจะไม่สามารถป้องกันการเปลี่ยนแปลงข้อความ หาก MAC ไม่ถูกนำไปใช้

4.8 ความแตกต่างจาก SSH-1

- การทำงานของ SSH-2 นั้นจะแยกการทำงานออกเป็น 4 โพรโตคอลด้วยกัน โดยในแต่ละโพรโตคอลนั้นจะมีหน้าที่การทำงานแตกต่างกันออกไปทำให้การทำงานมีความยืดหยุ่นกว่า SSH-1
- สามารถที่จะพัฒนาอัลกอริทึมขึ้นมาใช้งานเองได้ เพราะสนับสนุนการเรียกชื่ออัลกอริทึมด้วยสตริง
- มีการแลกเปลี่ยนกุญแจเซสชั่นเป็นระยะ
- สามารถเลือกได้อัลกอริทึมการเข้ารหัสลับ , การบีบอัดข้อมูล , MAC ได้
- สนับสนุนการเปลี่ยนรหัสผ่าน
- ใช้อัลกอริทึม Diffie-Hellman ในการหาการกุญแจเซสชันซึ่งขจัดความต้องการของกุญแจเซิร์ฟเวอร์ได้ ใช้แต่กุญแจโฮสต์เท่านั้น

แหล่งข้อมูลเพิ่มเติม

[FIPS-186] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 18 May 1994.

[RFC-854] Postel, J. and Reynolds, J: "Telnet Protocol Specification", May 1983.

[RFC-1282] Kantor, B: "BSD Rlogin", December 1991.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, Nov 1987.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC1766] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

บทที่ 5

โพรโตคอลชั้นทรานสปอร์ตเอสเอสเอช 2

5.1 แนะนำ

ชั้นทรานสปอร์ตเอสเอสเอช 2 เป็นโพรโตคอลนำส่งในระดับล่างที่ปลอดภัย ทำให้เกิดการเข้ารหัสที่แข็งแกร่ง การพิสูจน์โฮสต์แบบเข้ารหัส และการปกป้องความสมบูรณ์ของข้อมูล

การพิสูจน์ตนในระดับโพรโตคอลนี้ตั้งอยู่บนพื้นฐานของโฮสต์ (host-based) โดยโพรโตคอลนี้ไม่ได้ดำเนินการพิสูจน์ตนผู้ใช้ โดยการพิสูจน์ตนผู้ใช้งานจะกระทำในโพรโตคอลระดับที่สูงขึ้นไป

โพรโตคอลนี้ถูกออกแบบมาให้เรียบง่าย ยืดหยุ่น อนุญาตให้มีการเจรจาพารามิเตอร์ และลดจำนวนรอบส่งไป-กลับ (number of round-trips) ให้น้อยที่สุด จะมีการเจรจาวิธีการแลกเปลี่ยนคีย์ (key exchange method) อัลกอริทึมคีย์สาธารณะ (public key algorithm) อัลกอริทึมเข้ารหัสแบบสมมาตร (symmetric encryption algorithm) อัลกอริทึมพิสูจน์ข้อความ (message authentication algorithm) และอัลกอริทึมแฮช (hash algorithm) คาดหวังว่าในสภาพการณ์ส่วนใหญ่ จะใช้การส่งไป-กลับเพียง 2 รอบเท่านั้นสำหรับการแลกเปลี่ยนคีย์ทั้งหมด การพิสูจน์ตนเซิร์ฟเวอร์ การร้องขอบริการ และการแจ้งการยอมรับคำร้องขอบริการ กรณีที่เลวร้ายที่สุดจะใช้ถึง 3 รอบ

5.2 การก่อตั้งการเชื่อมต่อ (Connection Setup)

โพรโตคอลเอสเอสเอช 2 ทำงานบน 8 บิตอย่างชัดเจน มีการส่งข้อมูลแบบไบนารี-ทรานส์พาเรนท์ (binary-transparent transport) ในการส่งนั้นควรป้องกันการเกิดความผิดพลาดเนื่องจากการส่งผ่าน เช่นความผิดพลาดอันเป็นสาเหตุให้การเชื่อมต่อเอสเอสเอช 2 ต้องยุติลง ไคลเอ็นต์เริ่มทำการเชื่อมต่อ

5.2.1 ใช้งานบนทีซีพี/ไอพี (Use over TCP/IP)

เมื่อใช้งานบนทีซีพี/ไอพี ปกติเซิร์ฟเวอร์จะรอฟังการเชื่อมต่อที่พอร์ต 22 ซึ่งหมายเลขพอร์ตนี้ได้ลงทะเบียนกับ IANA ที่ได้กำหนดให้แก่โพรโตคอลเอสเอสเอช 2 อย่างเป็นทางการแล้ว

5.2.2 การแลกเปลี่ยนเวอร์ชันของโพรโตคอล (Protocol Version Exchange)

เมื่อการเชื่อมต่อถูกก่อตั้งขึ้น ทั้ง 2 ฝ่ายต้องทำการส่งสตริงไอเดนติฟิเคชั่น (identification string) ซึ่งมันอยู่ในรูปแบบ "*SSH-protoversion-softwareversioncomments*" ตามด้วยอักขระ carriage return และ newline (ASCII 13 และ 11 ตามลำดับ) ทั้งสองฝั่งต้องสามารถดำเนินการกับสตริงไอเดนติฟิเคชั่นที่ไม่มี

อักขระ carriage return ด้วยได้ ไม่มีการส่งอักขระ null สตริงมีความยาวสูงสุดได้ 255 ตัวอักขระ ซึ่งนับรวมอักขระ carriage return และ newline แล้ว

ในส่วนของสตริงไอเดนติฟิเคชั่นที่นำหน้าอักขระ carriage return และ newline นั้น จะถูกนำไปใช้ในการแลกเปลี่ยนคีย์แบบดิฟฟี่-เฮลแมน (Diffie-Hellman key exchange)

เซิร์ฟเวอร์อาจจะส่งบรรทัดของข้อมูลอื่นก่อนส่งสตริงเวอร์ชัน แต่ละบรรทัดควรจบท้ายด้วยอักขระ carriage return และ newline แต่ละบรรทัดต้องไม่ขึ้นต้นด้วย "*SSH-*" และควรถูกเข้ารูปแบบแบบ ISO-10646 UTF-8 [RFC-2279] ไคลเอ็นต์ต้องสามารถดำเนินการกับแต่ละบรรทัดได้ โดยอาจจะเพิกเฉยหรือแสดงให้แก่ผู้ใช้งานบนฝั่งไคลเอ็นต์ ถ้าแสดง ควรใช้การฟิลเตอร์อักขระควบคุม (อธิบายในบทที่ 6) การใช้งานหลักของคุณสมบัตินี้คืออนุญาตให้ทีซีพี-แรพเพอร์ (TCP-wrappers) แสดงข้อความความผิดพลาดก่อนยุติการเชื่อมต่อ

สตริงเวอร์ชันต้องประกอบขึ้นจากอักขระ US-ASCII ที่สามารถพิมพ์เห็นได้ ไม่รวมอักขระว่างเปล่า (whitespace) หรือเครื่องหมายลบ (minus sign) สตริงเวอร์ชันถูกใช้เพื่อให้เพิ่มความเข้ากันได้ และบ่งบอกความสามารถที่แต่ละการอิมพลีเมนต์มี สตริงคอมเมนต์ควรบรรจุข่าวสารเพิ่มเติมที่อาจมีประโยชน์ในการแก้ปัญหาของผู้ใช้ เวอร์ชันของโพรโตคอลนี้คือ 2.0

การแลกเปลี่ยนคีย์จะเริ่มทันทีหลังจากการส่งไอเดนติฟายเออร์นี้ ทุกแพ็กเก็ตที่ตามหลังสตริงไอเดนติฟิคชั่นจะต้องใช้โพรโตคอลแพ็กเก็ตไบนารี (binary packet protocol) ซึ่งจะได้อธิบายต่อไป

5.2.3 ความเข้ากันได้กับเวอร์ชันเอสเอสเอช 2 ที่เก่ากว่า (Compatibility With Old SSH Versions)

ในช่วงแห่งการเปลี่ยนแปลง จำเป็นที่จะต้องทำงานเข้ากันได้กับไคลเอ็นต์และเซิร์ฟเวอร์ที่ได้เคยติดตั้งไปแล้วซึ่งใช้โพรโตคอลที่เป็นเวอร์ชันที่เก่ากว่า ในหัวข้อนี้เกี่ยวข้องกับการอิมพลีเมนต์ที่รองรับความเข้ากันได้กับเวอร์ชันเอสเอสเอช 2 1.x เท่านั้น

5.2.3.1 ไคลเอ็นต์เก่า เซิร์ฟเวอร์ใหม่

การอิมพลีเมนต์เซิร์ฟเวอร์ อาจรองรับแฟลก "*compatibility*" ที่สามารถกำหนดค่าได้ เมื่อแฟลกถูกเซ็ต จะสามารถเข้ากันได้กับเวอร์ชันเก่า เซิร์ฟเวอร์ควรบ่งบอกเวอร์ชันของโพรโตคอลเป็น "1.99" ไคลเอ็นต์ที่ใช้เวอร์ชัน 2.0 ต้องสามารถบ่งชี้ได้ว่าเหมือนเวอร์ชัน "2.0" ในโหมดนี้เซิร์ฟเวอร์ไม่ควรส่งอักขระ carriage return (ASCII 13) หลังสตริงไอ นติฟิเคชั่นของเวอร์ชัน

ในโหมด compatibility เซิร์ฟเวอร์ไ วรส่งข้อมูลอื่นอีกหลังจากส่งสตริงของตน จนกว่าจะได้รับสตริงไอเดนติฟิเคชั่นจากไคลเอ็นต์ ากนั้นเซิร์ฟเวอร์จะสามารถพิจารณาได้ว่าเซิร์ฟเวอร์ใช้โพรโตคอลเก่าหรือไม่ และสามารถเปลี่ยนไป โพรโตคอลเก่าได้ถ้าจำเป็น ในโหมด compatibility เซิร์ฟเวอร์ต้องไม่ส่งข้อมูลเพิ่มก่อนสตริงเวอร์ชัน

เมื่อความเข้ากันได้กับเวอร์ชันเก่าไม่มีความจำเป็นต่อไป เซิร์ฟเวอร์อาจส่งข้อมูลแลกเปลี่ยนคีย์ส่วนแรกทันทีหลังจากสตริงไอเดนติฟิเคชั่น

5.2.3.2 ไคลเอ็นต์ใหม่ เซิร์ฟเวอร์เก่า

เนื่องจากไคลเอ็นต์ใหม่อาจส่งข้อมูลเพิ่มทันทีหลังจากสตริงไอเดนติฟิเคชั่นของมัน (ก่อนการรับไอเดนติฟิเคชั่นของเซิร์ฟเวอร์) ทำให้เมื่อไคลเอ็นต์รู้ว่าเซิร์ฟเวอร์ใช้โพรโตคอลเก่า โพรโตคอลของเซิร์ฟเวอร์อาจล้มเหลวไปแล้ว กรณีนี้ ไคลเอ็นต์ควรทำการปิดการเชื่อมต่อกับเซิร์ฟเวอร์ และทำการเชื่อมต่อใหม่ด้วยโพรโตคอลเก่า

5.3 โพรโตคอลแพ็กเก็ตแบบไบนารี (Binary Packet Protocol)

แต่ละแพ็กเก็ตจะอยู่ในรูปแบบดังนี้

uint32 packet_length
byte padding_length
byte[n1] payload; n1 = packet_length - padding_length - 1
byte[n2] random padding; n2 = padding_length
byte[m] mac (message authentication code); m = mac_length

โดย

packet_length เป็นความยาวของแพ็กเก็ตในหน่วยไบต์ ไม่รวม MAC หรือฟิลด์ *packet_length* เอง

padding_length เป็นความยาวของส่วน *padding* ในหน่วยไบต์

payload เนื้อความที่มีประโยชน์ของแพ็กเก็ต ถ้าการบีบอัดได้ถูกเจรจากันแล้ว ฟิลด์นี้จะถูกบีบอัด โดยในตอนเริ่มต้นการบีบอัดต้องเป็น "*none*"

padding เป็น *padding* ความยาวไม่เจาะจง ซึ่งทำให้ความยาวรวมของ (*packet_length* || *padding_length* || *payload* || *padding*) เป็นจำนวนเท่าของขนาดบล็อคของไซเฟอร์หรือ 8 มันต้องมี *padding* อย่างน้อยที่สุด 4 ไบต์ ส่วน *padding* ควรประกอบขึ้นจากกลุ่มไบต์สุ่ม ความยาวสูงสุดของ *padding* คือ 255 ไบต์

mac คือรหัสพิสูจน์ข้อความ (message authentication code) ถ้าการพิสูจน์ข้อความถูกเจรจาแล้ว ฟิลด์นี้จะประกอบด้วยไบต์ของ MAC โดยในตอนเริ่มต้น อัลกอริทึม MAC ต้องเป็น "*none*"

สังเกตว่าความยาวของการต่อส่วน *packet_length*, *padding_length*, *payload* และ *padding* ต้องเป็นจำนวนเท่าของขนาดบล็อคของไซเฟอร์หรือ 8 ข้อบังคับนี้ต้องถูกบังคับใช้เมื่อใช้ไซเฟอร์แบบสตรีม สังเกตว่าฟิลด์ *packet_length* จะถูกเข้ารหัสด้วย และการดำเนินการกับมันต้องการความเอาใจใส่เป็นพิเศษในขณะทำการส่งหรือรับแพ็กเก็ต

5.3.1 ความยาวแพ็กเก็ตสูงสุด (Maximum Packet Length)

ทุกการอิมพลีเมนต์ต้องสามารถดำเนินการกับแพ็กเก็ตที่มีความยาวของ *payload* ที่ไม่ถูกบีบอัดขนาด 32768 ไบต์หรือน้อยกว่า และขนาดรวมของแพ็กเก็ต (ประกอบด้วย *packet_length*, *padding_length*, *payload*, *padding* และ *MAC*) 35000 ไบต์หรือน้อยกว่าได้ การอิมพลีเมนต์ควรรองรับแพ็กเก็ตที่ยาวกว่า หากมันอาจจำเป็น เช่นถ้าการอิมพลีเมนต์ต้องการส่งใบรับรอง (certificate) จำนวนมาก เป็นต้น บางแพ็กเก็ตอาจถูกส่งได้ถ้าสตริงเวอร์ชันบ่งบอกว่าอีกฝ่ายสามารถดำเนินการกับมันได้ อย่างไรก็ตาม การอิมพลีเมนต์ควรตรวจสอบความยาวแพ็กเก็ตว่ามีเหตุผลสมควร เพื่อหลีกเลี่ยงการโจมตีเพื่อให้หยุดบริการ (denial-of-service attack) หรือและ การโจมตีเพื่อให้เกิดการโอเวอร์โฟลว์ของบัฟเฟอร์ (buffer overflow attack)

5.3.2 การบีบอัด (Compression)

ถ้าการบีบอัดถูกเจรจา ฟิลด์ *payload* (และเพียงฟิลด์นี้เท่านั้น) จะถูกบีบอัดด้วยอัลกอริทึมที่เจรจา ฟิลด์ *length* และ *MAC* จะถูกคำนวณจาก payload ที่บีบอัดแล้ว การเข้ารหัสจะถูกทำหลังจากการบีบอัด

การบีบอัดอาจเป็นแบบสเตทฟูล (stateful) ขึ้นอยู่กับวิธีการ การบีบอัดของแต่ละทิศทางต้องเป็นอิสระต่อกัน และการอิมพลีเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมสำหรับแต่ละทิศทางเป็นอิสระจากกันด้วย

วิธีการบีบอัดถูกนิยามไว้ในขณะนี้ดังนี้

none	*REQUIRED*	*no compression*
zlib	*OPTIONAL*	*GNU ZLIB (LZ77) compression*

การบีบอัด *zlib* ถูกอธิบายไว้ใน [RFC-1950] และใน [RFC-1951] เนื้อความการบีบอัดถูกเตรียมหลังจากแต่ละการแลกเปลี่ยนคีย์ และถูกส่งจากแพ็กเก็ตหนึ่งไปยังแพ็กเก็ตถัดไปด้วยการพาเชียลฟลัช (partial flush) ครั้งหนึ่ง โดยทำที่ส่วนท้ายของแต่ละแพ็กเก็ต การพาเชียลฟลัชหมายถึงข้อมูลทั้งหมดจะถูกเป็นเอาท์พุท แต่แพ็กเก็ตถัดไปจะใช้ตารางการบีบอัด (compression tables) ต่อจากท้ายแพ็กเก็ตที่แล้ว

วิธีการเพิ่มเติมอาจถูกนิยามได้ตามข้อกำหนดในบทที่ 6

5.3.3 การเข้ารหัส (Encryption)

อัลกอริทึมการเข้ารหัสและคีย์จะถูกเจรจาระหว่างการแลกเปลี่ยนคีย์ เมื่อการเข้ารหัสมีผลใช้ ฟิลด์ *packet_length*, *padding_length*, *payload* และ *padding* ของแต่ละแพ็กเก็ตต้องถูกเข้ารหัสด้วยอัลกอริทึมที่กำหนดไว้

ข้อมูลที่เข้ารหัสในทุกแพ็กเก็ตส่งในทิศทางเดียวควรถูกพิจารณาเป็นสตรีมข้อมูลเดียว ยกตัวอย่างเช่น อินิเชียลไลเซชั่นเวกเตอร์ (initialization vector) ควรถูกส่งผ่านจากท้ายของแพ็กเก็ตหนึ่งไปยังส่วนต้นของแพ็กเก็ตถัดไป ทุกไซเฟอร์ควรใช้ความยาวคีย์ที่มีประสิทธิภาพ ขนาด 128 บิตหรือมากกว่า

ไซเฟอร์ในแต่ละทิศทางต้องทำงานเป็นอิสระจากทิศตรงข้าม และการอิมพลีเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมสำหรับแต่ละทิศทางเป็นอิสระกัน (ถ้าอัลกอริทึมที่อนุญาตให้ใช้ตามนโยบายของแต่ละฝั่ง)

ไซเฟอร์ถูกนิยามไว้ในขณะนี้ดังนี้

3des-cbc	*REQUIRED*	*three-key 3DES in CBC mode*
blowfish-cbc	*RECOMMENDED*	*Blowfish in CBC mode*
twofish256-cbc	*OPTIONAL*	*Twofish in CBC mode, with 256-bit key*
twofish-cbc	*OPTIONAL*	*alias for "twofish256-cbc" (this is being retained for historical reasons)*
twofish192-cbc	*OPTIONAL*	*Twofish with 192-bit key*
twofish128-cbc	*RECOMMENDED*	*Twofish with 128-bit key*
aes256-cbc	*OPTIONAL*	*AES (Rijndael) in CBC mode, with 256-bit key*
aes192-cbc	*OPTIONAL*	*AES with 192-bit key*
aes128-cbc	*RECOMMENDED*	*AES with 128-bit key*
serpent256-cbc	*OPTIONAL*	*Serpent in CBC mode, with 256-bit key*
serpent192-cbc	*OPTIONAL*	*Serpent with 192-bit key*
serpent128-cbc	*OPTIONAL*	*Serpent with 128-bit key*
arcfour	*OPTIONAL*	*the ARCFOUR stream cipher*
idea-cbc	*OPTIONAL*	*IDEA in CBC mode*
cast128-cbc	*OPTIONAL*	*CAST-128 in CBC mode*
none	*OPTIOAL*	*no encryption; NOT RECOMMENDED*

ไซเฟอร์ *3des-cbc* triple-DES แบบ 3 คีย์ (เข้ารหัส-ถอดรหัส-เข้ารหัส) โดยที่ 8 ไบต์แรกของคีย์ถูกใช้สำหรับการเข้ารหัสครั้งแรก 8 ไบต์ถัดมาใช้สำหรับการถอดรหัส และ 8 ไบต์ที่ตามมาสำหรับการเข้ารหัสครั้งสุดท้าย ทำให้ต้องการคีย์ขนาด 24 ไบต์ (จาก 168 บิตที่ใช้จริง) เป็นไซเฟอร์แบบบล็อคด้วยขนาดบล็อคขนาด 8 ไบต์ อัลกอริทึมนี้ถูกนิยามไว้ใน [SCHNEIER]

ไซเฟอร์ *blowfish-cbc* เป็น Blowfish แบบ CBC ด้วยคีย์ขนาด 128 บิต เป็นไซเฟอร์แบบบล็อคด้วยบล็อคขนาด 8 ไบต์

ไซเฟอร์ *twofish-cbc*
หรือ *twofish256-cbc* เป็น Twofish ในโหมด CBC ด้วยคีย์ขนาด 128 บิต ซึ่งอธิบายไว้ใน [TWOFISH] เป็นไซเฟอร์แบบบล็อคด้วยบล็อคขนาด 16 ไบต์

ไซเฟอร์ *twofish192-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 192 บิต

ไซเฟอร์ *twofish128-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 128 บิต

ไซเฟอร์ *aes256-cbc* เป็น AES (Advanced Encryption Standard) ชื่อเดิมคือ Rijndael ในโหมด CBC เวอร์ชันนี้ ใช้คีย์ขนาด 256 บิต

ไซเฟอร์ *aes192-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 192 บิต

ไซเฟอร์ *aes128-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 128 บิต

ไซเฟอร์ *serpent256-cbc* ในโหมด CBC ด้วยคีย์ขนาด 256 บิต ตามที่อธิบายใน serpent AES submission

ไซเฟอร์ *serpent192-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 192 บิต

ไซเฟอร์ *serpent128-cbc* เหมือนกับข้างบน แต่คีย์ขนาด 128 บิต

ไซเฟอร์ *arcfour* เป็นไซเฟอร์สตรีม Arcfour ด้วยคีย์ขนาด 128 บิต เชื่อว่าไซเฟอร์ Arcfour สามารถเข้ากันได้กับไซเฟอร์ RC4 [SCHNEIER] RC4 เป็นเครื่องหมายจดทะเบียนของ RSA Data Security Inc. Arcfour (และ RC4) มีปัญหาเรื่องคีย์ที่อ่อนแอ และควรถูกใช้ด้วยความระมัดระวัง

ไซเฟอร์ *idea-cbc* เป็นไซเฟอร์ IDEA ในโหมด CBC [SCHNEIER] IDEA ถูกจดสิทธิบัตรโดย Ascam AG.

ไซเฟอร์ *cast128-cbc* เป็นไซเฟอร์ CAST-128 ในโหมด CBC [RFC-2144]

อัลกอริทึม *none* เป็นการระบุว่าจะไม่มีการทำการเข้ารหัส สังเกตว่าวิธีไม่ทำให้เกิดการปกป้องความมั่นใจ และไม่แนะนำให้ใช้ ถ้าวิธีนี้ถูกเลือก บางการทำงาน (เช่นการพิสูจน์สิทธิ์ด้วยรหัสผ่าน) อาจถูกยกเลิกการใช้งาน (disable) เพื่อเหตุผลด้านความปลอดภัย

วิธีการเพิ่มเติมอาจถูกนิยามได้ตามข้อกำหนดในบทที่ 6

5.3.4 ความสมบูรณ์ของข้อมูล (Data Integrity)

ความสมบูรณ์ของข้อมูลถูกปกป้องด้วยการเพิ่มแต่ละแพ็กเก็ตด้วย MAC ซึ่งถูกคำนวณมาจากความลับร่วม หมายเลขลำดับแพ็กเก็ต และข้อมูลภายในแพ็กเก็ต

อัลกอริทึมพิสูจน์ข้อความและคีย์ ถูกเจรจาระหว่างการแลกเปลี่ยนคีย์ ในตอนเริ่มต้น MAC จะไม่มีผล และความยาวของมันต้องเป็นศูนย์ หลังจากการแลกเปลี่ยนคีย์ MAC ที่เลือกไว้จะถูกคำนวณก่อนการเข้ารหัสจากการต่อข้อมูลในแพ็กเก็ตคือ

mac = MAC(key, sequence_number || unencrypted_packet)

โดย

unencrypted_packet คือทั้งแพ็กเก็ต ไม่รวม MAC (*packet_length*, *padding_length*, *payload*, และ *padding*)

sequence_number เป็นหมายเลขลำดับแพ็กเก็ตโดยอ้อม ที่อยู่ในรูปของ uint32 หมายเลขลำดับถูกเริ่มต้นที่ 0 ที่แพ็กเก็ตแรก และถูกเพิ่มขึ้นทุกๆแพ็กเก็ต (โดยไม่คำนึงว่าใช้การเข้ารหัส หรือ MAC หรือไม่) มันจะไม่ถูกรีเซ็ต แม้ว่าคีย์หรืออัลกอริทึมจะถูกเจรจาใหม่ในภายหลัง มันจะวนกลับมาที่ 0 เมื่อครบทุกๆ 2^{32} แพ็กเก็ต หมายเลขลำดับจะไม่ถูกรวมไปในแพ็กเก็ตที่ส่งออกไป

อัลกอริทึม MAC สำหรับแต่ละทิศทางต้องทำงานเป็นอิสระกัน และการอิมพลีเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมเป็นอิสระต่อกันระหว่างทั้งสองทิศทาง

ไบต์ของ MAC ที่เป็นผลลัพธ์จากอัลกอริทึม MAC ต้องถูกส่งลำเลียงโดยไม่ถูกเข้ารหัส โดยมันจะเป็นส่วนท้ายของแพ็กเก็ต จำนวนไบต์ของ MAC ขึ้นกับอัลกอริทึมที่เลือก

อัลกอริทึม MAC ที่นิยามไว้ในขณะนี้มีดังนี้

hmac-sha1	*REQUIRED*	*HMAC-SHA1 (digest length = key length = 20)*
hmac-sha1-96	*RECOMMENDED*	*first 96 bits of HMAC-SHA1 (digest length = 12, key length = 20)*
hmac-md5	*OPTIONAL*	*HMAC-MD5 (digest length = key length = 16)*
hmac-md5-96	*OPTIONAL*	*first 96 bits of HMAC-MD5 (digest length = 12, key length = 16)*
none	*OPTIONAL*	*no MAC; NOT RECOMMENDED*

อัลกอริทึม *hmac-** ถูกอธิบายไว้ใน [RFC-2104] MAC ในรูป *-n จะใช้เพียง n บิตแรกของค่าผลลัพธ์

อัลกอริทึมแฮชถูกอธิบายไว้ใน [SCHNEIER]

วิธีการ "*none*" ไม่แนะนำให้ใช้ ถ้าใช้วิธีการนี้ ผู้โจมตีแบบแอกทีฟอาจสามารถเปลี่ยนแปลงข้อมูลที่ส่งผ่าน

วิธีการเพิ่มเติมอาจถูกนิยามตามข้อกำหนดในบทที่ 6

5.3.5 วิธีการแลกเปลี่ยนคีย์ (Key Exchange Methods)

วิธีการแลกเปลี่ยนคีย์ระบุถึงการสร้างคีย์สำหรับเซสชันแบบใช้ครั้งเดียว (one-time session keys) ที่จะนำไปใช้ในการเข้ารหัสและการพิสูจน์ตน นั้นทำอย่างไร และระบุถึงการพิสูจน์เซิร์ฟเวอร์นั้นทำอย่างไร

มีเพียงวิธีการแลกเปลี่ยนคีย์เดียวที่จำเป็น นิยามไว้ดังนี้

diffie-hellman-group1-sha1 *REQUIRED*

5.3.6 อัลกอริทึมคีย์สาธารณะ (Public Key Algorithms)

โพรโตคอลนี้ถูกออกแบบมาให้สามารถใช้งานร่วมกับแทบทุก รูปแบบคีย์สาธารณะ การเข้ารูปแบบ และอัลกอริทึม (การลงลายเซ็น และหรือ การเข้ารหัส)
มีหลายแง่มุมที่ใช้ในการนิยามชนิดของคีย์สาธารณะ

- **รูปแบบคีย์ (key format)** คีย์ถูกเข้ารูปแบบอย่างไร และใบรับรองอยู่ในรูปแบบใด บล็อบคีย์ในโพรโตคอลนี้อาจบรรจุใบรับรอง
- **อัลกอริทึมการลงลายเซ็น และหรือการเข้ารหัส** คีย์บางชนิดไม่รองรับทั้งการลงลายเซ็นและการเข้ารหัส การใช้คีย์อาจถูกบังคับตามข้อกำหนดของนโยบายเช่นในใบรับรอง ในกรณีนี้ นโยบายที่ต่างกันควรกำหนดคีย์ที่ต่างชนิดกัน
- **การเข้ารูปแบบของ ลายเซ็น และหรือ ข้อมูลที่เข้ารหัสแล้ว** อาจรวมถึง การแพดดิ้ง, ลำดับไบต์ และรูปแบบข้อมูล

รูปแบบของ คีย์สาธารณะ และหรือ ใบรับรอง นิยามไว้ดังนี้

ssh-dss	*REQUIRED*	*sign*	*Simple DSS*
ssh-rsa	*RECOMMENDED*	*sign*	*Simple RSA*
x509v3-sign-rsa	*RECOMMENDED*	*sign*	*X.509 certificates (RSA key)*
x509v3-sign-dss	*RECOMMENDED*	*sign*	*X.509 certificates (DSS key)*
spki-sign-rsa	*OPTIONAL*	*sign*	*SPKI certificates (RSA key)*
spki-sign-dss	*OPTIONAL*	*sign*	*SPKI certificates (DSS key)*
pgp-sign-rsa	*OPTIONAL*	*sign*	*OpenPGP certificates (RSA key)*
pgp-sign-dss	*OPTIONAL*	*sign*	*OpenPGP certificates (DSS key)*

อาจนิยามชนิดของคีย์เพิ่มเติมได้ ตามข้อกำหนดในบทที่ 6

ชนิดของคีย์ต้องสามารถรู้เสมอ (จากการเจรจาอัลกอริทึมหรือจากแหล่งอื่น) ปกติแล้วมันจะไม่อยู่ในบล็อบคีย์

ใบรับรองและคีย์สาธารณะถูกเข้ารูปแบบดังนี้

string *certificate or public key format identifier*

byte[n] *key/certificate data*

ส่วนของใบรับรองอาจเป็นสตริงว่าง (ความยาวศูนย์) แต่คีย์สาธารณะจำเป็นต้องมี เป็นคีย์สาธารณะที่จะถูกใช้ในการพิสูจน์ตน กลุ่มใบรับรองที่บรรจุในบล็อบใบรับรองสามารถนำไปใช้ในการกำหนดสิทธิ์

รูปแบบคีย์ "*ssh-dss*" กำหนดไว้ดังนี้

string *"ssh-dss"*

mpint *p*

mpint *q*

mpint *g*

mpint *y*

โดยพารามิเตอร์ *p*, *q*, *g* และ *y* ก่อรูปแบบเป็นบล็อบคีย์ลายเซ็น (signature key blob)

การลงลายเซ็นและการตรวจสอบที่ใช้รูปแบบคีย์นี้ โดยเป็นไปตามมาตรฐานลายเซ็นดิจิตอล (Digital Signature Standard) [FIPS-186] ที่ใช้อัลกอริทึมแฮช SHA-1 รายละเอียดอธิบายใน [SCHNEIER]

ลายเซ็นผลลัพธ์มีรูปแบบดังนี้

string *"ssh-dss"*

string *dss_signature_blob*

dss_signature_blob ถูกเข้ารูปแบบเป็นสตริงที่บรรจุ *r* และ *s* (เป็น long integer ขนาด 160 บิต ไม่มีส่วนความยาวหรือแพดดิ้ง ไม่ลงลายเซ็น และเรียงลำดับไบต์แบบเน็ตเวิร์ก)

รูปแบบคีย์ "*ssh-rsa*" ถูกกำหนดการเข้ารูปแบบดังนี้

string *"ssh-rsa"*

mpint *e*

mpint *n*

โดยพารามิเตอร์ *e* และ *n* ก่อรูปแบบเป็นบล็อบคีย์ลายเซ็น

การลงลายเซ็นและการตรวจสอบที่ใช้รูปแบบคีย์นี้ โดยเป็นไปเอกสาร [SCHNEIER] และ [PKCS1] ที่ใช้อัลกอริทึมแฮช SHA-1

ลายเซ็นผลลัพธ์มีรูปแบบดังนี้

string *"ssh-rsa"*

string *rsa_signature_blob*

rsa_signature_blob ถูกเข้ารูปแบบเป็นสตริงที่บรรจุ s (เป็น integer ไม่มีส่วนความยาวหรือแพดดิ้ง ไม่ลงลายเซ็น และเรียงลำดับไบต์แบบเน็ตเวิร์ก)

วิธี "*x509v3-sign-rsa*"	บ่งชี้ถึงใบรับรอง คีย์สาธารณะ และลายเซ็นผลลัพธ์ในรูปแบบ DER-encoded ที่เข้ากันได้กับ X.509V3 (X.509v3 compatible DER-encoded format) รูปแบบที่ใช้ใน X.509V3 อธิบายไว้ใน [RFC-2459] วิธีการนี้บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ RSA
วิธี "*x509v3-sign-dss*"	คล้ายกับวิธี "x509v3-sign-rsa" แต่บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ DSS
วิธี "*spki-sign-rsa*"	บ่งชี้ถึงบล็อบใบรับรองบรรจุกลุ่มใบรับรองแบบ SPKI รูปแบบใบรับรอง SPKI ถูกอธิบายไว้ใน [RFC-2693] วิธีนี้บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ RSA
วิธี "*spki-sign-dss*"	คล้ายกับวิธี "*spki-sign-rsa*" แต่บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ DSS
วิธี "*pgp-sign-rsa*"	บ่งชี้ถึงใบรับรอง คีย์สาธารณะ และลายเซ็นรูปแบบไบนารีที่เข้ากันได้กับ OpenPGP (OpenPGP compatible binary format) ซึ่งถูกอธิบายไว้ใน [RFC-2440] วิธีนี้บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ RSA
วิธี "*pgp-sign-dss*"	คล้ายกับวิธี "*pgp-sign-rsa*" แต่บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ DSS

5.4 การแลกเปลี่ยนคีย์ (Key Exchange)

การแลกเปลี่ยนคีย์เริ่มโดยแต่ละฝั่งส่งลิสต์ของอัลกอริทึมที่รองรับ แต่ละฝั่งจะมีอัลกอริทึมเสนอในแต่ละหมวด และถือว่าการอิมพลีเมนต์ส่วนใหญ่ใช้อัลกอริทึมเสนอเหมือนกัน แต่ละฝั่งอาจเดาอัลกอริทึมที่อีกฝั่งจะใช้ และอาจส่งแพ็กเก็ตแลกเปลี่ยนคีย์เริ่มต้นที่เป็นไปตามอัลกอริทึมถ้าตรงกับวิธีการเสนอ

การเดาถูกพิจารณาว่าผิด ถ้า

- เดาอัลกอริทึมแลกเปลี่ยนคีย์ และหรือ อัลกอริทึมคีย์โฮสต์ผิด (เซิร์ฟเวอร์และไคลเอ็นต์มีอัลกอริทึมเสนอต่างกัน) หรือ
- อัลกอริทึมอื่น ๆ ไม่สามารถใช้ตกลงกันได้ (การทำงานถูกนิยามไว้ในหัวข้อ การเจรจาอัลกอริทึม)

หากไม่ตรงกับที่กล่าวมา การเดาจะถูกพิจารณาว่าถูกต้อง และแพ็กเก็ตที่ถูกส่งจะถูกจัดการเป็นแพ็กเก็ตแลกเปลี่ยนคีย์เริ่มต้น

อย่างไรก็ตาม ถ้าการเดาผิดพลาด และแพ็กเก็ตได้ถูกส่งจากฝั่งใดฝั่งหนึ่งหรือจากทั้งสองฝั่งแล้ว แพ็กเก็ตเหล่านั้นจะต้องถูกเพิกเฉย (แม้ว่าการเดาผิดพลาดไม่ได้มีผลต่อข้อมูลภายในแพ็กเก็ต) และฝั่งที่เหมาะสมต้องส่งแพ็กเก็ตเริ่มต้นที่ถูกต้อง

การพิสูจน์เซิร์ฟเวอร์ในการแลกเปลี่ยนคีย์อาจทำโดยอ้อม หลังจากการแลกเปลี่ยนคีย์ด้วยการพิสูจน์เซิร์ฟเวอร์โดยอ้อมแล้ว ไคลเอ็นต์ต้องรอการตอบสนองของแมสเซจร้องขอบริการที่ส่งไป ก่อนที่จะทำการส่งข้อมูลอื่นต่อไป

5.4.1 การเจรจาอัลกอริทึม (Algorithm Negotiation)

การแลกเปลี่ยนคีย์เริ่มโดยแต่ละฝั่งส่งแพ็กเก็ตต่อไปนี้

byte SSH_MSG_KEXINIT
byte[16] cookie (random bytes)
string kex_algorithms
string server_host_key_algorithms
string encryption_algorithms_client_to_server
string encryption_algorithms_server_to_client
string mac_algorithms_client_to_server
string mac_algorithms_server_to_client
string compression_algorithms_client_to_server
string compression_algorithms_server_to_client
string languages_client_to_server
string languages_server_to_client
boolean first_kex_packet_follows
uint32 0 (reserved for future extension)

แต่ละสตริงอัลกอริทึมต้องเป็นลิสต์ของชื่ออัลกอริทึมที่คั่นด้วยอักขระ , (ดูการตั้งชื่ออัลกอริทึมในบทที่ 6) แต่ละอัลกอริทึมที่รองรับ (อนุญาต) ต้องถูกลิสต์ตามลำดับการเสนอ

อัลกอริทึมแรกในแต่ละลิสต์ต้องเป็นอัลกอริทึมเสนอ (เดา) แต่ละสตริงต้องบรรจุชื่ออัลกอริทึมอย่างน้อยที่สุด 1 ชื่อ

cookie

คุ้กกี้ต้องเป็นค่าสุ่มที่สร้างขึ้นโดยผู้ส่ง จุดประสงค์คือเพื่อทำให้การที่ฝั่งใดฝั่งหนึ่งรู้คีย์ และไอเดนติฟายเออร์ของเซสชัน (session identifier) นั้นเป็นไปไม่ได้

kex_algorithm

อัลกอริทึมการแลกเปลี่ยนคีย์ได้นิยามดังที่กล่าวมา อัลกอริทึมแรกต้องเป็นอัลกอริทึมเสนอ (และเดา) ถ้าทั้งสองฝั่งเดาตรงกัน อัลกอริทึมนั้นจะต้องถูกใช้ ในทางตรงกันข้าม อัลกอริทึมต่อไปจะต้องถูกเลือกใช้ในการแลกเปลี่ยนคีย์ ทำซ้ำบน *kex_algorithm* ของไคลเอ็นต์ภายใน 1 รอบ เลือกอัลกอริทึมแรกที่สอดคล้องเงื่อนไขต่อไปนี้

- เซิร์ฟเวอร์รองรับอัลกอริทึมนั้น
- ถ้าอัลกอริทึมต้องการคีย์โฮสต์ที่สามารถใช้ในการเข้ารหัส มีอัลกอริทึมที่สามารถใช้ในการเข้ารหัสใน *server_host_key_algorithms* ของเซิร์ฟเวอร์ ซึ่งไคลเอ็นต์รองรับด้วย
- ถ้าอัลกอริทึมต้องการคีย์โฮสต์ที่สามารถใช้ในการลงลายเซ็น มีอัลกอริทึมที่สามารถใช้ในการลงลายเซ็นใน *server_host_key_algorithms* ของเซิร์ฟเวอร์ ซึ่งไคลเอ็นต์รองรับด้วย

ถ้าไม่พบอัลกอริทึมที่สอดคล้องเงื่อนไข การเชื่อมต่อจะล้มเหลว และทั้งสองฝั่งต้องยุติการเชื่อมต่อ

server_host_key_algorithms

ลิสต์ของอัลกอริทึมที่รองรับคีย์โฮสต์ของเซิร์ฟเวอร์ (server host key) เซิร์ฟเวอร์จะลิสต์อัลกอริทึมที่มีคีย์โฮสต์ ไคลเอ็นต์จะลิสต์อัลกอริทึมที่ตนตั้งใจจะยอมรับ (โฮสต์หนึ่งอาจมีหลายคีย์โฮสต์ อาจใช้กับอัลกอริทึมที่ต่างกัน)

บางคีย์โฮสต์อาจไม่รองรับทั้งการลงลายเซ็นและการเข้ารหัส (สามารถพิจารณาจากอัลกอริทึม) และทุกคีย์โฮสต์ไม่สามารถใช้กับทุกวิธีการแลกเปลี่ยนคีย์ได้

การเลือกอัลกอริทึมขึ้นกับอัลกอริทึมการแลกเปลี่ยนคีย์ที่เลือก ต้องการคีย์โฮสต์ที่สามารถลงลายเซ็นหรือเข้ารหัสได้หรือไม่ มันต้องสามารถพิจารณาได้จากชื่ออัลกอริทึมคีย์สาธารณะ อัลกอริทึมแรกในลิสต์ของไคลเอ็นต์ที่สอดคล้องกับความต้องการ และเซิร์ฟเวอร์รองรับ ต้องถูกเลือก ถ้าไม่มีอัลกอริทึมใดเลย ทั้งสองฝั่งต้องยุติการเชื่อมต่อ

encryption_algorithms

เป็นลิสต์ของอัลกอริทึมการเข้ารหัสแบบสมมาตรที่ยอมรับ ตามลำดับการเสนอ อัลกอริทึมการเข้ารหัสที่เลือกสำหรับแต่ละทิศทาง ต้องเป็นอัลกอริทึมแรกในลิสต์ของไคลเอ็นต์ซึ่งอยู่ในลิสต์ของเซิร์ฟเวอร์ด้วย ถ้าไม่มีอัลกอริทึมใดเลย ทั้งสองฝั่งต้องยุติการเชื่อมต่อ
สังเกตว่า "*none*" ต้องอยู่ในลิสต์ด้วย ถ้ายอมรับ ชื่ออัลกอริทึมที่นิยามไว้ แสดงไว้ในหัวข้อการเข้ารหัส

mac_algorithms

เป็นลิสต์ของอัลกอริทึม MAC ที่ยอมรับ ตามลำดับการเสนอ อัลกอริทึม MAC ที่เลือกต้องเป็นอัลกอริทึมแรกในลิสต์ของไคลเอ็นต์ซึ่งอยู่ในลิสต์ของเซิร์ฟเวอร์ด้วย ถ้าไม่มีอัลกอริทึมใดเลย ทั้งสองฝั่งต้องยุติการเชื่อมต่อ

สังเกตว่า "*none*" ต้องอยู่ในลิสต์ด้วย ถ้ายอมรับ ชื่ออัลกอริทึม MAC แสดงไว้ในหัวข้อความสมบูรณ์ของข้อมูล

compress_algorithms

เป็นลิสต์ของอัลกอริทึมบีบอัดที่ยอมรับ ตามลำดับการเสนอ อัลกอริทึมที่เลือกต้องเป็นอัลกอริทึมแรกในลิสต์ของไคลเอ็นต์ซึ่งอยู่ในลิสต์ของเซิร์ฟเวอร์ด้วย ถ้าไม่มีอัลกอริทึมใดเลย ทั้งสองฝั่งต้องยุติการเชื่อมต่อ

สังเกตว่า "*none*" ต้องอยู่ในลิสต์ด้วย ถ้ายอมรับ ชื่ออัลกอริทึมบีบอัดแสดงไว้ในหัวข้อการบีบอัด

langauges

เป็นลิสต์ของแท็กภาษา [RFC-1766] ที่คั่นด้วยอักขระ , ตามลำดับการเสนอ ทั้งสองฝั่งอาจเพิกเฉยลิสต์นี้ ถ้าไม่มีภาษาที่เสนอ ลิสต์นี้ควรว่าง

first_kex_packet_follows

บ่งชี้ว่าแพ็กเก็ตการแลกเปลี่ยนคีย์โดยเดา (guessed key exchange packet) จะมีตามมาหรือไม่ ถ้าแพ็กเก็ตเดาจะถูกส่ง ต้องเป็น TRUE ในกรณีตรงกันข้าม ต้องเป็น FALSE

หลังจากรับแพ็กเก็ต SSH_MSG_KEXINIT จากฝั่งตรงข้ามแล้ว แต่ละฝั่งจะรู้ว่าการเดาของตนถูกต้องหรือไม่ ถ้าการเดาของฝั่งตรงข้ามผิด ฟิลด์นี้จะเป็น TRUE แพ็กเก็ตต่อไปจะถูกเพิกเฉย และทั้งสองฝั่งต้องปฏิบัติตามที่กำหนดโดยวิธีการแลกเปลี่ยนคีย์ที่เจรจาไว้ ถ้าการเดาถูกต้อง การแลกเปลี่ยนคีย์ต้องดำเนินการต่อโดยใช้แพ็กเก็ตเดา

หลังจากการแลกเปลี่ยนแพ็กเก็ต KEXINIT แล้ว อัลกอริทึมการแลกเปลี่ยนคีย์จะทำงาน มันอาจเกี่ยวเนื่องกับการแลกเปลี่ยนหลายแพ็กเก็ต ตามที่ระบุโดยวิธีการแลกเปลี่ยนคีย์

5.4.2 เอาท์พุทที่ได้จากการแลกเปลี่ยนคีย์ (Output from Key Exchange)

การแลกเปลี่ยนคีย์สร้าง 2 ค่าคือ ค่าความลับร่วม (shared secret) *K* และแฮชแลกเปลี่ยน (exchange hash) *H* คีย์การเข้ารหัสและคีย์การพิสูจน์ตนได้มาจาก 2 ค่านี้ แฮชแลกเปลี่ยน *H* จากการแลกเปลี่ยนคีย์ครั้งแรก ถูกใช้เป็นไอเดนติฟายเออร์ของเซสชันด้วย ซึ่งเป็นไอเดนติฟายเออร์ที่เป็นเอกลักษณ์สำหรับการเชื่อมต่อ ถูกใช้ในการพิสูจน์ตนโดยเป็นส่วนหนึ่งของข้อมูลที่ถูกลงลายเซ็น เพื่อใช้พิสูจน์ความเป็นเจ้าของของคีย์ส่วนตัว เมื่อคำนวณแล้ว ไอเดนติฟายเออร์ของเซสชันจะไม่ถูกเปลี่ยนแปลง แม้ว่าคีย์จะถูกเปลี่ยนแปลงใหม่ในภายหลัง

แต่ละวิธีการแลกเปลี่ยนคีย์ระบุฟังก์ชันแฮชที่ใช้ในการแลกเปลี่ยนคีย์ ในการได้มาซึ่งคีย์ต้องใช้อัลกอริทึมแฮชเดียวกัน จากนี้จะเรียกว่า *HASH*

คีย์เข้ารหัสต้องถูกคำนวณจากค่าแฮชของค่าที่รู้และ *K* ดังนี้

- IV เริ่มต้นจากไคลเอ็นต์ไปยังเซิร์ฟเวอร์ : *HASH(K || H || "A" || session_id)* โดย *K* ถูกเข้ารูปแบบเป็น *mpint* และ "*A*" เป็น *byte* และ *session_id* เป็นข้อมูลดิบ "*A*" หมายถึงอักขระ *A* ตัวเดียว ASCII คือ 65
- IV เริ่มต้นจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์ : *HASH(K || H || "B" || session_id)*
- คีย์เข้ารหัสจากไคลเอ็นต์ไปยังเซิร์ฟเวอร์ : *HASH(K || H || "C" || session_id)*
- คีย์เข้ารหัสจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์ : *HASH(K || H || "D" || session_id)*
- คีย์ความสมบูรณ์จากไคลเอ็นต์ไปยังเซิร์ฟเวอร์ : *HASH(K || H || "E" || session_id)*
- คีย์ความสมบูรณ์จากเซิร์ฟเวอร์ไปยังไคลเอ็นต์ : *HASH(K || H || "F" || session_id)*

ข้อมูลคีย์ต้องนำมาจากส่วนต้นของเอาท์พุทที่ได้จากการแฮช สำหรับอัลกอริทึมที่มีคีย์ความยาวเปลี่ยนแปลงได้ควรใช้คีย์ความยาว 128 บิต (16 ไบต์) สำหรับอัลกอริทึมอื่นๆ ไบต์มากเท่าที่จำเป็นจะนำมาจากส่วนต้นของค่าแฮช ถ้าความยาวคีย์ยาวกว่าเอาท์พุทของ *HASH* คีย์จะถูกเพิ่มด้วยการคำนวณ *HASH* ของการต่อ *K* และ *H* และทั้งคีย์ และนำไบต์ผลลัพธ์ (มากเท่าที่แฮชสร้างขึ้น) ต่อท้ายเข้าที่คีย์ กระบวนการนี้ถูกทำซ้ำจนกระทั่งคีย์จะยาวพอ คีย์จะได้จากส่วนต้นของค่านี้

K1 = HASH(K || H || X || session_id) *(X is e.g. "A")*

K2 = HASH(K || H || K1)

K3 = HASH(K || H || K1 || K2)

...

key = K1 || K2 || K3 || ...

กระบวนการนี้จะสูญเสียเอนโทรปี (entropy) ถ้าผลรวมของเอนโทรปีใน *K* มากกว่าขนาดสเตทภายในของ *HASH*

5.4.3 การนำคีย์ไปใช้งาน (Taking Keys Into Use)

การแลกเปลี่ยนคีย์จบลงโดยแต่ละฝั่งทำการส่งแมสเซจ *SSH_MSG_NEWKEYS* แมสเซจนี้ถูกส่งพร้อมกับคีย์เดิมและอัลกอริทึมเดิม ทุกแมสเซจที่ส่งหลังจากนี้ต้องใช้คีย์ใหม่และอัลกอริทึมใหม่

เมื่อได้รับแมสเซจนี้ คีย์ใหม่และอัลกอริทึมใหม่ต้องถูกนำมาใช้ในการรับ

แมสเซจนี้เป็นแมสเซจที่ถูกต้องหลังจากการแลกเปลี่ยนคีย์เท่านั้น รวมถึงแมสเซจ *SSH_MSG_DEBUG, SSH_MSG_DISCONNECT* และ *SSH_MSG_IGNORE* ด้วย จุดประสงค์ของแมสเซจนี้เพื่อให้แน่ใจว่า แต่ละฝั่งสามารถที่จะตอบสนองด้วยแมสเซจยุติการเชื่อมต่อที่อีกฝั่งสามารถเข้าใจมื่อการแลกเปลี่ยนคีย์มีบางสิ่งผิดพลาดไป หลังจากการแลกเปลี่ยนคีย์ การอิมพลีเมนต์ต้องไม่ยอมรับแมสเซจอื่นๆก่อนที่จะได้รับ *SSH_MSG_NEWKEYS*

byte *SSH_MSG_NEWKEYS*

5.5 การแลกเปลี่ยนคีย์ใหม่ (Key Re-Exchange)

การแลกเปลี่ยนคีย์ใหม่จะถูกเริ่มโดยการส่งแพ็กเก็ต *SSH_MSG_KEXINIT* เมื่อไม่สามารถทำการแลกเปลี่ยนคีย์ได้ (อธิบายไว้ในหัวข้อการเจรจาอัลกอริทึม) เมื่อได้รับแมสเซจนี้ จะต้องตอบสนองด้วยแมสเซจ *SSH_MSG_KEXINIT* ของตนเอง ยกเว้นเมื่อ *SSH_MSG_KEXINIT* ที่ได้รับได้ถูกตอบกลับไปเรียบร้อยแล้ว ฝ่ายใดฝ่ายหนึ่งอาจทำการเริ่มแลกเปลี่ยนคีย์ใหม่ แต่บทบาทต้องไม่ถูกเปลี่ยนไป (เช่น เซิร์ฟเวอร์ยังคงเป็นเซิร์ฟเวอร์ และไคลเอ็นต์ก็ยังคงเป็นไคลเอ็นต์)

การแลกเปลี่ยนคีย์ใหม่ถูกดำเนินการ โดยใช้การเข้ารหัสอะไรก็ได้ซึ่งจะมีผลเมื่อการแลกเปลี่ยนถูกเริ่มต้นขึ้น การเข้ารหัส, การบีบอัด และวิธีการ MAC จะ ไม่เปลี่ยนก่อนที่ *SSH_MSG_NEWKEYS* ใหม่จะถูกส่งหลังจากการแลกเปลี่ยนคีย์ (เหมือนกับการแลกเปลี่ยนคีย์เริ่มต้น) การแลกเปลี่ยนใหม่ถูกทำกระบวนการเหมือนกับการแลกเปลี่ยนคีย์เริ่มต้นเว้นแต่ไอเดนติฟายเออร์ของเซสชันจะยังคงไม่เปลี่ยนแปลง ทั้งนี้ยอมให้มีการเปลี่ยนอัลกอริทึมบางส่วน หรือทั้งหมดระหว่างการแลกเปลี่ยนคีย์ใหม่ และคีย์โฮสต์สามารถเปลี่ยนใหม่ได้ด้วย คีย์และอินิเชียลไลเซชันเวกเตอร์ทั้งหมดถูกคำนวณใหม่หลังจากการแลกเปลี่ยน การบีบอัดและการเข้ารหัสจะถูกรีเซ็ต

แนะนำให้เปลี่ยนคีย์หลังจากทำการส่งผ่านข้อมูลครบทุกๆ กิกะไบต์ หรือทุก ๆ ชั่วโมงของเวลาการเชื่อมต่อ โดยอะไรครบก็ทำวิธีนั้นก่อน อย่างไรก็ตามเนื่องจากการแลกเปลี่ยนคีย์ใหม่เป็นการทำงานคีย์ของสาธารณะ จำเป็นที่ใช้การประมวลผลที่คุ้มค่าและ ไม่ควรทำบ่อยเกินไป

ข้อมูลโปรแกรมประยุกต์จำนวนมากอาจถูกส่งหลังจากแพ็กเก็ต *SSH_MSG_NEWKEYS* ได้ถูกส่งไปแล้ว การแลกเปลี่ยนคีย์ไม่มีผลกระทบต่อโพรโตคอลที่ทำงานตัวอยู่เหนือชั้นทรานสปอร์ตเอสเอสเอช2

5.6 การร้องขอบริการ (Service Request)

หลังจากการแลกเปลี่ยนคีย์ ไคลเอ็นต์ร้องขอบริการ บริการถูกบ่งชี้โดยชื่อ รูปแบบของชื่อและกระบวนการนิยามชื่อใหม่ได้นิยามไว้ในบทที่ 6

ปัจจุบันชื่อต่อไปนี้ถูกสงวนไว้

ssh-userauth

ssh-connection

นโยบายการตั้งชื่อบนโลคอลถูกประยุกต์เข้ากับชื่อบริการ เหมือนกับที่ประยุกต์เข้ากับชื่ออัลกอริทึม บริการบนโลคอลควรใช้ไวยากรณ์ "*servicename@domain*"

byte *SSH_MSG_SERVICE_REQUEST*

string *service name*

ถ้าเซิร์ฟเวอร์ปฏิเสธการร้องขอบริการนี้ มันควรส่งแมสเซจ *SSH_MSG_DISCONNECT* ที่เหมาะสม และต้องยุติการเชื่อมต่อ

เมื่อการบริการเริ่มขึ้น มันสามารถเข้าใช้ไอเดนติฟายเออร์ของเซสชันที่สร้างขึ้นระหว่างการแลกเปลี่ยนคีย์

ถ้าเซิร์ฟเวอร์รองรับการบริการ (และอนุญาตให้ไคลเอ็นต์ใช้ได้) เซิร์ฟเวอร์ต้องตอบสนองดังนี้

byte *SSH_MSG_SERVICE_ACCEPT*

string *service name*

หมายเลขแมสเซจที่ใช้โดยบริการควรอยู่ในขอบเขตสงวนไว้สำหรับบริการนั้น ๆ (ดูในหัวข้อสรุปหมายเลขแมสเซจ) ชั้นทรานสปอร์ตจะดำเนินการกระบวนการด้วยแมสเซจของมันต่อไป

สังเกตว่าหลังจากการแลกเปลี่ยนด้วยการพิสูจน์เซิร์ฟเวอร์แล้ว ไคลเอ็นต์ต้องรอการตอบสนองต่อแมสเซจการร้องขอบริการของมันก่อนจะส่งข้อมูลใด ๆ ต่อไป

5.7 แมสเซจเพิ่มเติม (Additional Messages)

แต่ละฝั่งอาจส่งแมสเซจตอไปนี้ ณ เวลาใดก็ได้

5.7.1 แมสเซจยุติการเชื่อมต่อ (Disconnection Message)

byte *SSH_MSG_DISCONNECT*

uint32 *reason code*

string *description [RFC-2279]*

string *language tag [RFC-1766]*

แมสเซจนี้เป็นเหตุให้การเชื่อมต่อยุติลงทันที ทุกการอิมพลีเมนต์ต้องสามารถจัดการกับแมสเซจนี้ได้ มันควรสามารถส่งแมสเซจนี้

ผู้ส่งต้องไม่ส่งหรือรับข้อมูลใด ๆ หลังจากแมสเซจนี้ และผู้รับต้องไม่ยอมรับข้อมูลใด ๆ หลังจากได้รับแมสเซจนี้ ฟิลด์ '*description*' ให้คำอธิบายเพิ่มซึ่งอยู่ในรูปแบบที่สามารถอ่านได้ '*error code*' บอกถึงเหตุผลอยู่ในรูปแบบที่เครื่องเข้าใจ และมีค่าดังนี้

#define SSH_DISCONNECT_HOST_NOT_ALLOWED_TO_CONNECT 1
#define SSH_DISCONNECT_PROTOCOL_ERROR 2
#define SSH_DISCONNECT_KEY_EXCHANGE_FAILED 3
#define SSH_DISCONNECT_RESERVED 4
#define SSH_DISCONNECT_MAC_ERROR 5
#define SSH_DISCONNECT_COMPRESSION_ERROR 6
#define SSH_DISCONNECT_SERVICE_NOT_AVAILABLE 7
#define SSH_DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED 8
#define SSH_DISCONNECT_HOST_KEY_NOT_VERIFIABLE 9
#define SSH_DISCONNECT_CONNECTION_LOST 10
#define SSH_DISCONNECT_BY_APPLICATION 11
#define SSH_DISCONNECT_TOO_MANY_CONNECTIONS 12
#define SSH_DISCONNECT_AUTH_CANCELLED_BY_USER 13
#define SSH_DISCONNECT_NO_MORE_AUTH_METHODS_AVAILABLE 14
#define SSH_DISCONNECT_ILLEGAL_USER_NAME 15

ถ้าสตริง '*description*' ถูกแสดง การฟิลเตอร์อักขระควบคุมดังที่อธิบายในบทที่ 6 ควรถูกนำมาใช้ เพื่อหลีกเลี่ยงการโจมตีโดยการส่งอักขระควบคุมเทอร์มินอล

5.7.2 แมสเซจข้อมูลที่เพิกเฉย (Ignored Data Message)

byte *SSH_MSG_IGNORE*
string *data*

ทุกการอิมพลีเมนต์ต้องเข้าใจ (และเพิกเฉย) แมสเซจนี้ (หลังจากได้รับเวอร์ชันโพรโตคอล) ไม่มีการอิมพลีเมนต์ใดจำเป็นต้องส่งมัน

5.7.3 แมสเซจดีบัก (Debug Message)

byte *SSH_MSG_DEBUG*

boolean always_display

string message [RFC-2279]

string language tag [RFC-1766]

ทุกการอิมพลีเมนต์ต้องเข้าใจแมสเซจนี้ แต่อนุญาตให้เพิกเฉยต่อแมสเซจนี้ แมสเซจนี้ถูกใช้ในการส่งผ่านข้อมูลที่ใช้ช่วยในการดีบักแก่อีกฝั่ง ถ้า '*always_display*' เป็น TRUE '*message*' ควรถูกแสดง ในทางกลับกัน มันไม่ควรถูกแสดงออกมา เว้นแต่ผู้ใช้ได้ร้องขอข้อมูลดีบัก

'*message*' ไม่จำเป็นต้องมีอักขระ *newline* แต่อย่างไรก็ตาม อนุญาตให้ประกอบขึ้นจากหลายบรรทัดได้ ซึ่งแยกแต่ละบรรทัดด้วยคู่อักขระ CRLF (Carriage Return-Line Feed)

ถ้าสตริง '*message*' ถูกแสดงผล ควรใช้การฟิลเตอร์อักขระควบคุมเทอร์มินอลดังที่อธิบายไว้ในบทที่ 6 เพื่อหลีกเลี่ยงการโจมตีโดยการส่งอักขระควบคุมเทอร์มินอล

5.7.4 แมสเซจสงวน (Reserved Message)

การอิมพลีเมนต์ต้องตอบสนองต่อทุกแมสเซจที่ไม่สามารถแยกแยะเข้าใจได้ด้วยแมสเซจ SSH_MSG_UNIMPLEMENTED ตามลำดับแมสเซจที่ได้รับซึ่งแมสเซจเหล่านี้ต้องถูกเพิกเฉย เวอร์ชันโพรโตคอลใหม่อาจนิยามความหมายอื่นแก่ชนิดของแมสเซจเหล่านี้

byte SSH_MSG_UNIMPLEMENTED

uint32 packet sequence number of rejected message

5.8 สรุปหมายเลขแมสเซจ

หมายเลขแมสเซจต่อไปนี้ถูกนิยามไว้ในโพรโตคอลนี้

```
#define SSH_MSG_DISCONNECT          1
#define SSH_MSG_IGNORE              2
#define SSH_MSG_UNIMPLEMENTED       3
#define SSH_MSG_DEBUG               4
#define SSH_MSG_SERVICE_REQUEST     5
#define SSH_MSG_SERVICE_ACCEPT      6

#define SSH_MSG_KEXINIT             20
#define SSH_MSG_NEWKEYS             21
```

/* หมายเลข 30 ถึง 49 ใช้สำหรับแพ็กเก็ต kex วิธีการ kex ที่ต่างกันอาจใช้หมายเลขในช่วงนี้ใหม่ได้*/

```
#define SSH_MSG_KEXDH_INIT        30
#define SSH_MSG_KEXDH_REPLY       31
```

5.9 พิจารณาด้านความปลอดภัย

โพรโตคอลนี้ทำให้เกิดแชนเนลเข้ารหัสที่ปลอดภัย มันทำให้มีการพิสูจน์โฮสต์เซิร์ฟเวอร์ แลกเปลี่ยนคีย์ เข้ารหัส และปกป้องความสมบูรณ์ของข้อมูล มันยังให้ไอเดนติฟายเออร์ของเซสชันที่เป็นเอกลักษณ์ซึ่งอาจถูกนำไปใช้โดยโพรโตคอลชั้นที่สูงกว่า

บางครั้งโพรโตคอลนี้อาจจะถูกใช้โดยปราศจากการร้องขอยืนยันใช้ความสัมพันธ์ที่ไว้วางใจได้ของระหว่างคีย์โฮสต์เซิร์ฟเวอร์และชื่อโฮสต์เซิร์ฟเวอร์ การใช้เช่นนี้ทำให้ไม่แยกออกจากความไม่ปลอดภัยได้ แต่อาจจะจำเป็นในสภาพแวดล้อมที่ไม่ปลอดภัย และยังคงป้องกันการโจมตีแบบพาสซีฟ อย่างไรก็ตาม ผู้อิมพลีเมนต์โพรโตคอลที่ทำงานบนโพรโตคอลนี้ควรคำนึงถึงความเป็นไปได้นี้ไว้ด้วย

โพรโตคอลนี้ถูกออกแบบมาให้ใช้บนการลำเลียงที่น่าเชื่อถือ ถ้าการส่งมีความผิดพลาด การเชื่อมต่อต้องถูกปิดลง ถ้ากรณีนี้เกิดขึ้นการเชื่อมต่อควรถูกก่อตั้งขึ้นใหม่ การโจมตีเพื่อให้หยุดบริการ (denial of sevice) ในรูปแบบนี้จึงแทบจะเป็นไปไม่ได้ที่จะหลีกเลี่ยง

โพรโตคอลไม่ได้ถูกออกแบบมาให้ทำลายแชนเนลที่ลอบมอง ยกตัวอย่างเช่น แพดดิ้ง, แมสเซจ *SSH_MSG_IGNORE* และจุดอื่นอีกหลายจุด ในโพรโตคอลสามารถถูกใช้ในการส่งข้อมูลที่แอบแฝง และผู้รับไม่มีทางไว้ใจในการตรวจสอบว่าข้อมูลข่าวสารจำพวกนี้กำลังถูกส่งหรือไม่

แหล่งข้อมูลเพิ่มเติม

[FIPS-186] Federal Information Processing Standards Publication (FIPS PUB) 186, Digital Signature Standard, 18 May 1994.

[RFC-2459] Housley, R., et al: "Internet X.509 Public Key Infrastructure, Certificate and CRL Profile", January 1999.

[RFC-1766] Alvestrand, H: "Tags for the Identification of Languages", March 1995.

[RFC-1950] Deutch, P. and Gailly, J-L: "ZLIB Compressed Data Format Specification version 3.3", May 1996.

[RFC-1951] Deutch, P: "DEFLATE Compressed Data Format Specification version 1.3", May 1996.

[RFC-2279] Yergeau, F: "UTF-8, a transformation format of ISO 10646", January 1998.

[RFC-2104] Krawczyk, H., Bellare, M., and Canetti, R: "HMAC: Keyed-Hashing for Message Authentication", February 1997

[RFC-2144] Adams, C: "The CAST-128 Encryption Algorithm", May 1997.

[RFC-2440] Callas, J., et al: "OpenPGP Message Format", November 1998.

[RFC-2693] Ellison, C., et al: "SPKI Certificate Theory", September 1999.

[SCHNEIER] Schneier, B: "Applied Cryptography Second Edition: protocols, algorithms, and source code in C", 2nd edition, John Wiley & Sons, New York, NY, 1996.

บทที่ 6

โพรโตคอลการพิสูจน์ตนเอสเอสเอช 2

6.1 แนะนำ

โพรโตคอลการพิสูจน์ตนเอสเอสเอช 2 เป็นโพรโตคอลการพิสูจน์ตนผู้ใช้ที่ทำงานแบบจุดประสงค์ทั่วไป (general purpose) มุ่งหมายให้ทำงานบนโพรโตคอลชั้นทรานสปอร์ตเอสเอสเอช 2 โพรโตคอลนี้เป็นโพรโตคอลพื้นฐานที่ให้การป้องกันด้านความสมบูรณ์ (integrity) และด้านความเชื่อมั่น (confidentiality)

ชื่อบริการสำหรับโพรโตคอลนี้คือ "*ssh-userauth*"

เมื่อโพรโตคอลนี้เริ่มทำงาน มันจะรับไอเด็นติฟายเออร์ของเซสชันจากโพรโตคอลที่ต่ำกว่า (เป็นแฮชแลกเปลี่ยน H จากการแลกเปลี่ยนคีย์ครั้งแรก) เซสชันไอเด็นติฟายเออร์บ่งชี้เซสชันอย่างมีเอกลักษณ์และเหมาะต่อการลงลายเซ็น (signing) เพื่อใช้ในการพิสูจน์ความเป็นเจ้าของคีย์ส่วนตัว (private key) โพรโตคอลจึงจำเป็นต้องรู้ว่าโพรโตคอลที่ต่ำกว่าได้ให้การป้องกันด้านความมั่นใจหรือไม่

6.2 เฟรมเวิร์กโพรโตคอลการพิสูจน์ตน (The Authentication Protocol Framework)

เซิร์ฟเวอร์จะกระตุ้นให้เกิดการพิสูจน์ตน โดยบอกไคลเอ็นต์ว่ามีวิธีการพิสูจน์ตนวิธีใดบ้างที่สามารถใช้ในการแลกเปลี่ยน ไคลเอ็นต์มีอิสระในการเลือกวิธีในรวยการที่เซิร์ฟเวอร์ให้มา ทำให้เซิร์ฟเวอร์ควบคุมกระบวนการพิสูจน์ตนได้ถ้าต้องการ แต่ก็ให้ความยืดหยุ่นแก่ไคลเอ็นต์ในการเลือกใช้วิธีที่ไคลเอ็นต์รองรับหรือสะดวกใช้ที่สุดแก่ผู้ใช้ เมื่อเซิร์ฟเวอร์เสนอให้หลายวิธีการ

วิธีการพิสูจน์ตนถูกบ่งชี้ด้วยชื่อวิธี ตามที่ระบุในสถาปัตยกรรมโพรโตคอลเอสเอสเอช 2 วิธี "none" ถูกสงวนไว้ และต้องไม่อยู่ในลิสต์ที่รองรับ อย่างไรก็ตามอาจจะถูกส่งโดยไคลเอ็นต์ก็ได้ เซิร์ฟเวอร์ต้องปฏิเสธการร้องขอนี้เสมอ เว้นแต่ว่าไคลเอ็นต์ได้รับอนุญาตให้ไม่ต้องมีการพิสูจน์ตนใดๆ กรณีนี้เซิร์ฟเวอร์ต้องยอมรับการร้องขอ จุดประสงค์หลักของการส่งการร้องขอนี้คือเพื่อร้องขอลิสต์วิธีที่รองรับจากเซิร์ฟเวอร์

เซิร์ฟเวอร์ควรมีระยะหมดเวลาในการพิสูจน์ตน และหยุดการเชื่อมต่อ ถ้าการพิสูจน์ตนไม่ได้รับการยอมรับภายในระยะเวลาหมด แนะนำให้ระยะเวลาหมดเป็น 10 นาที ยิ่งไปกว่านั้นการอิมพลีเมนต์

ควรจำกัดจำนวนครั้งของความพยายามในการพิสูจน์ตนที่ล้มเหลวของไคลเอ็นต์ในเซสชันหนึ่งๆ (แนะนำว่าจำกัดไว้ที่ 20 ครั้ง) ถ้าความพยายามเกินที่จำกัดไว้ เซิร์ฟเวอร์ควรหยุดการเชื่อมต่อลง

6.2.1 การร้องขอพิสูจน์ตน (Authentication Requests)

ทุกการร้องขอพิสูจน์ตนต้องใช้รูปแบบดังนี้

byte *SSH_MSG_USERAUTH_REQUEST*
string *user name (in ISO-10646 UTF-8 encoding [RFC2279])*
string *service name (in US-ASCII)*
string *method name (US-ASCII)*
The rest of the packet is method-specific.

มีเพียงฟิลด์แรกๆที่ถูกนิยามไว้ ส่วนฟิลด์ที่เหลือขึ้นกับวิธีการพิสูจน์ตน

ชื่อผู้ใช้ *user name* และชื่อบริการ *service name* จะเหมือนเดิมทุกครั้งในความพยายามพิสูจน์ใหม่ และอาจเปลี่ยนได้ การอิมพลีเมนต์เซิร์ฟเวอร์ต้องตรวจสอบอย่างระมัดระวังเกี่ยวกับมันในทุกๆแมสเซจ และถ้ามันมีการเปลี่ยนเปลง เซิร์ฟเวอร์ต้องละทิ้งสเตทของการพิสูจน์ตนที่เก็บไว้ มันต้องหยุดการเชื่อมต่อถ้าชื่อผู้ใช้หรือบริการเปลี่ยนแปลง

ชื่อบริการระบุบริการที่จะสตาร์ตหลังจากการพิสูจน์ตน มีหลายบริการที่แตกต่างกันให้ใช้ ถ้าบริการที่ร้องขอไม่สามารถใช้ได้ เซิร์ฟเวอร์อาจหยุดการเชื่อมต่อทันทีหรือในภายหลังเวลาใดก็ได้ แนะนำให้ส่งแมสเซจหยุดการเชื่อมต่อในกรณีที่บริการที่ร้องขอไม่มี การพิสูจน์ตนต้องไม่ถูกยอมรับ

ถ้าไม่มีผู้ใช้ตามที่ร้องขอ เซิร์ฟเวอร์อาจจะหยุดการเชื่อมต่อ หรืออาจส่งลิสต์ปลอม (bogus list) ของวิธีการพิสูจน์ตนที่สามารถยอมรับได้ แต่จะไม่ยอมรับ ทำให้เซิร์ฟเวอร์หลีกเลี่ยงการเปิดเผยว่ามีรายชื่อผู้ใช้ใดบ้างที่มีอยู่จริง หากไม่มีผู้ใช้นั้นจริง การร้องขอพิสูจน์ตนต้องไม่ได้รับการยอมรับ

ไคลเอ็นต์สามารถส่งคำร้องขอที่เซิร์ฟเวอร์ไม่ได้ลงในลิสต์ยอมรับได้ การส่งคำร้องขอเช่นนี้ไม่ใช่ความผิดพลาด และเซิร์ฟเวอร์ควรปฏิเสธการร้องขอที่มันไม่รู้จัก

การร้องขอพิสูจน์ตนอาจให้ผลลัพธ์ในการแลกเปลี่ยนแมสเซจครั้งต่อไป แมสเซจทั้งหมดนี้จะขึ้นกับวิธีการพิสูจน์ตนที่ใช้ และไคลเอ็นต์อาจจะส่งแมสเซจ *SSH_MSG_USERAUTH_REQUEST* ใหม่ ณ.เวลาใดๆ ซึ่งในกรณีเช่นนี้ เซิร์ฟเวอร์ต้องยกเลิกความพยายามพิสูจน์ตนก่อนหน้า และทำการพิสูจน์ตนใหม่

6.2.2 การตอบสนองต่อการร้องขอพิสูจน์ตน (Responses to Authentication Requests)

ถ้าเซิร์ฟเวอร์ปฏิเสธการร้องขอพิสูจน์ตน มันต้องตอบสนองด้วยแมสเซจที่มีรูปแบบดังนี้

byte *SSH_MSG_USERAUTH_FAILURE*

string *authentications that can continue*

boolean partial success

โดย "*authentications that can continue*" เป็นลิสต์ชื่อวิธีการพิสูจน์ตนซึ่งแยกคั่นด้วยเครื่องหมายคอมม่า แนะนำว่าเซิร์ฟเวอร์ควรรวมเฉพาะวิธีที่มใช้ประโยชน์ได้จริงลงในลิสต์ อย่างไรก็ตาม ไม่ผิดกฎในการรวมวิธีการที่ไม่สามารถใช้ในการพิสูจน์ตนผู้ใช้

การพิสูจน์ตนที่จบลงด้วยความสำเร็จแล้ว ไม่ควรถูกรวมในลิสต์ เว้นแต่มันควรจะถูกปฏิบัติอีกครั้งด้วยเหตุผลบางข้อ

"*partial success*" ต้องเป็น TRUE ถ้าการร้องขอพิสูจน์ตนที่ได้รับการตอบสนองนี้ประสบผลสำเร็จ และต้องเป็น FALSE ถ้าการร้องขอทำงานแล้วไม่ประสบผลสำเร็จ

เมื่อเซิร์ฟเวอร์ยอมรับการพิสูจน์ตน มันต้องตอบสนองด้วยแมสเซจนี้

byte *SSH_MSG_USERAUTH_SUCCESS*

สังเกตว่าแมสเซจนี้จะไม่ถูกส่งหลังแต่ละขั้นตอนในลำดับการพิสูจน์ตนหลายวิธี แต่มันจะถูกส่งเพียงครั้งเดียวเมื่อการพิสูจน์ตนทั้งหมดสำเร็จ

ไคลเอ็นต์อาจส่งหลายการร้องขอพิสูจน์ตนโดยไม่ต้องรอการตอบสนองจากการร้องขอก่อนหน้า เซิร์ฟเวอร์ต้องแจ้งทุกการร้องขอที่ล้มเหลวด้วยแมสเซจ *SSH_MSG_USERAUTH_FAILURE* อย่างไรก็ตาม *SSH_MSG_USERAUTH_SUCCESS* ต้องถูกส่งอย่างน้อย 1 ครั้ง และเมื่อ *SSH_MSG_USERAUTH_SUCCESS* ถูกส่ง การร้องขอพิสูจน์ตนที่ได้รับภายหลังควรถูกเพิกเฉย

แมสเซจที่ไม่เกี่ยวกับการพิสูจน์ตนซึ่งส่งโดยไคลเอ็นต์ที่ได้รับแมสเซจ *SSH_MSG_USERAUTH_SUCCESS* แล้ว ต้องถูกส่งผ่านไปยังบริการที่ทำงานอยู่บนโพรโตคอลนี้ แมสเซจสามารถแยกแยะได้โดยหมายเลขแมสเซจที่จะอธิบายต่อไป

6.2.3 การร้องขอพิสูจน์ตนแบบ "none" (The "none" Authentication Request)

ไคลเอ็นต์อาจจะร้องขอลิสต์ของวิธีการพิสูจน์ตน และอาจจะใช้วิธีการพิสูจน์ตนแบบ "none"

ถ้าไม่มีความจำเป็นในการพิสูจน์ตนผู้ใช้ เซิร์ฟเวอร์ต้องส่ง *SSH_MSG_USERAUTH_SUCCE SS* กลับ ในทางตรงกันข้าม ถ้าจำเป็นต้องมีการพิสูจน์ตนผู้ใช้ เซิร์ฟเวอร์ต้องส่ง *SSH_MSG_USERAUTH_FAILURE* กลับและอาจส่งลิสต์ของวิธีการพิสูจน์ตนกลับด้วย

6.2.4 ความสำเร็จของการพิสูจน์ตนผู้ใช้ (Completion of User Authentication)

การพิสูจน์ตนจะจบลงเมื่อเซิร์ฟเวอร์ตอบสนองด้วย *SSH_MSG_USERAUTH_SUCCESS* ทุกแมสเซจที่เกี่ยวข้องกับการพิสูจน์ตนที่ได้รับหลังจากการส่งแมสเซจ *SSH_MSG_USERAUTH_SUCCESS* แล้ว ควรถูกเพิกเฉย

หลังจากเซิร์ฟเวอร์ส่ง *SSH_MSG_USERAUTH_SUCCESS* แล้ว เซิร์ฟเวอร์จะสตาร์ตบริการที่ร้องขอไว้

6.2.5 ข้อความแบนเนอร์ (Banner Message)

ในบางครั้ง การส่งข้อความเตือนก่อนทำการพิสูจน์ตนมีความสำคัญ เกี่ยวกับการคุ้มครองทางกฎหมาย เช่นในเครื่อง UNIX ปกติจะแสดงข้อความที่ได้จากไฟล์ /etc/issue หรือใช้ tcp wrappers หรือซอร์ฟแวร์ที่คล้ายกันนี้ ในการแสดงแบนเนอร์ก่อนจะแสดงพร้อมรับการล็อกอิน

เซิร์ฟเวอร์เอสเอสเอช 2 อาจส่งแมสเซจ *SSH_MSG_USERAUTH_BANNER* ในเวลาใดก็ได้ก่อนการพิสูจน์ตนจะประสบความสำเร็จ แมสเซจนี้ประกอบด้วยข้อความที่ถูกแสดงแก่ผู้ใช้ไคลเอ็นต์ก่อนจะมีความพยายามในการทำการพิสูจน์ตน รูปแบบของแมสเซจเป็นดังนี้

byte *SSH_MSG_USERAUTH_BANNER*
string *message (ISO-10646 UTF-8)*
string *language tag (as defined in [RFC1766])*

ไคลเอ็นต์ควรแสดงข้อความบนหน้าจอโดยดีฟอลต์ อย่างไรก็ตาม เนื่องจากแมสเซจนี้มักจะส่งให้ทุกๆครั้งที่มีความพยายามล็อกอิน และเนื่องจากบางซอร์ฟแวร์ไคลเอ็นต์จำเป็นต้องเปิดวินโดว์แยกตัวออกมาใหม่เพื่อแสดงคำเตือนนี้ ซอร์ฟแวร์ไคลเอ็นต์จึงอาจจะอนุญาตให้ผู้ใช้ดิสเอเบิล (disable) การแสดงแบนเนอร์ที่ได้จากเซิร์ฟเวอร์ได้ ฟิลด์ข้อความอาจประกอบขึ้นจากหลายบรรทัด

ถ้าสตริงข้อความถูกแสดง ควรใช้ฟิลเตอร์ตัวอักขระควบคุม (อธิบายในสถาปัตยกรรมโพรโตคอลเอสเอสเอช 2) เพื่อหลีกเลี่ยงการโจมตีโดยส่งตัวอักขระควบคุมเทอร์มินอล

6.3 หมายเลขแมสเซจของโพรโตคอลพิสูจน์ตน (Authentication Protocol Message Numbers)

หมายเลขแมสเซจทั้งหมดที่ใช้โดยโพรโตคอลพิสูจน์ตนอยู่ในช่วง 50 ถึง 79 ซึ่งเป็นส่วนหนึ่งในช่วงที่สงวนไว้ สำหรับโพรโตคอลที่ทำงานอยู่เหนือโพรโตคอลชั้นทรานสปอร์ตเอชเอชเอส

หมายเลขแมสเซจตั้งแต่ 80 ขึ้นไป ถูกสงวนไว้สำหรับโพรโตคอลที่ทำงานหลังจากโพรโตคอลพิสูจน์ตนนี้ ดังนั้นการได้รับแมสเซจหมายเลขเหล่านี้ ก่อนการพิสูจน์ตนจะสำเร็จ จะเป็นความผิดพลาด ซึ่งเซิร์ฟเวอร์ต้องตอบสนองด้วยการหยุดการเชื่อมต่อ (ควรส่งแมสเซจหยุดการเชื่อมต่อเป็นแมสเซจแรกเพื่อให้ง่ายต่อการแก้ไขปัญหา)

หลังจากการพิสูจน์ตนสำเร็จลง แมสเซจจะถูกส่งผ่านไปยังบริการชั้นที่สูงกว่า

#define SSH_MSG_USERAUTH_REQUEST 50

#define SSH_MSG_USERAUTH_FAILURE 51

#define SSH_MSG_USERAUTH_SUCCESS 52

#define SSH_MSG_USERAUTH_BANNER 53

หมายเลขแมสเซจในช่วง 60 ถึง 79 ถูกสงวนไว้สำหรับแมสเซจระบุลักษณะวิธีการ (method-specific messages) แมสเซจเหล่านี้ส่งได้โดยเซิร์ฟเวอร์เท่านั้น (ไคลเอ็นต์สามารถส่งได้เพียงแมสเซจ *SSH_ MSG_USERAUTH-REQUEST* เท่านั้น) วิธีการพิสูจน์ตนที่แตกต่างกันสามารถนำหมายเลขแมสเซจที่เหมือนกันมาใช้ใหม่ได้

6.4 วิธีการพิสูจน์ตนด้วยคีย์สาธารณะ : publickey (Public Key Authentication Method : publickey)

วิธีการพิสูจน์ตนที่จำเป็นต้องมีคือการพิสูจน์ตนแบบคีย์สาธารณะเท่านั้น ทุกการอิมพลีเมนต์ต้องรองรับวิธีนี้ อย่างไรก็ตาม ผู้ใช้ไม่ทุกคนที่ต้องการใช้คีย์สาธารณะ และนโยบายภายในมักไม่ต้องการใช้การพิสูจน์ตนแบบคีย์สาธารณะ

ด้วยวิธีการนี้ ความเป็นเจ้าของของคีย์ส่วนตัว (private key) ถูกนำมาใช้ในการพิสูจน์ตน วิธีการนี้ทำงานโดยการส่งลายเซ็นที่สร้างโดยคีย์ส่วนตัวของผู้ใช้ เซิร์ฟเวอร์ต้องตรวจสอบว่าคีย์เป็นตัวพิสูจน์ตนที่ถูกต้องสำหรับผู้ใช้นั้น และต้องตรวจสอบว่าลายเซ็นถูกต้อง ถ้าถูกทั้งสองกรณี การพิสูจน์ตนต้องถูกยอมรับ ในทางกลับกันจะต้องปฏิเสธ (สังเกตว่าเซิร์ฟเวอร์อาจจะต้องการการพิสูจน์ตนเพิ่มเติมหลังจากการพิสูจน์ตนสำเร็จแล้ว)

คีย์ส่วนตัวมักถูกเก็บไว้ในรูปแบบที่เข้ารหัสที่โฮสต์ไคลเอ็นต์ และผู้ใช้ต้องกรอกพาสเฟส (passphrase) ก่อน เพื่อให้ลายเซ็นสามารถถูกสร้างขึ้นได้ แม้ว่ามันจะไม่ได้ถูกเข้ารหัสไว้ กระบวการลงลายเซ็นก็ใช้การคำนวณมาก เพื่อหลีกเลี่ยงกระบวนการและการโต้ตอบกับผู้ใช้ที่ไม่จำเป็น แมสเซจข้างล่างนี้จะใช้ในการสืบค้นถาม (query) ว่าการใช้คีย์ในการพิสูจน์ตนนั้นถูกยอมรับหรือไม่

byte SSH_MSG_USERAUTH_REQUEST

string user name

string service

string "publickey"

boolean FALSE

string public key algorithm name

string public key blob

อัลกอริทึมคีย์สาธารณะใดๆ อาจถูกนำมาใช้ในการพิสูจน์ตน ถ้าเซิร์ฟเวอร์ไม่รองรับอัลกอริทึมมันต้องปฏิเสธการร้องขอนั้น

เซิร์ฟเวอร์ต้องตอบสนองต่อแมสเซจนี้ด้วย *SSH_MSG_USERAUTH_FAILURE* หรือด้วยแมสเซจข้างล่างนี้

byte *SSH_MSG_USERAUTH_PK_OK*
string *public key algorithm name from the request*
string *public key blob from the request*

เพื่อดำเนินการการพิสูจน์ตนอย่างแท้จริง ไคลเอ็นต์อาจส่งลายเซ็นที่สร้างโดยใช้คีย์ส่วนตัว ไคลเอ็นต์อาจส่งลายเซ็นโดยตรงก่อนโดยไม่มีการตรวจสอบก่อนว่าคีย์นั้นยอมรับได้หรือไม่ ลายเซ็นที่ส่งมีรูปแบบแพ็กเก็ตดังนี้

byte *SSH_MSG_USERAUTH_REQUEST*
string *user name*
string *service*
string *"publickey"*
boolean *TRUE*
string *public key algorithm name*
string *public key to be used for authentication*
string *signature*

'*signature*' เป็นลายเซ็นที่สอดคล้องกับคีย์สาธารณะบนข้อมูลที่ต่อไปนี้ ในลำดับที่แสดง

string *session identifier*
byte *SSH_MSG_USERAUTH_REQUEST*
string *user name*
string *service*
string *"publickey"*
boolean *TRUE*
string *public key algorithm name*
string *public key to be used for authentication*

เมื่อเซิร์ฟเวอร์ได้รับแมสเซจนี้ มันต้องตรวจสอบว่าคีย์ที่ให้มานี้ยอมรับได้สำหรับการพิสูจน์ตนหรือไม่ และถ้าได้ เซิร์ฟเวอร์ต้องตรวจสอบว่าลายเซ็นถูกต้องหรือไม่

ถ้าการตรวจสอบทั้งสองสำเร็จ วิธีการนี้จะประสบความสำเร็จ สังเกตว่าเซิร์ฟเวอร์อาจจะต้องการการพิสูจน์ตนเพิ่มเติม เซิร์ฟเวอร์ต้องตอบสนองด้วย *SSH_MSG_USERAUTH_SUCCESS* (ถ้าการพิสูจน์เพิ่มไม่มีความจำเป็น) หรือ *SSH_MSG_USERAUTH_FAILURE* (ถ้าการร้องขอล้มเหลว, หรือจำเป็นต้องมีการพิสูจน์ตนเพิ่ม)

หมายเลขแมสเซจกำหนดคุณลักษณะเฉพาะวิธีการ (method-specific message) ถูกใช้โดยวิธีการการพิสูจน์ตนแบบคีย์สาธารณะดังนี้

```
/* Key-based */
#define SSH_MSG_USERAUTH_PK_OK    60
```

6.5 วิธีการพิสูจน์ตนแบบรหัสผ่าน : password (Password Authentication Method : password)

การพิสูจน์ตนแบบรหัสผ่านใช้แพ็กเก็ตข้างล่างนี้ สังเกตว่าเซิร์ฟเวอร์อาจจะร้องขอผู้ใช้ให้เปลี่ยนรหัสผ่าน ทุกการิมพลีเมนต์ควรรองรับการพิสูจน์ตนแบบรหัสผ่าน

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "password"
boolean   FALSE
string    plaintext password (ISO-10646 UTF-8)
```

สังเกตว่ารหัสผ่านถูกเข้ารูปแบบแบบ ISO-10646 UTF-8 แล้วแต่เซิร์ฟเวอร์จะตีความหมายและตรวจสอบกับฐานข้อมูลรหัสผ่านอย่างไร อย่างไรก็ตาม ถ้าไคลเอ็นต์อ่านรหัสผ่านในรูปแบบอื่น (เช่น ISO 8859-1 (ISO Latin1)) มันต้องแปลงรหัสผ่านไปเป็น ISO-10646 UTF-8 ก่อนทำการส่ง และเซิร์ฟเวอร์ต้องแปลงรหัสผ่านไปเป็นรูปแบบที่ใช้สำหรับรหัสผ่านในระบบนั้น

สังเกตว่า แม้ว่ารหัสผ่านแบบข้อความที่อ่านได้ถูกส่งไปในแพ็กเก็ต แต่ทั้งแพ็กเก็ตก็ถูกเข้ารหัสโดยชั้นทรานสปอร์ต ทั้งเซิร์ฟเวอร์และไคลเอ็นต์ควรตรวจสอบว่าชั้นทรานสปอร์ตนั้นให้ความไว้วางใจ (confidentiality) ได้หรือไม่ (เช่น ใช้การเข้ารหัส) ถ้าไม่ให้ความไว้วางใจ (ไม่มีการเข้ารหัส), การพิสูจน์ตนแบบรหัสผ่านควรถูกยกเลิกใช้ และหากไม่ให้ความไว้วางใจหรือไม่มี MAC การเปลี่ยนรหัสผ่านควรถูกยกเลิกใช้

โดยปกติเซิร์ฟเวอร์ตอบสนองแก่แมสเซจนี้ด้วยความสำเร็จหรือล้มเหลว อย่างไรก็ตาม เซิร์ฟเวอร์อาจตอบสนองด้วย *SSH_MAG_USERAUTH_PASSWD_CHANGEREQ*

byte *SSH_MSG_USERAUTH_PASSWD_CHANGEREQ*

string *prompt (ISO-10646 UTF-8)*

string *language tag (as defined in [RFC1766])*

ในกรณีนี้ ไคลเอ็นต์ซอร์ฟแวร์ควรร้องขอรหัสผ่านใหม่จากผู้ใช้ และการส่งการร้องขอใหม่โดยใช้แมสเซจข้างล่างนี้ ไคลเอ็นต์อาจส่งแมสเซจนี้แทนการร้องขอการพิสูจน์ตนแบบรหัสผ่านปกติ โดยปราศจากการถามจากเซิร์ฟเวอร์

byte *SSH_MSG_USERAUTH_REQUEST*

string *user name*

string *service*

string *"password"*

boolean *TRUE*

string *plaintext old password (ISO-10646 UTF-8)*

string *plaintext new password (ISO-10646 UTF-8)*

เซิร์ฟเวอร์ต้องตอบกลับด้วย *SSH_MSG_USERAUTH_SUCCESS*, *SSH_MSG_USERAUTH_FAILURE* หรือ *SSH_MSG_USERAUTH_PASSWD_CHANGEREQ* ความหมายของแต่ละแมสเซจเป็นดังนี้

- *SSH_MSG_USERAUTH_SUCCESS* รหัสผ่านถูกเปลี่ยนแล้ว และการพิสูจน์ตนได้จบลงด้วยความสำเร็จ
- *SSH_MSG_USERAUTH_FAILURE with partial success* รหัสผ่านถูกเปลี่ยนแล้ว แต่การพิสูจน์ตนเพิ่มยังคงจำเป็น
- *SSH_MSG_USERAUTH_FAILURE without partial success* รหัสผ่านยังไม่ได้ถูกเปลี่ยน เกิดจากไม่รองรับการเปลี่ยนรหัสผ่าน หรือป้อนรหัสเก่าผิด สังเกตว่าถ้าเซิร์ฟเวอร์ได้ส่ง *SSH_MSG_USERAUTH_PASSWD_CHANGEREQ* แล้วเราจะรู้ว่ามันรองรับการเปลี่ยนรหัสผ่าน
- *SSH_MSG_USERAUTH_CHANGEREQ* รหัสผ่านไม่ได้ถูกเปลี่ยน เพราะไม่สามารถยอมรับรหัสใหม่ได้ (เช่น ง่ายต่อการคาดเดาเกินไป)

หมายเลขแมสเซจระบุลักษณะเฉพาะวิธีการ (method-specific) ข้างล่างนี้ ถูกใช้โดยวิธีการพิสูจน์ตนแบบรหัสผ่าน

#define SSH_MSG_USERAUTH_PASSWD_CHANGEREQ 60

6.6 การพิสูจน์ตนบนพื้นฐานของโฮสต์ (Host-Based Authentication : hostbased)

บางที่ประสงค์ที่จะอนุญาตให้ทำการพิสูจน์ตนบนพื้นฐานของโฮสต์ที่ผู้ใช้ได้ใช้เข้ามา และชื่อผู้ใช้บนเครื่องโฮสต์ทางไกล แม้ว่ารูปแบบการพิสูจน์ตนนี้ไม่เหมาะสมกับที่ที่ต้องการความปลอดภัยสูง แต่มันก็เหมาะสมกับในหลานสถานการณ์ รูปแบบการพิสูจน์นี้เป็นตัวเลือก (OPTIONAL) เมื่อถูกใช้ควรต้องระมัดระวังเป็นพิเศษในการป้องกันคีย์โฮสต์ส่วนตัว (private host key) จากผู้อื่น

ไคลเอ็นต์ร้องขอการพิสูจน์ตนรูปแบบนี้โดยส่งแมสเซจข้างล่างนี้ มันคล้ายกับรูปแบบการพิสูจน์ตน "rhost" และ "host.quiv" ใน UNIX เว้นแต่ไอเด็นติตี้ของโฮสต์ไคลเอ็นต์จะถูกตรวจสอบอย่างเข้มงวดมากกว่า

วิธีการนี้ทำงานโดยไคลเอ็นต์ส่งลายเซ็นที่สร้างด้วยคีย์ส่วนตัวของโฮสต์ไคลเอ็นต์ ซึ่งเซิร์ฟเวอร์จะสามารถตรวจสอบได้ด้วยคีย์สาธารณะของโฮสต์นั้น เมื่อไอเด็นติตี้ของโฮสต์ไคลเอ็นต์ถูกก่อตั้งขึ้นการให้เอกสิทธิ์ (แต่จะไม่มีการพิสูจน์ตนอีก) ถูกดำเนินการโดยตั้งอยู่บนพื้นฐานของชื่อผู้ใช้บนเซิร์ฟเวอร์และไคลเอ็นต์ และชื่อโฮสต์ไคลเอ็นต์

byte *SSH_MSG_USERAUTH_REQUEST*
string *user name*
string *service*
string *"hostbased"*
string *public key algorithm for host key*
string *public host key and certificates for client host*
string *client host name (FQDN; US-ASCII)*
string *client user name on the remote host (ISO-10646 UTF-8)*
string *signature*

ชื่ออัลกอริทึมคีย์สาธารณะที่ใช้ใน '*public key algorithm for host key*' ถูกนิยามในการกำหนดคุณลักษณะเฉพาะชั้นทรานสปอร์ต '*public host key for client host*' อาจรวมเซอร์ทิฟิเคท (certificate) ไปด้วย

'*signature*' เป็นลายเซ็นที่เซ็นด้วยคีย์โฮสต์ส่วนตัวของข้อมูลข้างล่างนี้ ในตามลำดับนี้

string *session identifier*
byte *SSH_MSG_USERAUTH_REQUEST*
string *user name*
string *service*
string *"hostbased"*

string *public key algorithm for host key*

string *public host key and certificates for client host*

string *client host name (FQDN; US-ASCII)*

string *client user name on the remote host (ISO-10646 UTF-8)*

เซิร์ฟเวอร์ต้องตรวจสอบว่าคีย์โฮสต์เป็นของโฮสต์ไคลเอ็นต์ที่มีชื่ออยู่ในแมสเซจ ซึ่งเป็นโฮสต์ไคลเอ็นต์ที่อนุญาตให้ผู้ใช้ล็อกอิน และลายเซ็นนั้นเป็นลายเซ็นที่ถูกต้องสอดคล้องกับคีย์โฮสต์ที่ให้ เซิร์ฟเวอร์อาจเพิกเฉยชื่อผู้ใช้บนไคลเอ็นต์ถ้าต้องการพิสูจน์ตนเฉพาะโฮสต์ไคลเอ็นต์

แนะนำว่าเมื่อใดก็ตามที่เป็นไปได้ เซิร์ฟเวอร์ดำเนินการตรวจสอบเพิ่มเพื่อตรวจสอบว่าแอดเดรสเครือข่าย (network address) ที่ได้จากเครือข่าย (ไม่น่าไว้ใจ) ตรงกับชื่อโฮสต์ไคลเอ็นต์ที่ได้ ทำให้การนำคีย์โฮสต์ที่ใช้ไม่ได้แล้วมาใช้ประโยชน์มีความยากมากขึ้น สังเกตว่ามันอาจต้องการการควบคุมเป็นพิเศษสำหรับการเชื่อมต่อที่เข้ามาผ่านไฟร์วอลล์ (firewall)

6.7 พิจารณาด้านความปลอดภัย

จุดประสงค์ของโพรโตคอลนี้คือเพื่อดำเนินการพิสูจน์ตนผู้ใช้บนฝั่งไคลเอ็นต์ มันจะถือว่าทำงานอยู่บนโพรโตคอลชั้นทรานสปอร์ตที่ปลอดภัยซึ่งได้ผ่านการพิสูจน์ตนกับเครื่องเซิร์ฟเวอร์ ก่อตั้งแชนเนลสื่อสารที่เข้ารหัส และคำนวณไอเด็นติฟายเออร์ของเซสชันที่เป็นเอกลักษณ์แก่เซสชันนี้ ชั้นทรานสปอร์ตให้การส่งผ่านที่เป็นความลับสำหรับการพิสูจน์ตนแบบรหัสผ่านและวิธีอื่นๆ ซึ่งเชื่อมั่นบนข้อมูลที่เป็นความลับ

เซิร์ฟเวอร์อาจเข้าสู่ช่วง "sleep" ได้ หลังจากการพิสูจน์ตนที่ไม่ประสบความสำเร็จติดต่อกันซ้ำๆ เพื่อให้การค้นหาคีย์ยากขึ้น

ถ้าชั้นทรานสปอร์ตไม่มีการเข้ารหัส วิธีการพิสูจน์ตนที่เชื่อมั่นบนข้อมูลที่เป็นความลับควรถูกยกเลิกการใช้งาน (disable) ถ้ามันไม่ให้การป้องกันด้วย MAC การร้องขอเปลี่ยนข้อมูลพิสูจน์ตน (เช่น การเปลี่ยนรหัสผ่าน) ควรถูกยกเลิกห้ามใช้งาน (disable) เพื่อหลีกเลี่ยงการโจมตีด้วยการเปลี่ยนแปลงข้อมูลที่เข้ารหัส (ciphertext) โดยที่เราไม่รู้ตัว การเรนเดอร์ (rendering) ข้อมูลพิสูจน์ตนใหม่จะไม่สามารถใช้ได้ (denial of service)

อนุญาตให้มีวิธีการพิสูจน์ตนหลายแบบหลายคุณลักษณะความปลอดภัยที่แตกต่างกันไป แล้วแต่นโยบายฝั่งโลคอลของแต่ละเซิร์ฟเวอร์ที่จะติดต่อด้วย ในการตัดสินใจว่าวิธีการใด (หรือใช้หลายวิธีการประกอบกัน) ที่มันจะยอมรับสำหรับแต่ละผู้ใช้

ควรให้ความใส่ใจเป็นพิเศษเมื่อทำการออกแบบข้อความดีบัก ถ้าการออกแบบไม่เหมาะสม แมสเซจเหล่านี้อาจจะเปิดเผยข้อมูลอย่างน่าตกใจเกี่ยวกับโฮสต์ ข้อความดีบักสามารถถูกยกเลิกห้ามใช้งาน (ระหว่างช่วงการพิสูจน์ตนผู้ใช้) เมื่อต้องการความปลอดภัยสูง

บทที่ 7

โพรโตคอลเชื่อมต่อเอสเอสเอช 2

7.1 แนะนำ

โพรโตคอลเชื่อมต่อถูกออกแบบมาให้ทำงานบนชั้นทรานสปอร์ต และโพรโตคอลการพิสูจน์ตนผู้ใช้ ทำให้เกิดเซสชันล็อกอินแบบอินเทอร์แอกทีฟ (interactive login sessions), การเอ็กซีคิวคำสั่งระยะไกล (remote execute of commands), การเชื่อมต่อทีซีพี/ไอพีแบบฟอร์เวิร์ด (forwarded TCP/IP connections) และการเชื่อมต่อ X11 แบบฟอร์เวิร์ด (forwarded X11 connections) ชื่อบริการสำหรับโพรโตคอลนี้ (หลังจากการพิสูจน์ตนผู้ใช้) เรียกว่า "ssh-connection"

7.2 การร้องขอทั่วไป (Global Requests)

การร้องขอหลายชนิดที่มีผลกระทบต่อสเตท (state) ของฝั่งรีโมท (remote end) โดยรวม จึงไม่ขึ้นกับแชนเนล ตัวอย่างเช่น การร้องขอเพื่อให้เริ่มทำการฟอร์เวิร์ดทีซีพี/ไอพีต่อพอร์ตที่ระบุ ทุกการร้องขอจะใช้รูปแบบ (format) ดังนี้

byte *SSH_MSG_GLOBAL_REQUEST*
string *request name (restricted to US-ASCII)*
boolean want reply
... request-specific data follows

เมื่อ '*want reply*' เป็น TRUE ผู้รับ (recipient) จะตอบสนองต่อแมสเซจ (message) นี้ด้วย *SSH_MSG_REQUEST_SUCCESS*, *SSH_MSG_REQUEST_FAILURE*, หรือ แมสเซจร้องการระบุต่อเนื่อง

byte *SSH_MSG_REQUEST_SUCCESS*

ถ้าผู้รับไม่สามารถแยกแยะหรือรองรับการร้องขอ จะตอบสนองด้วย *SSH_MSG_REQUEST_FAILURE*

byte *SSH_MSG_REQUEST_FAILURE*

7.3 กลไกของแชนเนล (Channel Mechanism)

ทุกเซสชันแบบเทอร์มินอล, การเชื่อมต่อแบบฟอร์เวิร์ด เป็นต้น เป็นแชนเนล ฝั่งใดฝั่งหนึ่งอาจเป็นผู้เปิดการเชื่อมต่อ หลายแชนเนลสามรถถูกมัลติเพลกซ์ (multiplex) เป็นการเชื่อมต่อเดียว (a single connection)

แชนเนลถูกบ่งชี้โดยหมายเลขที่แต่ละฝั่ง หมายเลขที่อ้างสำหรับหนึ่งแชนเนลที่แต่ละฝั่งอาจไม่เหมือนกัน การร้องขอเพื่อเปิดแชนเนลหนึ่งนั้นจะบรรจุหมายเลขแชนเนลของผู้ส่ง แมสเซจอื่นๆที่เกี่ยวข้องกับแชนเนลของผู้รับจะบรรจุหมายเลขแชนเนลของผู้รับสำหรับแชนเนลนั้นๆ

แชนเนลเป็นการควบคุมการไหล จะไม่มีข้อมูลถูกส่งไปยังแชนเนลจนกว่าจะได้แมสเซจที่บ่งบอกว่าเนื้อที่วินโดว์ (window space) ว่าง

7.3.1 การเปิดแชนเนล (Openning a Channel)

เมื่อฝ่ายหนึ่งฝ่ายหนึ่งประสงค์จะเปิดแชนเนลใหม่ จะต้องจองหมายเลขแชนเนลบนฝั่งตนไว้ จากนั้นส่งแมสเซจต่อไปนี้ไปยังอีกฝ่าย

byte SSH_MSG_CHANNEL_OPEN
string channel type (restricted to US-ASCII)
uint32 sender channel
uint32 initial window size
uint32 maximum packet size
... channel type specific data follows

แมสเซจนี้รวมหมายเลขแชนเนลบนฝั่งผู้ส่ง (local) ที่จองไว้ และขนาดวินโดว์เริ่มต้นไว้ในแมสเซจ (initial window size)

ชนิดของแชนเนลได้อธิบายไว้ในสถาปัตยกรรมของเอสเอสเอช 2 ด้วยกลไกขยายเพิ่มเติมที่คล้ายคลึงกัน

โดย

'*sender channel*' เป็นตัวบ่งชี้ฝั่งโลคอลสำหรับแชนเนลซึ่งถูกใช้โดยผู้ส่งแมสเซจนี้

'*initial window size*' ระบุจำนวนไบต์ของข้อมูลในแชนเนลที่สามารถส่งให้แก่ผู้ส่งแมสเซจนี้

'*maximum packet size*' ระบุถึงขนาดสูงสุดของแต่ละแพ็กเก็ตข้อมูล ที่สามารถส่งให้แก่ผู้ส่งแมสเซจนี้ (ตัวอย่างเช่น บางคนอาจจะต้องการใช้แพ็กเก็ตขนาดเล็กสำหรับการเชื่อมต่อแบบอินเทอร์แอ็กทีฟ เพื่อให้ได้การตอบสนองที่ดีขึ้น บนการเชื่อมต่อที่ช้า) จากนั้นฝั่งรีโมทจะตัดสินใจว่าจะสามารถเปิดแชนเนลได้หรือไม่ ถ้าเปิดได้จะตอบสนองด้วย

byte SSH_MSG_CHANNEL_OPEN_CONFIRMATION

uint32 recipient channel

uint32 sender channel

uint32 initial window size

uint32 maximum packet size

... channel type specific data follows

โดย

'recipient channel' เป็นหมายเลขแชนเนลเดิมที่มากับการร้องขอเปิดแชนเนล

'sender channel' เป็นหมายเลขแชนเนลที่จองโดยฝั่งรีโมท

หากถ้าไม่สามารถเปิดแชนเนลได้ ฝั่งรีโมทจะตอบสนองด้วย

byte SSH_MSG_CHANNEL_OPEN_FAILURE

uint32 recipient channel

uint32 reason code

string additional textual information (ISO-10646 UTF-8 [RFC2279])

string language tag (as defined in [RFC1766])

ถ้าผู้รับแมสเซจ *SSH_MSG_CHANNEL_OPEN* ไม่รองรับชนิดแชนเนลที่ระบุไว้ มันจะตอบสนองด้วย *SSH_MSG_CHANNEL_OPEN_FAILURE* ไคลเอ็นต์อาจจะแสดงข่าวสารเพิ่มเติมนี้แก่ผู้ใช้ และถ้าเกิดขึ้น ซอร์ฟแวร์ไคลเอ็นต์ควรจะป้องกันตามที่อธิบายไว้ในบทที่ 6

รหัสเหตุผล (reason code) ถูกนิยามดังนี้

#define SSH_OPEN_ADMINISTRATIVELY_PROHIBITED 1

#define SSH_OPEN_CONNECT_FAILED 2

#define SSH_OPEN_UNKNOWN_CHANNEL_TYPE 3

#define SSH_OPEN_RESOURCE_SHORTAGE 4

7.3.2 การถ่ายโอนข้อมูล (Data Transfer)

ขนาดวินโดว์บ่งบอกถึงจำนวนไบต์ที่อีกฝั่งสามารถส่งเพื่อไม่ให้มันต้องรอให้วินโดว์ถูกปรับเปลี่ยน ทั้ง2ฝ่ายใช้แมสเซจต่อไปนี้ในการปรับเปลี่ยนวินโดว์

byte SSH_MSG_CHANNEL_WINDOW_ADJUST

uint32 recipient channel

uint32 bytes to add

หลังจากผู้รับได้แมสเซจนี้แล้ว อาจจะส่งจำนวนไบต์มากกว่าที่เคยได้รับอนุญาต ขนาดวินโดว์จะถูกเพิ่มขึ้น

การถ่ายโอนข้อมูลถูกกระทำด้วยแมสเซจชนิดนี้

byte SSH_MSG_CHANNEL_DATA

uint32 recipient channel

string data

ขนาดสูงสุดของส่วนข้อมูล '*data*' ที่อนุญาตคือขนาดวินโดว์ปัจจุบัน ขนาดวินโดว์ลดลงโดยข้อมูลที่ส่งมา ทั้งสองฝ่ายอาจจะเพิกเฉยข้อมูลพิเศษที่ส่งหลังจากวินโดว์ที่อนุญาตไว้ว่างเปล่า

ยิ่งไปกว่านั้น บางแชนเนลสามารถถ่ายโอนข้อมูลหลายชนิดได้ ตัวอย่างคือ stderr data จากเซสชันอินเทอร์แอกทีฟ บางข้อมูลสามารถถูกส่งผ่านด้วยแมสเซจ *SSH_MSG_CHANNEL_EXTENDED_DATA* โดยใช้จำนวนเต็มเป็นตัวระบุชนิดข้อมูล ชนิดและความหมายขึ้นกับชนิดของแชนเนล

byte SSH_MSG_CHANNEL_EXTENDED_DATA

uint32 recipient_channel

uint32 data_type_code

string data

ข้อมูลที่ส่งด้วยแมสเซจนี้ จะใช้วินโดว์เดียวกับที่ใช้กับข้อมูลปกติ

ในขณะนี้มีเพียงชนิดเดียวที่นิยามไว้

#define SSH_EXTENDED_DATA_STDERR 1

7.3.3 การปิดแชนเนล (Closing a Channel)

เมื่อฝ่ายใดจะไม่ส่งข้อมูลแก่แชนเนลอีกต่อไป มันควรส่ง *SSH_MSG_CHANNEL_EOF* มีรูปแบบดังนี้

byte SSH_MSG_CHANNEL_EOF

uint32 recipient_channel

ไม่มีการตอบสนองโดยตรงแก่แมสเซจนี้ อย่างไรก็ตาม แอพพลิเคชั่นอาจส่ง EOF ให้แก่อะไรก็ตามที่อยู่อีกฝั่งของแชนเนล สังเกตว่าแชนเนลยังคงเปิดอยู่หลังส่งแมสเซจนี้ และข้อมูลยังคงสามารถส่งกลับมาได้ แมสเซจนี้ไม่ใช้เนื้อที่วินโดว์และสามารถส่งแม้ไม่มีเนื้อที่วินโดว์เหลือก็ตาม

เมื่อฝ่ายใดฝ่ายหนึ่งประสงค์จะยุติ (terminate) แชนเนล มันจะส่ง *SSH_MSG_CHANNEL_CLOSE* เมื่อได้รับแมสเซจนี้ ผู้รับต้องส่ง *SSH_MSG_CHANNEL_CLOSE* กลับไป เว้นแต่ว่าเป็นผู้ส่งแมสเซจสำหรับแชนเนลนี้ไป แชนเนลจะถือว่าปิดสำหรับแต่ละฝ่ายเมื่อได้ส่งและได้รับ *SSH_MSG_CHANNEL _CLOSE* แล้ว จากนั้นแล้วสามารถนำหมายเลขแชนเนลกลับมาใช้ใหม่ได้ แต่ละฝ่ายสามารถส่ง *SSH_ MSG_CHANNEL_CLOSE* ได้โดยปราศจากการส่ง *SSH_MSG_CHANNEL_EOF*

รูปแบบของแมสเซจเป็นดังนี้

byte *SSH_MSG_CHANNEL_CLOSE*
uint32 *recipient_channel*

แมสเซจนี้ไม่ใช้เนื้อที่วินโดว์ และสามารถส่งแม้ไม่มีเนื้อที่วินโดว์เหลือก็ตาม
แนะนำว่าถ้าเป็นไปได้ ทุกข้อมูลที่ส่งก่อนแมสเซจนี้ถูกนำส่งไปยังจุดหมายได้อย่างแท้จริง

7.3.4 การร้องขอกำหนดลักษณะเฉพาะของแชนเนล (Channel-Specific Requests)

หลายชนิดของแชนเนลมีส่วนขยายซึ่งเป็นลักษณะเฉพาะของชนิดแชนเนลพิเศษ ตัวอย่างเช่น การร้องขอเทอร์มินอลเทียม pty (pseudo terminal) สำหรับแชนเนลอินเทอร์แอ็กทีฟ

ทุกการร้องขอกำหนดลักษณะเฉพาะของแชนเนลใช้รูปแบบดังนี้

byte *SSH_MSG_CHANNEL_REQUEST*
uint32 *recipient channel*
string *request type (restricted to US-ASCII)*
boolean *want reply*
... type-specific data

ถ้า '*want reply*' เป็น FALSE จะไม่มีการตอบสนองต่อการร้องขอ ในทางตรงกันข้าม ผู้รับจะตอบสนองด้วย *SSH_MSG_CHANNEL_SUCCESS* หรือ *SSH_MSG_CHANNEL_FAILURE* หรือแมสเซจร้องขอกำหนดคุณลักษณะเฉพาะต่อเนื่อง ถ้าผู้รับแมสเซจ *SSH_MSG_CHANNEL_OPEN* ไม่รองรับชนิดแชนเนลที่ระบุไว้ มันจะตอบสนองด้วย *SSH_MSG_CHANNEL_OPEN_FAILURE*

แมสเซจนี้ไม่ใช้เนื้อที่วินโดว์ และสามารถส่งแม้ไม่มีเนื้อที่วินโดว์เหลือก็ตาม '*request type*' เป็นโลคอลสำหรับแต่ละชนิดแชนเนล

ไคลเอ็นต์จะได้รับอนุญาตให้ส่งแมสเซจต่อไป โดยไม่ต้องรอคอยการตอบสนองของการร้องขอนั้น

byte *SSH_MSG_CHANNEL_SUCCESS*
uint32 *recipient_channel*

byte *SSH_MSG_CHANNEL_FAILURE*
uint32 *recipient_channel*

แมสเซจทั้งสองนี้ไม่ใช้เนื้อที่วินโดว์ และสามารถส่งแม้ไม่มีเนื้อที่วินโดว์เหลือก็ตาม

7.4 เซสชันแบบอินเทอร์แอ็กทีฟ (Interactive Sessions)

เซสชันคือการเอ็กซีคิว (execution) โปรแกรมระยะไกล โปรแกรมอาจจะเป็นเชลล์ (shell), แอพพลิเคชัน, คำสั่งระบบ หรือระบบย่อยสำเร็จ (built-in subsystem) อาจจะมีหรือไม่มี tty ก็ได้ และอาจจะรวมถึงหรือไม่รวมถึงการฟอร์เวิร์ด X11 (X11 forwarding) เซสชันทั้งหมดสามารถทำงานด้วยตัวมันเองพร้อมกัน

7.4.1 การเปิดเซสชัน (Opening a Session)

เซสชันจะถูกเริ่มต้นโดยการส่งแมสเซจดังนี้

byte *SSH_MSG_CHANNEL_OPEN*
string *"session"*
uint32 *sender channel*
uint32 *initial window size*
uint32 *maximum packet size*

การอิมพลีเมนต์ ไคลเอ็นต์ควรปฏิเสธทุกการร้องขอเปิดแชนเนลเซสชัน (session channel) เพื่อให้ยากขึ้นสำหรับเซิร์ฟเวอร์ที่ประสงค์ไม่ดี จะมาโจมตีไคลเอ็นต์

7.4.2 การร้องขอเทอร์มินอลเทียม (Requesting a Pseudo-Terminal)

เทอร์มินอลเทียมสามารถถูกจัดสรรสำหรับเซสชันได้โดยการส่งแมสเซจนี้

byte *SSH_MSG_CHANNEL_REQUEST*
uint32 *recipient_channel*

string *"pty-req"*

boolean *want_reply*

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

uint32 *terminal width, characters (e.g., 80)*

uint32 *terminal height, rows (e.g., 24)*

uint32 *terminal width, pixels (e.g., 640)*

uint32 *terminal height, pixels (e.g., 480)*

string *encoded terminal modes*

การเข้ารหัสโหมดเทอร์มินอล '*encoded terminal modes*' ถูกอธิบายในหัวข้อ การเข้ารหัสของโหมดเทอร์มินอล (Encoding of Terminal Modes) ที่จะได้อธิบายต่อไปภายหลัง

พารามิเตอร์มิติ (dimension) ที่มีค่าเป็น 0 ต้องถูกเพิกเฉย เลือกใช้มิติอักขระ/แถว ก่อนเลือกใช้พิกเซล (เมื่อไม่เป็น 0) มิติในรูปพิกเซลบอกถึงพื้นที่ของวินโดว์ที่สามารถวาดได้ พารามิเตอร์มิติเป็นเพียงสนเทศ

ไคลเอ็นต์ควรจะเพิกเฉยต่อการร้องขอ pty

7.4.3 การฟอร์เวิร์ด X11 (X11 Forwarding)

7.4.3.1 การร้องขอฟอร์เวิร์ด X11 (Requesting X11 Forwarding)

การฟอร์เวิร์ด X11 อาจจะถูกร้องขอแก่เซสชันโดยการส่งแมสเซจดังนี้

byte *SSH_MSG_CHANNEL_REQUEST*

uint32 *recipient channel*

string *"x11-req"*

boolean *want reply*

boolean *single connection*

string *x11 authentication protocol*

string *x11 authentication cookie*

uint32 *x11 screen number*

แนะนำให้คุกกี้พิสูจน์ตน (authentication cookie) ที่ถูกส่งเป็นของปลอม (fake) ได้จากการสุ่ม และคุกกี้นี้จะถูกตรวจสอบและแทนด้วยคุ้กกี้จริงเมื่อการร้องขอการเชื่อมต่อได้รับแล้ว

การฟอร์เวิร์ดการเชื่อมต่อ X11 ควรหยุดเมื่อแชนเนลของเซสชันถูกปิดลง อย่างไรก็ตาม เมื่อแชนเนลของเซสชันถูกปิดลง การฟอร์เวิร์ดที่เปิดไว้แล้วควรจะไม่ถูกปิดอัตโนมัติ

ถ้า '*single connection*' เป็น TRUE จะมีการเชื่อมต่อเดียวที่ถูกฟอร์เวิร์ด จะไม่มีการเชื่อมต่ออื่นถูกฟอร์เวิร์ดอีกหลังจากการเชื่อมต่อแรกได้ฟอร์เวิร์ด หรือหลังจากแชนเนลของเซสชันได้ปิดลง

'*x11 authentication protocol*' เป็นชื่อของวิธีการพิสูจน์ตน X11เช่น"*MIT-MAGIC-COOKIE-1*" โพรโตคอลของ X อธิบายในเอกสารของ [SCHEIFLER]

7.4.3.2 แชนเนล X11 (X11 Channels)

แชนเนล X11 เปิดได้โดยการร้องขอเปิดแชนเนล แชนเนลที่ได้จะเป็นอิสระจากเซสชัน และการปิดแชนเนลของเซสชันจะไม่ปิดแชนเนล X11 ที่ฟอร์เวิร์ดแล้ว

byte SSH_MSG_CHANNEL_OPEN
string "x11"
uint32 sender channel
uint32 initial window size
uint32 maximum packet size
string originator address (e.g. "192.168.7.38")
uint32 originator port

ผู้รับควรตอบสนองด้วย *SSH_MSG_CHANNEL_OPEN_CONFIRMATION* หรือ *SSH_MSG_CHANNEL_OPEN_FAILURE*

การอิมพลีเมนต์ต้องปฏิเสธทุกการร้องขอเปิดแชนเนล X11 ถ้าไม่ร้องขอทำการฟอร์เวิร์ด X11 ก่อน

7.4.4 การผ่านค่าตัวแปรสภาพแวดล้อม (Environment Variable Passing)

ตัวแปรแวดล้อมอาจจะถูกส่งผ่านไปยัง เชลล์/คอมมานด์ (shell/command) เพื่อสตาร์ตในภายหลัง โดยปกติแต่ละเครื่องจะมีชุดกำหนดค่าเริ่มของตัวแปร (a preconfigured set of variables) เนื่องจากการตั้งค่าตัวแปรแวดล้อมโดยไม่ควบคุมเป็นอันตรายอย่างยิ่ง แนะนำว่าการอิมพลีเมนต์อนุญาตให้ตั้งค่าเพียงตัวแปรซึ่งมีชื่อบ่งบอกชัดเจนว่าอนุญาตให้ปรับแต่งได้

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "env"
boolean want reply
string variable name
string variable value

7.4.5 การสตาร์ตเชลล์หรือคอมมานด์ (Starting a Shell or a Command)

เมื่อเซสชันได้ถูกติดตั้งแล้ว โปรแกรมจะถูกสตาร์ตที่ฝั่งรีโมท โปรแกรมสามารถเป็นเชลล์, แอพพลิเคชั่น หรือระบบย่อยกับชื่อที่อิสระจากโฮสต์ (a subsystem with a host-independent name) มีเพียงหนึ่งในการร้องขอเหล่านี้ที่ทำได้สำเร็จต่อแชนเนล

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "shell"
boolean want reply

แมสเซจนี้จะร้องขอเชลล์ที่เป็นดีฟอลต์ของผู้ใช้ (ในระบบ UNIX โดยทั่วไปจะนิยามไว้ใน /etc/passwd) เพื่อสตาร์ตที่อีกฝั่ง

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "exec"
boolean want reply
string command

แมสเซจนี้จะร้องขอให้เซิร์ฟเวอร์สตาร์ตการเอ็กซีคิวคอมมานด์ '*command*' ที่ให้ สตริงคำสั่งอาจจะมีพาท (path) ต้องทำการป้องกันการเอ็กซีคิวคำสั่งที่ละเมิดสิทธิ์

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "subsystem"
boolean want reply
string subsystem name

รูปแบบสุดท้ายนี้ใช้เอ็กซีคิวระบบย่อยที่นิยามไ ซึ่งประกอบด้วยกลไกถ่ายโอนแฟ้มข้อมูลและอื่นๆไว้ การอิมพลีเมนต์อาจจะอนุญาตให้ปรับแต่งกล ต่างๆด้ย แนะนำให้โพรโตคอลระบบย่อยมีเมจิกคุ้กกี้ (magic cookie) ตั้งแต่เริ่มต้นการทรานแซ ชั่นของโพรโตคอล เพื่อให้สามารถแยกแยะจากเอาท์พุทของสคริปต์เริ่มต้นเชลล์ได้ เอาท์พุทจากเชลล์อาจจะถูกกรองออกที่เซิร์ฟเวอร์หรือไคลเอ็นต์

เซิร์ฟเวอร์ควรจะไม่หยุด (halt) การเอ็กซีคิวของสแตคโพรโตคอลในขณะที่สตาร์ตเชลล์หรือโปรแกรม อินพุทและเอาท์พุททั้งหมดควรถูกเปลี่ยนทิศทางใหม่ (redirect) ไปยังแชนเนลหรือไปยังทันเนล (tunnel) ที่เข้ารหัส

สำหรับแมสเซจเหล่านี้แนะนำว่าให้ร้องขอและตรวจสอบการตอบกลับ ไคลเอ็นต์ควรจะเพิกเฉยต่อแมสเซจเหล่านี้

7.4.6 การถ่ายโอนข้อมูลบนเซสชัน (Session Data Transfer)

การถ่ายโอนข้อมูลสำหรับเซสชันถูกทำโดยใช้แพ็กเก็ต *SHH_MSG_CHANNEL_DATA* และแพ็กเก็ต *SSH_MSG_CHANNEL_EXTENED_DATA* และกลไกวินโดว์ ชนิดข้อมูลส่วนขยาย *SSH_EXTENED_DATA_STDERR* ถูกนิยามไว้สำหรับข้อมูล stderr

7.4.7 แมสเซจเปลี่ยนขนาดของวินโดว์ (Window Dimension Change Message)

เมื่อขนาดวินโดว์ (เทอร์มินอล) เปลี่ยนบนฝั่งไคลเอ็นต์ มันอาจจะส่งแมสเซจไปยังอีกฝั่งเพื่อแจ้งขนาดใหม่ รูปแบบแมสเซจเป็นดังนี้

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient_channel
string "window-change"
boolean FALSE
uint32 terminal width, columns
uint32 terminal height, rows
uint32 terminal width, pixels
uint32 terminal height, pixels

ควรไม่ส่งการตอบสนองต่อแมสเซจนี้

7.4.8 โฟลว์คอนโทรลฝั่งโลคอล (Local Flow Control)

ในหลายๆระบบ เป็นไปได้ที่จะกำหนดว่าเทอร์มินอลใช้โฟลวคอนโทรล control-S/control-Q หากได้รับอนุญาตให้ใช้โฟลวคอนโทรลได้ บ่อยครั้งที่มีความต้องการใช้โฟลวคอนโทรลที่ฝั่งไคลเอ็นต์เพื่อเพิ่มความเร็วของการตอบสนองต่อการร้องขอของผู้ใช้ มันถูกทำให้สะดวกโดยมีขั้นตอนคือ เริ่มต้นด้วยเซิร์ฟเวอร์จะรับผิดชอบเรื่องโฟลวคอนโทรล (ต่อไปให้เข้าใจว่า ไคลเอ็นต์หมายถึงฝั่งที่เริ่มก่อการสร้างเซสชัน และเซิร์ฟเวอร์หมายถึงอีกฝั่ง)

แมสเซจข้างล่างนี้ถูกใช้โดยเซิร์ฟเวอร์ เพื่อแจ้งแก่ไคลเอ็นต์ว่าเมื่อใดไคลเอ็นต์สามารถใช้หรือไม่สามารถใช้โฟลวคอนโทรลได้ ถ้า '*client can do*' เป็น TRUE ไคลเอ็นต์จะได้รับอนุญาตให้ทำโฟลวคอนโทรลโดยใช้ control-S และ control-Q ไคลเอ็นต์อาจเพิกเฉยต่อแมสเซจนี้

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "xon-xoff"
boolean FALSE
boolean client can do

จะไม่มีการส่งแมสเซจตอบสนองต่อแมสเซจนี้

7.4.9 ซิกแนล (Signals)

ซิกแนลสามารถถูกส่งไปยังกระบวนการหรือบริการบนฝั่งรีโมท (remote process/service) โดยใช้แมสเซจดังนี้

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "signal"
boolean FALSE
string signal name without the "SIG" prefix.

บางระบบอาจจะไม่อิมพลีเมนต์ซิกแนล ในกรณีเช่นนี้ควรเพิกเฉยต่อแมสเซจนี้
ชื่อซิกแนล '*signal name*' อยู่ในรูปแบบดังนี้

ABRT	*ALRM*	*FPE*	*HUP*	*ILL*
INT	*KILL*	*PIPE*	*QUIT*	*SEGV*
TERM	*USR1*	*USR2*		

ชื่อซิกแนลเพิ่มเติมสามารถส่งในรูปแบบ "sig-name@xyz" โดย 'sig-name' และ 'xyz' อาจเป็นอะไรก็ได้ที่ผู้อิมพลีเมนต์ต้องการ (ยกเว้นเครื่องหมาย '@') อย่างไรก็ตามถ้าใช้สคริปต์ 'configure' ชื่อซิกแนลที่ไม่เป็นมาตรฐานจะพบว่าอยู่ในรูปแบบ "SIG@xyz.config.guess" โดย 'SIG' เป็นชื่อซิกแนลซึ่งไม่มี 'SIG' นำหน้า และ 'xyz' เป็นชนิดโฮสต์ ตามที่กำหนดไว้ใน 'config.guess'

7.4.10 การคืนค่าสถานะจบ (Returning Exit Status)

เมื่อคำสั่งที่ทำงานบนฝั่งตรงข้ามจบลง แมสเซจต่อไปนี้ใช้ส่งเพื่อคืนค่าการจบของคำสั่ง แนะนำว่าการคืนค่าสถานะจะไม่มีการแจ้งการได้รับ (acknowledgement) แมสเซจนี้ แชนเนลจำเป็นต้องปิดด้วย *SSH_MSG_CHANNEL_CLOSE* หลังแมสเซจนี้ และไคลเอ็นต์อาจจะเพิกเฉยแมสเซจนี้

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient_channel
string "exit-status"
boolean FALSE
uint32 exit_status

คำสั่งที่ทำงานบนรีโมทอาจจบลงทันทีด้วยแมสเซจนี้
ในบางสภาพการณ์ อาจใช้แมสเซจข้างล่างนี้ ซึ่งค่าที่คืนเมื่อจบการทำงานด้วยความสำเร็จนั้นโดยปกติจะให้เป็นศูนย์

byte SSH_MSG_CHANNEL_REQUEST
uint32 recipient channel
string "exit-signal"
boolean FALSE
string signal name without the "SIG" prefix.
boolean core dumped
string error message (ISO-10646 UTF-8)
string language tag (as defined in [RFC1766])

'*error message*' บรรจุคำอธิบายเพิ่มเติมของข้อความผิดพลาด ข้อความนี้อาจประกอบด้วยหลายบรรทัด ซอฟต์แวร์ไคลเอ็นต์อาจจะแสดงข้อความนี้แก่ผู้ใช้ หากเคยเกิดขึ้นแล้ว ซอร์ฟแวร์ไคลเอ็นต์ควรจะมีการป้องกันเช่นอธิบายไว้ในบทที่ 6

7.5 การฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี (TCP/IP Port Forwarding)

7.5.1 การร้องขอฟอร์เวิร์ดพอร์ต (Requesting Port Forwarding)

ไม่จำเป็นที่จะร้องขอทำการฟอร์เวิร์ดจากฝั่งของตนไปยังทิศทางอื่นๆ อย่างไรก็ตาม หากต้องการให้การเชื่อมต่อไปยังอีกฝั่งถูกฟอร์เวิร์ดไปยังฝั่งโลคอล มันต้องมีการร้องขอดังนี้

byte *SSH_MSG_GLOBAL_REQUEST*

string *"tcpip-forward"*

boolean *want reply*

string *address to bind (e.g. "0.0.0.0")*

uint32 *port number to bind*

'*Address to bind*' และ '*port number to bind*' ระบุถึงแอดเดรสแบบไอพี (IP Address) และพอร์ตที่ซ็อกเก็ตจับจองเพื่อใช้คอยฟัง แอดเดรสควรเป็น "0.0.0.0" ถ้าอนุญาตให้มีการเชื่อมต่อจากที่ใดก็ได้ (สังเกตว่า ไคลเอ็นต์ยังคงสามารถคัดสรรการเชื่อมต่อบนพื้นฐานของข้อมูลที่ส่งผ่านมากับการร้องขอเปิด)

การอิมพลีเมนต์ควรอนุญาตให้ทำการฟอร์เวิร์ดพอร์ตที่มีสิทธิ์ก็ต่อเมื่อผู้ใช้ได้ผ่านการพิสูจน์ตนว่าเป็นผู้ใช้ที่มีสิทธิ์แล้ว

การอิมพลีเมนต์ไคลเอ็นต์ควรปฏิเสธแมสเซจเหล่านี้เพราะโดยปกติแล้วมันจะถูกส่งมาจากไคลเอ็นต์เท่านั้น

การฟอร์เวิร์ดพอร์ตสามารถยกเลิกได้ด้วยแมสเซจข้างล่างนี้ สังเกตว่า การร้องขอเปิดแชนเนลอาจได้รับจนกระทั่งการตอบกลับต่อแมสเซจนี้ได้รับแล้ว

byte *SSH_MSG_GLOBAL_REQUEST*

string *"cancel-tcpip-forward"*

boolean *want reply*

string *address_to_bind (e.g. "127.0.0.1")*

uint32 *port number to bind*

การอิมพลีเมนต์ไคลเอ็นต์ควรปฏิเสธแมสเซจเหล่านี้เพราะโดยปกติแล้วมันจะถูกส่งมาจากไคลเอ็นต์เท่านั้น

7.5.2 แชนเนลฟอร์เวิร์ดทีซีพี/ไอพี (TCP/IP Forwarding Channels)

เมื่อมีการเชื่อมต่อมาที่พอร์ตซึ่งมีการร้องขอให้ทำการฟอร์เวิร์ดแล้ว แชนเนลจะถูกเปิดเพื่อฟอร์เวิร์ดพอร์ตไปยังอีกฝั่ง

byte *SSH_MSG_CHANNEL_OPEN*

string *"forwarded-tcpip"*

uint32 *sender channel*

uint32 *initial window size*

uint32 *maximum packet size*

string *address that was connected*

uint32 *port that was connected*

string *originator IP address*

uint32 *originator port*

การอิมพลีเมนต์ควรปฏิเสธแมสเซจนี้ เว้นแต่ว่าก่อนหน้านี้ได้ร้องขอทำการฟอร์เวิร์ดพอร์ตทีซีพี/ไอพีที่พอร์ตที่ต้องการแล้ว

เมื่อการเชื่อมต่อมาถึงที่พอร์ตทีซีพี/ไอพีที่ฟอร์เวิร์ดบนโลคอล แพ็กเก็ตต่อไปนี้จะถูกส่งไปยังฝั่งตรงข้าม สังเกตว่าแมสเซจเหล่านี้อาจถูกส่งแก่พอร์ตซึ่งไม่เคยร้องขอทำการฟอร์เวิร์ด ฝั่งผู้รับต้องตัดสินใจว่าจะอนุญาตให้ทำการฟอร์เวิร์ดหรือไม่

byte *SSH_MSG_CHANNEL_OPEN*

string *"direct-tcpip"*

uint32 *sender channel*

uint32 *initial window size*

uint32 *maximum packet size*

string *host to connect*

uint32 *port to connect*

string *originator IP address*

uint32 *originator port*

'*host to connect*' และ '*port to connect*' ระบุโฮสต์และพอร์ตที่ผู้รับควรเชื่อมต่อแชนเนล 'host to connect' อาจเป็นชื่อโดเมนหรือไอพีแอดเดรส

'*originator IP address*' เป็นหมายเลขไอพีแอดเดรสของเครื่องที่ทำการร้องขอการเชื่อมต่อ และ 'originator port' เป็นพอร์ตที่เครื่องทำการร้องขอการเชื่อมต่อใช้เชื่อมต่อ

แชนเนลทีซีพี/ไอพีที่ฟอร์เวิร์ดแล้วจะเป็นอิสระต่อเซสชันใดๆ และการปิดลงของเซสชันแชนเนลไม่ทำให้การเชื่อมต่อที่ฟอร์เวิร์ดไว้ถูกปิดลง

การอิมพลีเมนต์ส่วนไคลเอ็นต์ควรปฏิเสธการร้องขอการเปิดทีซีพี/ไอพีโดยตรงด้วยเหตุผลด้านความปลอดภัย

7.6 การเข้ารูปแบบโหมดของเทอร์มินอล (Encoding of Terminal Modes)

โหมดของเทอร์มินอล (ที่มากับการร้องขอเทอร์มินอลเทียม pty) ถูกเข้ารูปแบบเป็นสตรีมของไบต์ (byte stream) โดยมุ่งหมายให้ส่งผ่านข้ามแวดล้อมที่ต่างกันได้

คำอธิบายโหมดของ tty เป็นสตรีมของไบต์ สตรีมประกอบขึ้นด้วยคู่ของ ออปโค้ด-อากิวเมนต์ ซึ่งจะปิดท้ายด้วยออปโค้ด TTY_OP_END ออปโค้ดที่ 1 ถึง 159 มีอากิวเมนต์ uint32 หนึ่งค่า ส่วนออปโค้ด 160 ถึง 255 ยังไม่ได้นิยามไว้และทำให้การพาร์ (parsing) หยุดลง

ไคลเอ็นต์ควรจะใส่ทุกโหมดที่รู้จักลงในสตรีม และเซิร์ฟเวอร์เองอาจเพิกเฉยโหมดที่ไม่รู้จัก ทำให้เกิดความไม่ขึ้นกับเครื่อง โดยอย่างน้อยที่สุดทั้งสองระบบใช้อินเทอร์เฟสของ tty แบบคล้าย POSIX (POSIX-like)

ออปโค้ดข้างล่างนี้ได้ถูกนิยามไว้ ชื่อของออปโค้ดเป็นไปตามแฟลกโหมดของเทอร์มินอลแบบ POSIX

0	TTY_OP_END	Indicates end of options.
1	VINTR	Interrupt character; 255 if none. Similarly for the other characters. Not all of these characters are supported on all systems.
2	VQUIT	The quit character (sends SIGQUIT signal on POSIX systems).
3	VERASE	Erase the character to left of the cursor.
4	VKILL	Kill the current input line.
5	VEOF	End-of-file character (sends EOF from the terminal).
6	VEOL	End-of-line character in addition to carriage return and/or linefeed.
7	VEOL2	Additional end-of-line character.
8	VSTART	Continues paused output (normally control-Q).
9	VSTOP	Pauses output (normally control-S).
10	VSUSP	Suspends the current program.
11	VDSUSP	Another suspend character.
12	VREPRINT	Reprints the current input line.
13	VWERASE	Erases a word left of cursor.
14	VLNEXT	Enter the next character typed literally, even if it is a special character
15	VFLUSH	Character to flush output.
16	VSWTCH	Switch to a different shell layer.
17	VSTATUS	Prints system status line (load, command, pid etc).
18	VDISCARD	Toggles the flushing of terminal output.
30	IGNPAR	The ignore parity flag. The parameter SHOULD be 0 if this flag is FALSE set, and 1 if it is TRUE.

31	PARMRK	Mark parity and framing errors.
32	INPCK	Enable checking of parity errors.
33	ISTRIP	Strip 8th bit off characters.
34	INLCR	Map NL into CR on input.
35	IGNCR	Ignore CR on input.
36	ICRNL	Map CR to NL on input.
37	IUCLC	Translate uppercase characters to lowercase.
38	IXON	Enable output flow control.
39	IXANY	Any char will restart after stop.
40	IXOFF	Enable input flow control.
41	IMAXBEL	Ring bell on input queue full.
50	ISIG	Enable signals INTR, QUIT, [D]SUSP.
51	ICANON	Canonicalize input lines.
52	XCASE	Enable input and output of uppercase characters by preceding their lowercase equivalents with `\'.
53	ECHO	Enable echoing.
54	ECHOE	Visually erase chars.
55	ECHOK	Kill character discards current line.
56	ECHONL	Echo NL even if ECHO is off.
57	NOFLSH	Don't flush after interrupt.
58	TOSTOP	Stop background jobs from output.
59	IEXTEN	Enable extensions.
60	ECHOCTL	Echo control characters as ^(Char).
61	ECHOKE	Visual erase for line kill.
62	PENDIN	Retype pending input.
70	OPOST	Enable output processing.
71	OLCUC	Convert lowercase to uppercase.
72	ONLCR	Map NL to CR-NL.
73	OCRNL	Translate carriage return to newline (output).
74	ONOCR	Translate newline to carriage return-newline (output).
75	ONLRET	Newline performs a carriage return (output).
90	CS7	7 bit mode.
91	CS8	8 bit mode.
92	PARENB	Parity enable.

93	PARODD	Odd parity, else even.
128	TTY_OP_ISPEED	Specifies the input baud rate in bits per second.
129	TTY_OP_OSPEED	Specifies the output baud rate in bits per second.

7.7 สรุปหมายเลขหมายเลขแมสเซจ

#define SSH_MSG_GLOBAL_REQUEST	*80*
#define SSH_MSG_REQUEST_SUCCESS	*81*
#define SSH_MSG_REQUEST_FAILURE	*82*
#define SSH_MSG_CHANNEL_OPEN	*90*
#define SSH_MSG_CHANNEL_OPEN_CONFIRMATION	*91*
#define SSH_MSG_CHANNEL_OPEN_FAILURE	*92*
#define SSH_MSG_CHANNEL_WINDOW_ADJUST	*93*
#define SSH_MSG_CHANNEL_DATA	*94*

7.8 พิจารณาด้านความปลอดภัย

โพรโตคอลนี้จะถือว่าได้ทำงานอยู่บนการลำเลียงที่ผ่านการพิสูจน์ตนที่มีความปลอดภัย โพรโตคอลชั้นล่างทำให้เกิดการพิสูจน์ตนผู้ใช้และการป้องกันต่อการโจมตีระดับเน็ตเวิร์ก (network level attack)

อย่างไรก็ตาม โพรโตคอลนี้สามารถถูกใช้ในการเอ็กซีคิวคอมมานด์บนเครื่องรีโมท โพรโตคอลยังอนุญาตให้เซิร์ฟเวอร์รันคอมมานด์บนไคลเอ็นต์อีกด้วย การอิมพลีเมนต์อาจประสงค์ให้ไม่อนุญาตให้ทำเช่นนี้ เพื่อป้องกันผู้โจมตีที่มากับเซิร์ฟเวอร์เข้าจู่โจมไคลเอ็นต์

การฟอร์เวิร์ด X11 ทำให้เกิดการปรับปรุงด้านความปลอดภัยหลักจากการฟอร์เวิร์ด X11 ที่อยู่บนพื้นฐานของคุ้กกี้ปกติ คุ้กกี้ไม่เคยจำเป็นต้องส่งโดยไม่เข้ารหัส การสื่อสารถูกเข้ารหัสและปกป้องความสมบูรณ์ของข้อมูล จะไม่มีข้อมูลพิสูจน์ตนที่มีประโยชน์เหลืออยู่บนเครื่องเซิร์ฟเวอร์หลังจากการเชื่อมต่อถูกปิดลง ในทางตรงกันข้าม ในบางสถานการณ์การเชื่อมต่อ X11 ที่ทำการฟอร์เวิร์ดอาจถูกใช้เพื่อเข้าถึงเซิร์ฟเวอร์ X ที่ฝั่งโลคอลผ่านขอบเขตความปลอดภัยได้

การฟอร์เวิร์ดพอร์ตสามารถเป็นการอนุญาตให้ผู้ประสงค์ร้ายข้ามผ่านขอบเขตความปลอดภัยอย่างเช่นไฟร์วอลล์เข้ามาได้ มันไม่ควรยอมให้ผู้ใช้ทำในสิ่งที่ไม่ควรทำ อย่างไรก็ตามมันทำให้การเปิดทันเนลง่ายมาก การอิมพลีเมนต์ควรอนุญาตให้ควบคุมนโยบายบนบริการที่สามารถถูกฟอร์เวิร์ดได้ ผู้ดูแลระบบควรสามารถที่จะปฏิเสธการฟอร์เวิร์ดตามความสมค

เนื่องจากโพรโตคอลนี้ทำงานอยู่ภายในทันเนลที่เข้ารหัส ไฟร์วอลล์จะไม่สามารถตรวจสอบการส่งผ่านนี้ได้

ถ้าคีย์โฮสต์ถูกเปลี่ยนแปลง แนะนำว่าการอิมพลีเมนต์ควรยกเลิกการใช้งานที่อาจนำซึ่งอันตราย เช่น การฟอร์เวิร์ดเอเจนต์, การฟอร์เวิร์ด X11 และการฟอร์เวิร์ดทีซีพี/ไอพี

เอกสารอ้างอิงประกอบ

[RFC1766] Alvestrand, H., "Tags for the Identification of Languages", RFC 1766, March 1995.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[POSIX] ISO/IEC, 9945-1., "Information technology -- Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API) C Language", ANSI/IEE Std 1003.1, July 1996.

[SCHEIFLER] Scheifler, R., "X Window System : The Complete Reference to Xlib, X Protocol, Icccm, Xlfd, 3rd edition.", Digital Press ISBN 1555580882, Feburary 1992.

บทที่ 8

ภาษาจาวา

8.1 ประวัติของภาษาจาวา

หลายปีก่อนบางคนเชื่อว่าภาษาสำหรับโปรแกรมเป็นวิชาที่ตายแล้ว นั่นอาจเป็นเพราะว่าไม่มีอะไรที่ภาษาซีทำไม่ได้ และคงต้องอีกหลายปีที่คนส่วนใหญ่จะเข้าใจว่าคุณค่าที่แท้จริงของภาษาซีพลัสพลัส แต่เมื่อตอนต้นปี 1996 ภาษาจาวาก็มีชื่อขึ้นหน้าปกวารสารคอมพิวเตอร์ชั้นนำเกือบทุกยี่ห้อทั่วโลก และเป็นข่าวใน CNN อย่างต่อเนื่องนานนับปี ไม่เคยมีภาษาใดที่ได้รับความสนใจเช่นนี้มาก่อน ผู้เชี่ยวชาญบางคนกล่าวว่า ภาษาจาวาจะเปลี่ยนวงการคอมพิวเตอร์เหมือนกับที่ล้อเลื่อนเปลี่ยนความเป็นอยู่ของมนุษยชาติ

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

ผลกระทบเหล่านี้ทำให้ภาษาจาวาเป็นที่น่าสนใจอย่างยิ่ง ในขณะที่บริษัทผู้ผลิตซอฟต์แวร์ทั้งหลายก็ให้การต้อนรับภาษานี้ในทิศทางที่ดีเกินความคาดหมาย ประกอบด้วยภาษาจาวามีประสิทธิภาพในการสร้างโปรแกรมอย่างยิ่ง จึงเป็นเครื่องมือชั้นดีสำหรับเทคโนโลยีการสร้างโปรแกรมด้วยคอมโพเน็นต์และวิชวลโปรแกรมมิ่ง เชื่อได้แน่ว่าในอนาคตอันใกล้นี้ อุตสาหกรรมซอฟต์แวร์จะเปลี่ยนโฉมหน้าไปจากเดิมและการเรียนรู้ภาษาจาวาจะเป็นเรื่องที่หลีกเลี่ยงไม่ได้

ในช่วงต้น 1990s ตลาดเครื่องใช้ไฟฟ้า เช่นเครื่องซักผ้า, หม้อหุงข้าว,โทรทัศน์, ไมโครเวฟและอื่นๆ มีมูลค่าสูงกว่าตลาดคอมพิวเตอร์หลายเท่าตัว อีกทั้งระบบคอมพิวเตอร์ขนาดเล็กสำหรับควบคุมเครื่องใช้ไฟฟ้าเหล่านี้ก็ถูกพัฒนาดีขึ้นเรื่อยๆ บริษัท ซัน ไมโครซิสเต็มส์ ซึ่งประสบความสำเร็จในตลาดระบบเครือข่ายคอมพิวเตอร์อย่างมากในตอนนั้น เห็นว่าควรใช้ความได้เปรียบของบริษัท เร่งพัฒนา

เทคโนโลยีเพื่อยึดครองตลาดเครื่องใช้ไฟฟ้าที่มีคอมพิวเตอร์ควบคุมไว้ก่อนคนอื่น จึงจัดทีม Green group ขึ้นในปี 1991 มี James Gosling เป็นหัวหน้า ทำการพัฒนาระบบซอฟต์แวร์สำหรับควบคุมเครื่องใช้ไฟฟ้าขนาดเล็ก พวกเขาสร้างเครื่องต้นแบบที่เรียกว่า Star 7 เป็นระบบรีโมตคอลโทรลขนาดมือถือ สามารถควบคุมการเคลื่อนไหวของวัตถุโดยใช้นิ้วสัมผัสบนแป้นแสดงผล ระบบนี้ถูกทดสอบใช้การแสดงควบคุมตัว The Duke (ซึ่งต่อมาเป็นตัวนำโชคของภาษาจาวา) ใช้เคลื่อนที่ผ่านกลุ่มของวัตถุในมิติสมมติ

พวกเขาใช้ภาษาซีพลัสพลัสเขียนโปรแกรมของ Star 7 ผลที่ได้เป็นโปรแกรมที่มักจะทำงานผิดพลาด และล้มเหลวบ่อยๆ จนต้องสรุปว่าภาษาซีพลัสพลัสไม่เหมาะสำหรับงานแบบนี้ เพราะมีข้อจำกัดหลายอย่าง นั่นคือเครื่องใช้ไฟฟ้าขนาดเล็กมักมีหน่วยความจำน้อย ไม่เหมาะกับโปรแกรมภาษาซีพลัสพลัส ที่มักมีขนาดใหญ่ อีกทั้งในเครื่องเหล่านี้ไม่มีระบบปฏิบัติการ โปรแกรมจึงไม่สามารถเรียกใช้บริการของระบบปฏิบัติการได้เหมือนบนเครื่องที่มีระบบปฏิบัติการ ดังนั้นความสามารถของภาษาจึงจำกัดไปอย่างมาก เราจึงต้องสร้างโปรแกรมสำหรับทำงานพื้นฐานเอง นอกจากนี้ซีพลัสพลัสยังเป็นภาษาที่ไม่ปลอดภัย เพราะยอมให้มีการใช้พอยน์เตอร์อย่างไม่มีจำกัด และละเลยการทำการตรวจสอบชนิดข้อมูล โปรแกรมจึงมักมีความผิดพลาดซ่อนอยู่เป็นจำนวนมาก

ปัญหาที่สำคัญกว่าคือ หน่วยประมวลผลที่ใช้ในงานควบคุมมีมากหลายเบอร์หลายยี่ห้อ และมีชุดคำสั่งไม่เหมือนกัน โปรแกรมที่ทำงานได้บนหน่วยประมวลผลรุ่นหนึ่งจะต้องถูกคอมไพล์ใหม่ จึงจะนำไปใช้งานบนหน่วยประมวลผลอีกรุ่นหนึ่งได้ ด้วยเหตุนี้พวกเขาจึงพัฒนาภาษาใหม่ชื่อโอ๊ก (ได้ชื่อจากต้นโอ๊กที่อยู่ข้างตึกที่ทำงาน) ให้เป็นภาษาที่ง่ายต่อการเรียนรู้และใช้งาน ไม่มีข้อผิดพลาดในตอนทำงาน (bug–free) และเหมาะที่จะทำงานในระบบที่มีหน่วยความจำน้อย พวกเขาแน่ใจว่าระบบควบคุมจะมีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆจึงให้โอ๊กเป็นภาษาเชิงวัตถุ เพื่อง่ายในการสร้างและดูแลระบบขนาดใหญ่ ปัญหาใหญ่ของการออกแบบภาษาโอ๊กอยู่ที่ต้องการให้เป็นภาษาที่ทำงานบนหน่วยประมวลผลใดๆก็ได้ จึงนำเทคนิคการคอมไพล์โปรแกรมเป็นคำสั่งของหน่วยประมวลผลสมมติตัวหนึ่ง แล้วสร้างอินเทอร์พรีเตอร์ของหน่วยประมวลผลสมมติตัวนั้นให้แก่หน่วยประมวลผลที่จะทำงานโปรแกรมนั้น ด้วยวิธีนี้โปรแกรมที่สร้างขึ้นจึงสามารถนำไปทำงานบนเครื่องที่มีหน่วยประมวลผลต่างรุ่นได้ เราเรียกคุณสมบัติอย่างนี้ว่า platform independent

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

ในปี 1993 เมื่อ ไฮเพอร์เท็กซ์และบราวเซอร์ เปลี่ยนแปลงอินเทอร์เน็ตไปอย่างมากและมีผู้ใช้มากขึ้นอย่างรวดเร็ว บริษัท ซัน ไมโครซิสเต็มส์ มองเห็นความจำเป็นที่ต้องมีภาษาสำหรับสร้างโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ใดๆก็ได้ และเพิ่งนึกถึงคุณสมบัติ platform independent ของภาษาโอ๊ก จึงนำมาปัดฝุ่นปรับปรุงใหม่ และทดลองสร้างเว็บบราวเซอร์ชื่อเว็บรันเนอร์ที่ทำงานโปรแกรมภาษาโอ๊กได้ เมื่อทดลองจนได้ผล พวกเขาทราบว่ามีสิ่งที่สำคัญอย่างยิ่งในมือแล้ว แต่โอ๊กเป็นชื่อทางการค้าที่มีผู้ใช้อยู่ก่อน จึงเปลี่ยนเป็นจาวาในตอนต้นปี 1995 พร้อมกับเปลี่ยนชื่อเว็บรันเนอร์เป็น ฮอตจาวา (HotJava)

เมื่อเริ่มต้นจาวาทำงานบนระบบโซลาริสเท่านั้น แต่เพียงภายในฤดูร้อนของ 1995 ก็พัฒนาให้ทำงานได้บนวินโดวส์ เอ็นที, วินโดวส์ 95 และลีนุกซ์ พอถึงปลายปี 1995 บริษัทเน็ตสเคปก็สร้างเน็ตสเคป 2.0ให้ทำงานจาวาได้ หลังจากนั้นบริษัทไมโครซอฟต์กับไอบีเอ็ม ก็ประกาศสนับสนุนภาษาจาวาด้วย และในตอนปลายปี 1995 บริษัท ซัน ไมโครซิสเต็มส์ นำโปรแกรมชุดพัฒนาภาษาจาวา (JDK) รุ่น 1.0 ขึ้นแจกจ่ายใน อินเตอร์เน็ต

8.2 คุณสมบัติของภาษาจาวา

ภาษาจาวาถูกออกแบบให้มีลักษณะดังต่อไปนี้

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

- ภาษาจาวานำไวยากรณ์ภาษาส่วนใหญ่มาจากภาษาซีและซีพลัสพลัส ผู้ที่คุ้นเคยกับภาษาซีหรือซี พลัสพลัสอยู่แล้ว จะเข้าใจไวยากรณ์ภาษาจาวาได้ง่าย หรือใช้เวลาศึกษาไม่นานนัก
- ภาษาจาวามีกลไกของภาษาจำนวนไม่มากและไม่ซับซ้อนโดยตัดกลไกของภาษาซีและซี พลัสพลัสส่วนที่ทำให้ภาษายุ่งยากออกไปอย่างเช่น pointer arithmetic, default argument, scope resolution, protected and private inheritance และ operator overloading แต่ในขณะเดียวกันก็เพิ่มความสามารถให้คอมไพเลอร์ของภาษาจาวา ทำให้ไม่มี preprocessor commands ดังนั้นจะไม่มี macros definitions, included files, conditional compilation และ header files เมื่อจาวาเป็นภาษาเชิงวัตถุแล้ว กลไกอย่างเช่น structures, unions, bit fields และ enumerated types รวมทั้งการทำ typedef ก็ไม่มีความจำเป็นจึงถูกตัดออกไป ภาษาจาวาถูกออกแบบให้เป็นภาษาเชิงวัตถุอย่างรอบคอบกว่าภาษาซีพลัสพลัส ดังจะเห็นได้ว่ากลไกที่อาจทำให้เกิดความยุ่งยากวมอย่างเช่น multiple inheritance และ copy constructor และเป็นกลไกที่อาจจะทำลายแบบแผนการเขียนโปรแกรมเชิงวัตถุที่ดี ไป และ friend methods ก็ถูกตัดออกไปด้วย
- ภาษาจาวาประสบความสำเร็จอย่างมาก ด้วยการใช้เทคนิคภาษาเชิงวัตถุจึงช่วยสร้างโปรแกรมที่ยุ่งยากให้เป็นไปได้ง่ายขึ้น ดังจะเห็นได้ว่าหากเป็นภาษาอื่น การสร้างโปรแกรมที่เกี่ยวกับ graphic user interfaces, multitasking, network และ distributed objects ผู้ใช้โปรแกรมจะต้องมีความรู้ขั้นสูงจึงจะทำงานได้ หรือไม่ก็ต้องอาศัยโปรแกรมประเภทวิซาร์ดอย่างในวิชวลซีพลัสพลัสช่วยสร้างโปรแกรม แต่ภาษาจาวามีกลไกของโปรแกรมเชิงวัตถุที่ส่งเสริมการนำโปรแกรมที่สร้างไว้แล้ว

นำมาใช้งานใหม่ (reuse) โดยสามารถเปลี่ยนแปลงหรือเพิ่มเติมโปรแกรมบางส่วน ให้เหมาะกับงานที่ต้องการได้โดยไม่จำเป็นต้องทราบรายละเอียดของการสร้างโปรแกรมเดิมทั้งหมด จึงสามารถสร้างโปรแกรมสำหรับงานที่ยุ่งยากขึ้นจากโปรแกรมที่มีผู้สร้างไว้แล้วโดยง่าย

- ภาษาที่ถูกเรียกว่า typed language จะทำการตรวจสอบเกี่ยวกับ type เพื่อช่วยให้โปรแกรมทำงานโดยไม่มีข้อผิดพลาดเกี่ยวกับ types แต่การเขียนโปรแกรมภาษาเช่นนี้มีข้อยุ่งยาก อย่างเช่นตัวอักษรบวกกับเลขจำนวนเต็มไม่ได้ เนื่องจากค่าที่จะนำมาทำการคำนวณ หรือกำหนดค่า (assignment) ให้แก่กันจะต้องเป็น type ชนิดเดียวกัน ภาษาจาวาเป็น typed language แต่เพื่อให้ใช้งานง่าย จึงสร้างกฎเกณฑ์เกี่ยวกับ automatic type conversion ซึ่งเป็นการเปลี่ยนแปลงค่าระหว่าง type ที่ต่างกัน ช่วยให้การเขียนโปรแกรมง่ายขึ้น โดยลดภาระการเปลี่ยน types แก่ผู้เขียนโปรแกรม

2. โปรแกรมที่สร้างขึ้นด้วยภาษาจาวาจะไม่มีความผิดพลาดจากข้อบกพร่องของภาษา นั่นคือโปรแกรมจะต้องไม่ล้มเหลวลงด้วยความผิดปรกติเพียงเล็กน้อยที่ไม่เกี่ยวกับตรรกะของโปรแกรม คุณสมบัติของภาษาอย่างนี้เรียกว่า ความคงทน (Robust) ภาษาจาวาถูกออกแบบให้มีความคงทนด้วยวิธีการดังนี้
- ภาษาจาวาเน้นการใช้กลไก exception handling เพื่อให้โปรแกรมสามารถจัดการกับความผิดปกติบางอย่างที่เกิดขึ้นในขณะโปรแกรมทำงานให้โปรแกรมทำงานต่อไปได้โดยไม่ต้องหยุดลง
- ภาษาจาวาตัดกลไกบางอย่างในภาษาซีและซีพลัสพลัส ที่อาจจะทำให้เกิดความผิดพลาด หากใช้ไม่ระวัง เช่น global variables, variable length arguments และ goto statement ภาษาจาวาตัดการอ้างถึงแอดเดรสของตัวแปร และการใช้พอยน์เตอร์สำหรับอ่านหรือเขียนข้อมูลในหน่วยความจำโดยตรง
- ภาษาจาวาไม่มีกลไกสำหรับคืน (de-allocation) หน่วยความจำที่ขอมาในขณะโปรแกรมทำงาน (dynamic memory allocation) อย่างที่มีในภาษาซีและซีพลัสพลัส คือ free() และ delete() ภาษาจาวาอาศัย automatic garbage collector ทำหน้าที่เก็บหน่วยความจำที่ไม่สามารถอ้างถึงได้แล้วกลับไปใช้งานใหม่
- ภาษาจาวาเป็นภาษาประเภท strongly typed หมายถึง ภาษาที่เน้นความถูกต้องของชนิดข้อมูล (type) ที่ใช้ในโปรแกรม คอมไพเลอร์ของภาษาประเภทนี้จะทำการตรวจสอบว่าโปรแกรมการจัดการกับชนิดข้อมูลของตัวแปรถูกต้องหรือไม่ เรียกกิจกรรมของคอมไพเลอร์นี้ว่า type checking ความผิดพลาดเกี่ยวกับชนิดข้อมูลทั้งหมดจะถูกปฏิเสธตั้งแต่ตอนคอมไพล์โปรแกรมจึงไม่มีความผิดพลาดเกี่ยวกับชนิดข้อมูลเกิดขึ้นในระหว่างที่โปรแกรมทำงาน นอกจากนั้นยังมีการตรวจสอบอีกว่า ในระหว่างโปรแกรมทำงานมีการเปลี่ยนชนิดข้อมูล (casting) ระหว่างค่าต่าง type ถูกต้องหรือไม่ และการอ้างถึงสมาชิกในอาร์เรย์หรือสตริงอยู่ในขอบเขตที่ถูกต้องหรือไม่

3. โปรแกรมภาษาจาวามักถูกส่งผ่านระบบเครือข่ายไปทำงานบนเครื่องคอมพิวเตอร์ของผู้อื่น ดังนั้นภาษาจาวาต้องมีหลักประกันให้แก่ผู้รับโปรแกรมนั้นไปทำงานว่า จะไม่ก่อให้เกิดอันตรายต่อเครื่องหรือระบบของเขา ภาษาจาวาจึงมีข้อกำหนดหลายอย่าง เพื่อให้โปรแกรมไม่สามารถทำอันตรายหรือทำสิ่งที่ไม่สมควรทำ ต่อระบบที่รับโปรแกรมนั้นไปทำงาน คุณสมบัติของภาษาอย่างนี้เรียกว่า

ปลอดภัย (Secure) อย่างไรก็ตาม ไม่มีภาษาใดที่ปลอดภัยร้อยเปอร์เซ็นต์ ภาษาจาวาถูกจัดว่ามีความปลอดภัยในระดับสูงเท่านั้น เพราะถูกออกแบบมาเพื่อความปลอดภัยมากกว่าภาษาอื่น โดยแบ่งการป้องกันออกเป็นหลายระดับดังนี้

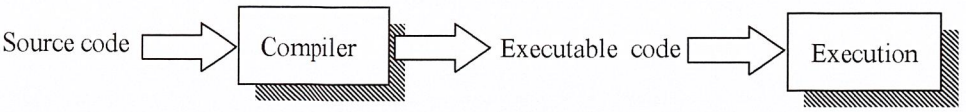
- ดังที่กล่าวมาแล้วว่า ภาษาจาวาไม่ยอมให้อ้างถึงค่าในหน่วยความจำผ่านทางพอยน์เตอร์และจะทำการตรวจสอบว่าการอ้างถึงสมาชิกในอาร์เรย์อยู่ในขอบเขตหรือไม่ โปรแกรมจึงไม่สามารถเขียนหรืออ่านค่าในหน่วยความจำที่ไม่มีสิทธิ์อ้างถึง การเปลี่ยนแปลงโปรแกรมหรือค่าในหน่วยความจำ เพื่อสร้างโปรแกรมที่จะเป็นอันตรายต่อคนอื่นด้วยวิธีนี้จึงเกิดขึ้นไม่ได้
- จาวาอินเทอร์พรีเตอร์ มี byte-code verifier ทำหน้าที่ตรวจสอบโปรแกรมที่จะถูกทำงานว่ามีคำสั่งที่ผิดปกติ หรือมีการทำงานที่ไม่สมควรหรือไม่ หากพบก็จะถูกปฏิเสธที่จะทำงานโปรแกรมนั้น
- ภาษาจาวามีระบบรักษาความปลอดภัยที่เรียกว่า sandbox model นั่นคือโปรแกรมที่ถูกนำมาจากเครื่องอื่นผ่านระบบเครือข่ายจะถือว่า เป็นโปรแกรมที่ไม่น่าไว้ใจ (un – trusted codes) และถูกเก็บอยู่ในภาวะที่เรียกว่า sandbox โปรแกรมที่อยู่ใน sandbox จะมีข้อจำกัดในการทำงานหลายอย่าง ซึ่งถูกควบคุมโดย security manager เช่น ไม่สามารถอ่านหรือเขียนไฟล์ได้เป็นต้น

4. จุดมุ่งหมายสำคัญของการออกแบบภาษาจาวา คือโปรแกรมต้องสามารถทำงานบนเครื่องต่างระบบกันได้ เรียกคุณสมบัติภาษาอย่างนี้ว่า “ไม่ขึ้นกับระบบ” (architecture neutral หรือ platform independent) เพื่อให้ได้คุณสมบัตินี้ ภาษาจาวาต้องมีการแปลภาษาแบบทั้ง Compilation และ interpretation รวมทั้งการกำหนดเครื่องจักรสมมติของจาวา (Java Virtual Machine) ดังจะกล่าวต่อไปนี้

8.3 คอมไพเลชัน และอินเทอร์พรีเทชัน (Compilation and interpretation)

ภาษาระดับสูง (high level language) อย่างเช่น ฟอร์แทรน, ปาสคาล, ซี, ซีพลัสพลัส หรือจาวา เป็นต้นมีความใกล้เคียงกับภาษาคณิตศาสตร์ เพื่อสะดวกในการคำนวณหรือแก้ปัญหาได้ง่ายกว่าการใช้คำสั่งที่หน่วยประมวลผลทำงานได้ (machine code) โดยตรง ภาษาระดับสูงช่วยให้การสร้างโปรแกรมมีประสิทธิภาพมากขึ้น เพราะผู้โปรแกรมสามารถคิดในระดับของการแก้ปัญหา แทนที่จะเป็นขั้นตอนการทำงานของหน่วยประมวลผลสำหรับแก้ปัญหานั้น ซึ่งหมายถึงผู้โปรแกรมไม่ต้องศึกษาทำความเข้าใจโครงสร้างและชุดคำสั่งของหน่วยประมวลผลนั้นก็สามารถสร้างโปรแกรมได้ อย่างไรก็ตาม หน่วยประมวลผลส่วนใหญ่ไม่สามารถทำงานโปรแกรมภาษาระดับสูงได้โดยตรง จึงต้องมีตัวแปลภาษา (Language translator) เปลี่ยนโปรแกรมต้นฉบับ (source code) ของภาษาระดับสูงให้เป็นโปรแกรมของคำสั่งที่หน่วยประมวลผลทำงานได้ เรียกว่า เอ็กซิคิวต์เอเบิลโค้ด โดยทั่วไปวิธีแปลภาษามีสองแบบคือ

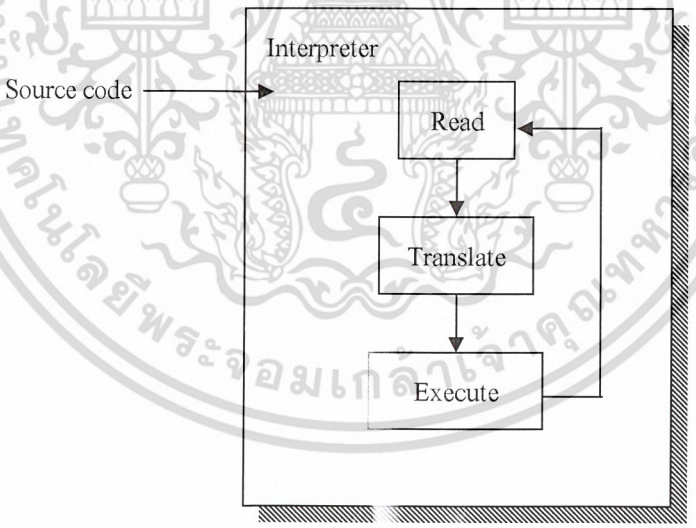
- **คอมไพเลชัน** ตัวแปลภาษาในกรณีนี้เรียกว่าคอมไพเลอร์ ทำหน้าที่วิเคราะห์โปรแกรมต้นฉบับแล้วสร้างเอ็กซิคิวต์เอเบิลโค้ดเป็นผลลัพธ์การกระทำดังกล่าวเรียกว่า การคอมไพล์โปรแกรม สรุปได้ดังนี้



รูปที่ 8. แสดงการการคอมไพล์โปรแกรม

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

- **อินเทอร์พรีเทชัน** ตัวแปลภาษาในกรณีนี้เรียกว่า อินเทอร์พรีเตอร์ ทำหน้าที่อ่านโปรแกรมต้นฉบับทีละบรรทัด แล้วแปลโปรแกรมบรรทัดนั้นเป็นเอ็กซิคิวต์เอเบิลโค้ดและทำงาน จากนั้นก็อ่านโปรแกรมต้นฉบับบรรทัดต่อไปมาทำเช่นเดิม ไปจนกว่าจะจบโปรแกรม การทำเช่นนี้เรียกว่าอินเทอร์เพต สรุปได้ดังรูปนี้

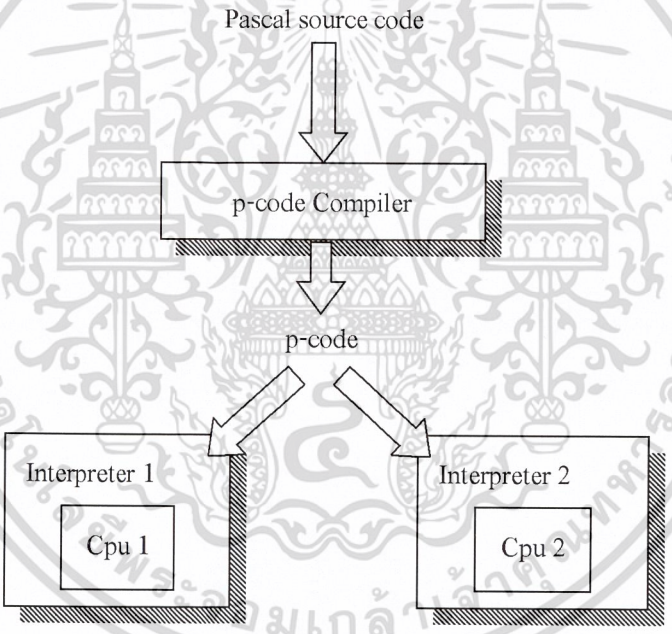


รูปที่ 8-2 แสดงการทำงานของอินเทอร์พรีเตอร์

วิธีนี้จะมีทั้งการแปลภาษาและทำงานโปรแกรมเกิดขึ้นสลับกันไป โปรแกรมจึงทำงานช้ากว่าการคอมไพเลชัน แต่มีข้อดีคือ อินเทอร์พรีเตอร์ถูกสร้างขึ้นได้ง่ายและมีขนาดเล็กกว่า เพราะการวิเคราะห์และแปลโปรแกรมทีละบรรทัดทำได้ง่ายกว่าทำทั้งโปรแกรม และการแปลภาษาระหว่างโปรแกรมทำงานจะทำให้ได้ภาษาที่มีความยืดหยุ่นในการเขียนโปรแกรมกว่า ตัวอย่างภาษาที่ใช้การแปลภาษาแบบอินเทอร์พรีเทชัน คือ ลิปซ์, เบสิก, โปรล็อก และสมอลล์ทอร์ก

วิธีแปลภาษาทั้งสองแบบมีทั้งข้อดีและข้อเสีย โดยปกติภาษาหนึ่งจะเลือกใช้การแปลภาษาเพียงแบบใดแบบหนึ่ง แต่ก็มีภาษาที่ใช้การแปลภาษาทั้งสองแบบเพื่อประโยชน์บางประการตัวอย่างคือ

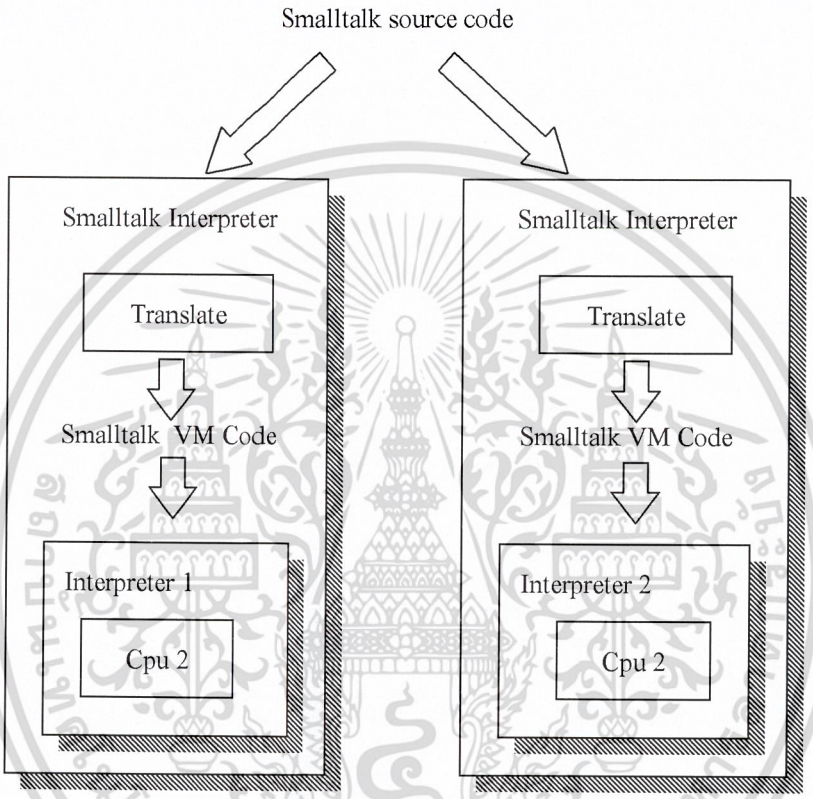
ในตอนต้น 1970s นักวิจัยที่ UC San Diego สร้างคอมไพเลอร์ที่แปลโปรแกรมภาษาปาสคาลเป็นโปรแกรมของภาษาสมมติหนึ่ง เรียกว่า p-code จากนั้นโปรแกรม p-code จะถูกทำงานโดย อินเทอร์พรีเตอร์ สำหรับหน่วยประมวลผลแต่ละรุ่น เหตุที่ต้องทำเช่นนี้ ก็เพราะในสมัยนั้นการสร้างคอมไพเลอร์ที่แปลภาษา ปาสคาลไปเป็นภาษาเครื่อง ถือเป็นเรื่องที่ยากมาก อีกทั้งในเวลานั้นยังไม่มีบริษัทคอมพิวเตอร์ใดครองส่วนแบ่งตลาดจำนวนมากได้อย่างปัจจุบัน จึงมีหน่วยประมวลผลหลายยี่ห้อใช้อยู่เป็นจำนวนมาก หากจะสร้างคอมไพเลอร์ภาษาปาสคาลให้แก่หน่วยประมวลผลรุ่นใดรุ่นหนึ่งก็อาจไม่คุ้ม เพราะไม่ได้ใช้อย่างแพร่หลายเขาจึงใช้วิธีคอมไพล์จากภาษาปาสคาล ไปเป็นภาษาสมมติ p-code ซึ่งเป็นภาษาที่ใกล้เคียงกับคำสั่งของหน่วยประมวลผลทั่วไป แล้วสร้างอินเทอร์พรีเตอร์ของ p-code สำหรับหน่วยประมวลผลแต่ละรุ่น ซึ่งไม่ใช่งานที่ยากนัก สรุปได้ดังรูปนี้



รูปที่ 8-3 การทำงานของโปรแกรม p-code

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

ในปลายยุค 1970s ภาษาสมอลล์ทอร์ก พัฒนาความคิดนี้ไปอีกแนวหนึ่งคือ แทนที่จะกำหนดเพียงภาษาสมมติเท่านั้น แต่กำหนดไปถึงโครงสร้างและพฤติกรรมของเครื่องจักรสมมติ (virtual machine) สำหรับทำงานโปรแกรมภาษาสมมตินั้นด้วย ภาษาสมอลล์ทอร์กใช้การแปลภาษาแบบอินเทอร์พรีเทชัน โดยสมอลล์ทอร์กอินเทอร์พรีเตอร์ ทำหน้าที่แปลโปรแกรมทีละบรรทัดไปเป็นโปรแกรมภาษาสมมติ แล้วทำงานโดยใช้เครื่องจักรสมมติที่จำลองขึ้น สรุปได้ดังรูปนี้

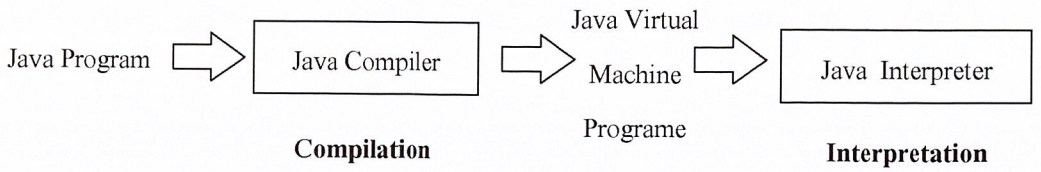


รูปที่ 8-4 แสดงการทำงานของเครื่องจักรสมมุติ

วิธีการนี้ทำให้โปรแกรมภาษาสมอลล์ทอร์กทำงานบนเครื่องคอมพิวเตอร์ใดๆที่มีสมอลล์ทอร์กอินเทอร์พรีเตอร์ให้ผลลัพธ์เหมือนกันหมด โดยไม่ต้องเปลี่ยนแปลงโปรแกรมส่วนใดเลย แม้ภาษาสมอลล์ทอร์กจะประสบความสำเร็จในเรื่องนี้ แต่ยังติดปัญหาที่โปรแกรมทำงานช้ามากจึงไม่ได้รับความนิยมเท่าที่ควร

8.4 เครื่องจักรสมมุติของจาวา (Java Virtual Machine)

ภาษาจาวานำความคิดการสร้างเครื่องจักรสมมติมาใช้ เพื่อให้โปรแกรมทำงานไม่ขึ้นกับระบบ โดยมีคอมไพเลอร์ทำการแปลภาษาให้เป็นโปรแกรมของ JVM แล้วนำโปรแกรมนั้นมาทำงานด้วยเครื่องจักรสมมติที่จำลองขึ้นโดยจาวาอินเทอร์พรีเตอร์ สรุปได้ดังรูปนี้



รูปที่ 8-5 การทำงานด้วยเครื่องจักรสมมติที่จำลองโดยจาวาอินเทอร์พรีเตอร์

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

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

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

JVM ถูกออกแบบให้สามารถจำลองได้บนประมวลผลทั่วไป แต่หน่วยประมวลผลทั่วไปมีจำนวน รีจีสเตอร์ไม่เท่ากัน บางรุ่นมีรีจีสเตอร์หน้าที่พิเศษที่รุ่นอื่นไม่มี ผู้ออกแบบจึงตัดปัญหานี้โดยให้ JVM ไม่มีรีจีสเตอร์และทำการคำนวณทั้งหมดบนสแต็ก ชุดคำสั่งของ JVM จึงเป็น stacked operations กล่าวได้ว่า JVM เป็น Stack machine

วิชวลโพรเซสเซอร์ใน JVM จะเปลี่ยนคำสั่งไบต์โค้ดไปเป็นเนทีฟโค้ดที่ทำหน้าที่เดียวกันแล้วทำงาน เนทีฟโค้ดนั้น สังเกตว่าเนทีฟโค้ดนั้นอาจเป็น Application Program Interfaces (API) ของระบบปฏิบัติการที่ใช้หรืออาจจะเป็นไลบราลีมาตราฐานที่สร้างขึ้นสำหรับหน่วยประมวลผลนั้น ทำให้ JVM หนึ่งอาจใช้งานได้ในระบบต่างๆ โดยเปลี่ยนแปลงแต่ไลบราลีมาตราฐานเท่านั้น

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

บทที่ 9

การออกแบบและการสร้าง

การออกแบบโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัยสามารถทำได้โดยการนำเอาโพรโตคอลที่สร้างความปลอดภัยเข้ามาประยุกต์ โดยได้ใช้โพรโตคอลเอสเอสเอชมาช่วยในการเข้ารหัสในส่วนของข้อมูลที่เกี่ยวข้องกับการติดต่อสื่อสารกับเซิร์ฟเวอร์ซึ่งรวมไปถึงการเข้ารหัสชื่อผู้ใช้, รหัสผ่านและข้อมูลต่างๆ นอกจากนี้ยังช่วยเพิ่มความปลอดภัยในการพิสูจน์ตนเพื่อป้องกันการโจมตีประเภทต่างๆ อีกด้วย

ในการพัฒนานั้น เริ่มด้วยการศึกษาการทำงานของโพรโตคอลเอสเอสเอชที่จะนำมาใช้ในการพัฒนา ซึ่งโพรโตคอลนี้ ประกอบด้วยการทำงานในรูปแบบเวอร์ชัน 1 และ 2 การทำงานของโพรโตคอลเอสเอสเอช 1 และ 2 นั้นทำงานแตกต่างกันโดยสิ้นเชิง อันเป็นผลเนื่องมาจากโพรโตคอลเอสเอสเอช 2 นั้นได้ถูกออกแบบและอิมพลีเมนต์ขึ้นมาใหม่ทั้งหมด โดยไม่ยึดติดหลักการทำงานและการอิมพลีเมนต์เดิมมาใช้ในการพัฒนา

การศึกษาได้แบ่งย่อยออกเป็น 2 ส่วน สอดคล้องกับแต่ละเวอร์ชันของโพรโตคอล เนื่องจากการทำงานของโพรโตคอลเอสเอสเอช 1 นั้นไม่ได้ออกแบบมาเพื่อการถ่ายโอนแฟ้มข้อมูลโดยเฉพาะ หากแต่ออกแบบมาเพื่อทำให้เกิดช่องทางที่ปลอดภัยแบบอเนกจุดประสงค์ ส่วนการพัฒนาให้รองรับโพรโตคอลเอสเอสเอช 2 นั้น จะพัฒนาตามหลักการทำงานของระบบย่อยเอสเอฟทีพี (sftp) ที่รองรับโดยโพรโตคอล โดยรายละเอียดจะได้อธิบายต่อไป

จากนั้นจึงเลือกภาษาที่จะนำมาใช้ในการพัฒนา ด้วยความปลอดภัยเป็นเงื่อนไขที่สำคัญในการพัฒนา จึงเลือกใช้ภาษาจาวาที่ให้ความปลอดภัยสูง อีกทั้งเป็นภาษาเชิงวัตถุที่สะดวกต่อการพัฒนาอีกด้วย

9.1 ภาพรวมของการออกแบบ

โพรโตคอลเอสเอสเอชเวอร์ชั่น 2 ที่นำมาใช้นั้น สามารถใช้การเข้ารหัสได้หลายวิธี และได้ออกแบบส่วนของการเรียกใช้งานให้เป็นโมดูลเพื่อที่สามารถเรียกใช้การเข้ารหัสแบบต่างๆได้ง่าย

คุณสมบัติของโพรโตคอลเอสเอสเอชเวอร์ชั่น 2 ที่สำคัญมากที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมที่พัฒนาขึ้นมา มี 2 ส่วนคือ IsagSFTP ใช้โพรโตคอลSFTP และ IsagTerm ซึ่งใช้โพรโตคอลSSH

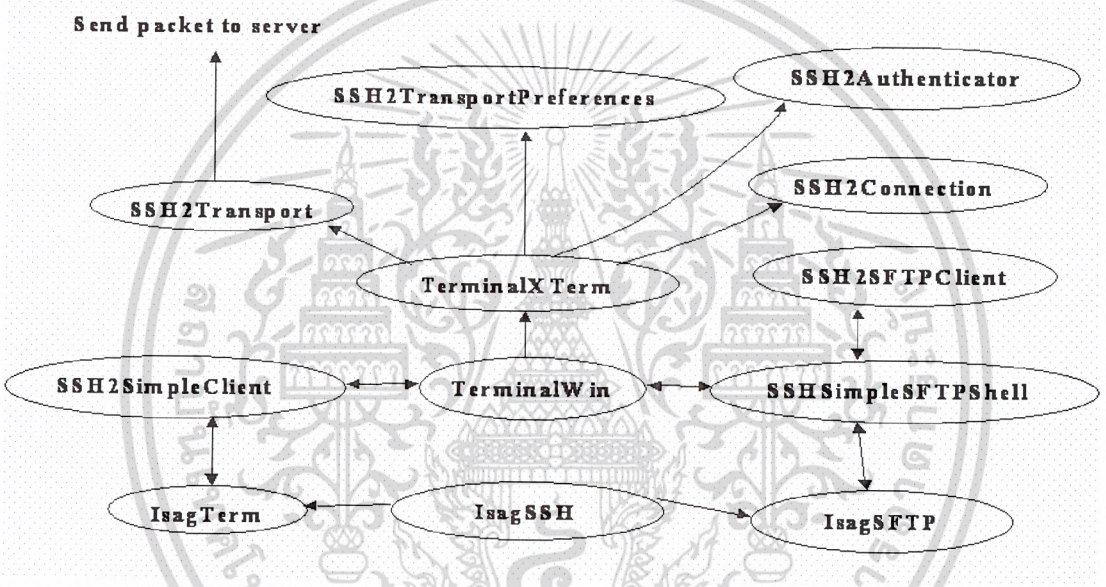
โดยโพรโตคอล SFTP นี้จะส่งข้อมูลต่างๆผ่านพอร์ท22 ซึ่งเป็นโพรโตคอล SSH โดยจะเข้ารหัสข้อมูล ทั้งหมดไม่ว่าจะเป็นการสร้างการเชื่อมต่อ การส่งคำสั่ง หรือการรับ-ส่งข้อมูล ซึ่งโดยปกติแล้ว

โพรโตคอลSSH เวอร์ชั่นแรก จะใช้การทำ Port Forwording ซึ่งจะเข้ารหัสเฉพาะการสร้างการเชื่อมต่อและการส่งคำสั่งเท่านั้น ส่วนข้อมูลจะไม่มีการเข้ารหัสและส่งไปทางพอร์ตของ FTP ปกตินั่นเอง

ส่วนโพรโตคอล SSH นั้นโดยปกติจะใช้แทนการจำลองเทอร์มินอลระยะไกลแบบเดิม เช่น Telnet , Remote Shell(Rsh) เป็นต้น แต่จะมีความปลอดภัยมากกว่า โดยจะนำโมดูลการเข้ารหัสแบบต่างๆมาใช้ในการออกแบบ

9.2 การออกแบบโปรแกรม

โปรแกรมที่ได้พัฒนามานี้มีชื่อว่า IsagSSH2547 มีลักษณะในมุมมองของการใช้งานของผู้ใช้จะแบ่งออกเป็นส่วนๆ ดังรูป



คุณสมบัติของโพรโตคอลเอสเอสเอช ที่สำคัญมากที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมที่พัฒนาขึ้นมา มี 2 ส่วนคือ การใช้โพรโตคอลเอสเอฟทีพีในโพรโตคอลเอสเอสเอช 2 และการจำลองเทอร์มินอลระยะไกลผ่านโพรโตคอล SSH เวอร์ชั่น 2

โดยแต่ละคลาสมีหน้าที่ดังนี้

- **IsagSSH** คลาสนี้จะทำการเลือกที่จะใช้งาน IsagSFTP หรือ IsagTerm หรือโหลดการตั้งค่าจากไฟล์ที่ระบุผ่านพารามิเตอร์
- **IsagSFTP** คลาสนี้จะทำหน้าที่จัดรูปแบบของ GUI และทำการกำหนดลำดับการติดต่อกับ SSH Server โดยจะทำงานติดต่อผ่าน SSH2SimpleSFTPShell
- **IsagTerm** คลาสนี้จะทำการเชื่อมต่อกับSSH Server และทำการกำหนดลำดับขั้นตอนการติดต่อกับ SSH Server โดยจะทำงานติดต่อผ่าน SSH2SimpleClient
- **SSH2SimpleSFTPShell** คลาสนี้จะทำการสร้างคำสั่งต่างๆของSFTP

- **SSH2SimpleClient** คลาสนี้จะทำการเชื่อมต่อกับเซิร์ฟเวอร์ด้วยโพรโตคอลเอสเอสเอช 2 ทำการพิสูจน์ตน แล้วจึงโอนการส่งข้อมูลไปยังserver
- **SSH2SFTPClient** คลาสนี้จะทำงานตามโพรโตคอลเอสเอฟทีพี โดยการทำงานจะมีลักษณะคล้ายกับโพรโตคอลเอฟทีพีทั่วไป แต่อาศัยการเชื่อมต่อที่ได้จากโพรโตคอลเอสเอสเอช 2
- **SSH2TransportPreferences** คลาสนี้จะทำการเก็บค่าคุณสมบัติต่างๆ ที่ต้องใช้ในการสร้างการติดต่อกับเซิร์ฟเวอร์
- **SSH2Transport** คลาสนี้จะนำคุณสมบัติต่างๆจากคลาส SSH2TransportPreferences มาใช้ แล้วจึงทำการสร้างซ็อกเก็ตเพื่อใช้ในการติดต่อกับเซิร์ฟเวอร์ และทำหน้าที่สร้างแพ็กเก็ตที่ใช้รับหรือส่งระหว่างเซิร์ฟเวอร์ นอกจากนั้นยังทำหน้าที่ในการแลกเปลี่ยนคีย์สาธารณะอีกด้วย
- **SSH2Authenticator** คลาสนี้ทำหน้าที่ในการพิสูจน์ตนกับเซิร์ฟเวอร์ โดยจะส่งข้อมูลต่างๆไปยังคลาส SSH2Transport เพื่อทำการสร้างแพ็กเก็ต และทำการเข้ารหัส แล้วจึงส่งไปให้เซิร์ฟเวอร์
- **SSH2Connection** คลาสนี้จะทำการสร้างการเชื่อมต่อกับเซิร์ฟเวอร์โดยใช้ การพิสูจน์ตนจากคลาส SSH2Authenticator และใช้การรับและส่งแพ็กเก็ตจากคลาส SSH2Transport
- **TerminalXTerm** เป็นคลาสที่ทำหน้าที่ตรวจสอบคำสั่งและอักขระระหว่างไคลเอนท์และServer
- **TerminalWin** เป็นคลาสที่สร้างหน้าต่างTerminal และPrompt ในการรับคำสั่ง โดยอาศัยคำสั่ง Virtual Terminal ของ TerminalXTerm

9.3 รายละเอียดของการพัฒนา

ในขั้นตอนการสร้างโปรแกรมเป็นส่วนที่ต้องทำความเข้าใจกับคลาสต่างๆรวมถึงการใช้งานที่ถูกต้อง โดยการอิมพลีเมนต์ตามสถาปัตยกรรมของ โพรโตคอลเอสเอสเอช ก่อนอื่นเราต้องทำการสร้าง SSH-TRANS โดยใช้ซ็อกเก็ตเป็นตัวกำหนดการเชื่อมต่อ ซึ่งต้องมีการกำหนดโฮสต์และพอร์ต ในส่วนแรกนี้ทั้ง IsagTerm 2547 และ IsagSFTP 2547 ใช้งานเหมือนกัน

```
.
.
RandomSeed seed = new RandomSeed();
seed.addEntropyGenerator(terminal);
SecureRandomAndPad secureRandom = new SecureRandomAndPad(new
SecureRandom(seed.getBytesBlocking(20, false)) ;
.
.
SSH2Transport transport = new SSH2Transport(new Socket(host, port), prefs,
secureRandom);
.
```

ซึ่งในการส่งข้อมูลในลำดับชั้นนี้จะทำการรวบรวมข้อมูลโดย ซึ่งเป็นคลาสที่รองรับแพ็กเก็ตที่ใช้ส่งจริงบนอุโมงค์ข้อมูลที่มีการเข้ารหัส ไม่ว่าจะเป็นการเข้ารหัสและการบีบอัดข้อมูลก็จะกระทำที่ SSH2TransportPDU นี้

เมื่อทำการกำหนด SSH2 Transport แล้ว ก็ให้ทำการสร้างและกำหนดวิธีการพิสูจน์ตนใน SSH2 Authentication โดยโครงงานนี้จะใช้การพิสูจน์ตนโดยใช้รหัสลับ

```
SSH2AuthModule authmodule= new SSH2AuthPassword(password);
SSH2Authenticator authenticator = new SSH2Authenticator(username);
authenticator.addModule(authModule);
SSH2UserAuth userAuth = new SSH2UserAuth(transport, authenticator);
```

เมื่อกำหนดการพิสูจน์ตนแล้ว จึงสร้างการเชื่อมต่อใน SSH2 Connection

```
SSH2Connection connection = new SSH2Connection(userAuth, transport);
```

นำconnection ที่ได้ไปกำหนดเพื่อบอกกับ SSH Transport

```
transport.setConnection(connection);
```

ในส่วนของการกำหนดการทำงานให้กับ Terminal ซึ่งเป็น Client Console ที่ใช้ติดต่อกับเซิร์ฟเวอร์ โดยมีการใช้งานทั้ง IsagTerm 2547 และ IsagSFTP 2547 ดังตัวอย่าง

```
Frame frame = new Frame();
TerminalWin terminal = new TerminalWin(frame, new TerminalXTerm() , props);
```

การเขียนข้อความลงไปบนTerminal นั้น ทำได้โดยใช้

```
terminal.write("IsagSFTP2547");
```

ซึ่งเมธอดดังกล่าวจะไม่ส่งข้อมูลไปยังเซิร์ฟเวอร์ แต่จะแสดงข้อความลงไปบนTerminal เท่านั้น

ยังมีคลาสสำคัญที่ไม่ได้อยู่ในชุดคลาสเดียวกันกับTerminal แต่มีความสัมพันธ์กันซึ่งก็คือ

- SSH2ConsoleRemote ใช้ในการอิมพลีเมนต์การควบคุมRemote Command หรือ เชลล์รับคำสั่ง ซึ่งสามารถใช้สั่งให้มีการทำงานหรือควบคุมอินพุทเอาท์พุทจากเชลล์รับคำสั่งซึ่งกำหนดได้ดังนี้

SSH2ConsoleRemote console = new SSH2ConsoleRemote(client.getConnection());

- SSH2TerminalAdapterImpl เป็นคลาสที่ใช้อินเตอร์เฟสระหว่าง Terminal windows และ ssh2 ซึ่งต้องมีการกำหนดให้มีการติดต่อกันระหว่างคลาส โดย

SSH2TerminalAdapterImpl termAdapter = new SSH2TerminalAdapterImpl(terminal);

การกำหนดให้ Terminal ทำการตรวจสอบการส่งคำสั่งในการติดต่อกับเซิร์ฟเวอร์ ทำได้โดย

LineReaderTerminal lineReader = new LineReaderTerminal(terminal);

หากต้องการให้มีการกำหนดพรอมพท์รับคำสั่งหรือรับอินพุท ทำได้โดยใช้เมธอด prompLine ของ คลาส LineReaderTerminal

public String **promptLine**(String prompt, String defaultVal, boolean echoStar)

ดังตัวอย่างต่อไปนี้

String host = lineReader.promptLine(" \r\nsftp server : ", null, false);

และในส่วนของการออกแบบ IsagSFTP นั้นมีคลาสต่างๆที่สำคัญดังนี้

- SSH2SFTP เป็นคลาสที่ใช้กำหนดรายละเอียดของโพรโตคอลเอสเอฟทีพี เช่น Command Code ที่ใช้ควบคุมการส่งข้อมูล เป็นต้น

- SSH2SFTPTransfer เป็นคลาสที่ช่วยในการรับ-ข้อมูลกับเซิร์ฟเวอร์

- SSH2SFTPClient เป็นคลาสที่ใช้สร้างและกำหนดการทำงานของฝั่งไคลเอนต์ของโพรโตคอลเอสเอฟทีพี โดยการกำหนดคำสั่งหรือฟังก์ชันการทำงานต่างๆ

บทที่ 10

การทดลองและผลการทดลอง

หลังจากที่ได้ออกแบบและพัฒนาโปรแกรมแล้ว ต้องมีการทดสอบโปรแกรมที่พัฒนาขึ้นมา โดยทดลองติดตั้งโปรแกรมกับเครื่องที่มีระบบปฏิบัติการต่างๆ แล้วล็อกอินเข้าใช้งานเซิร์ฟเวอร์ที่เปิดบริการ Secure Shell

จุดประสงค์ของการทดลอง

- เพื่อทดสอบความแตกต่างระหว่าง โครงงานเดิมคือ IsagTerm 2542 และ IsagFTP ซึ่งใช้โพรโตคอล SSH-1 กับโครงงานซึ่งพัฒนามาใหม่คือ IsagTerm 2547 และ IsagSFTP ซึ่งใช้โพรโตคอล SSH-2
- เพื่อทดสอบความเข้ากันได้ของโปรแกรมที่พัฒนาขึ้นกับโพรโตคอลมาตรฐานของ Secure Shell version 2.0 ให้สามารถใช้งานได้จริงและการทำงานสื่อสารระหว่างทั้งสองฝั่งไม่ผิดพลาด
- เพื่อทดสอบความเข้ากันได้กับระบบปฏิบัติการต่างๆ ที่ใช้งานจาวาแอพพลิเคชั่นกับโปรแกรมที่พัฒนาขึ้นให้ทำงานร่วมกันได้
- เพื่อทดสอบความสามารถในการใช้งานภาษาไทยบนเทอร์มินอลที่ติดต่อ โดยไม่ขึ้นกับระบบและสภาพแวดล้อมในการทำงานของเครื่องคอมพิวเตอร์ที่ทำงาน
- เพื่อใช้เป็นแนวทางในการนำเอาระบบความปลอดภัยที่ใช้ในโพรโตคอล ไปประยุกต์ใช้กับระบบการรักษาความปลอดภัยทางข้อมูลอื่นๆ

การเตรียมอุปกรณ์และสภาวะการที่ทำงานเพื่อทดลองโปรแกรม

อุปกรณ์ที่ใช้ในการทดสอบโปรแกรมและอุปกรณ์ที่ต้องใช้ในระบบมีดังต่อไปนี้

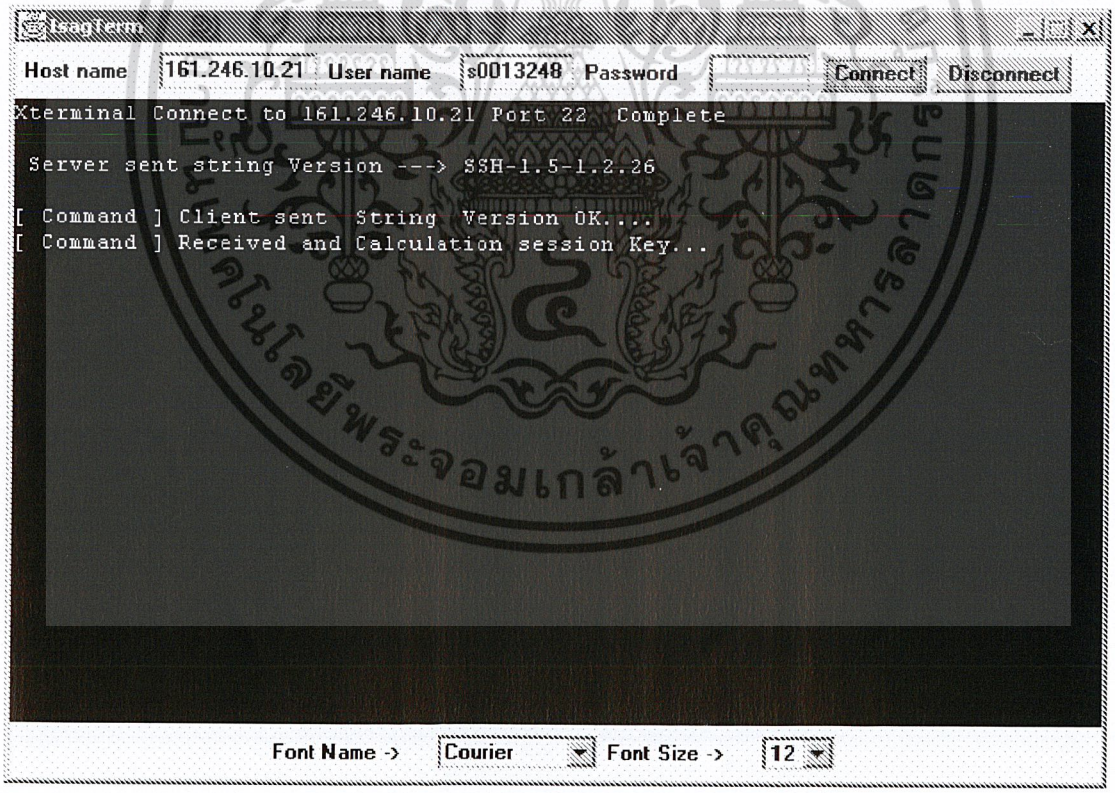
- เครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรม Java Develop Kit (JDK) เวอร์ชัน 1.1.8 ขึ้นไป ไม่ขึ้นกับระบบปฏิบัติการ
- เครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรม Java Virtual Machine
- เครื่องคอมพิวเตอร์ที่ให้บริการเป็นเซิร์ฟเวอร์และติดตั้ง Secure Shell Server ที่ให้การสนับสนุนเวอร์ชัน 1.0 ขึ้นไป
- ระบบฮาร์ดแวร์ที่ใช้ซีพียูความเร็ว 133MHz ขึ้นไป

การทดลองโปรแกรมและผลการทดลอง

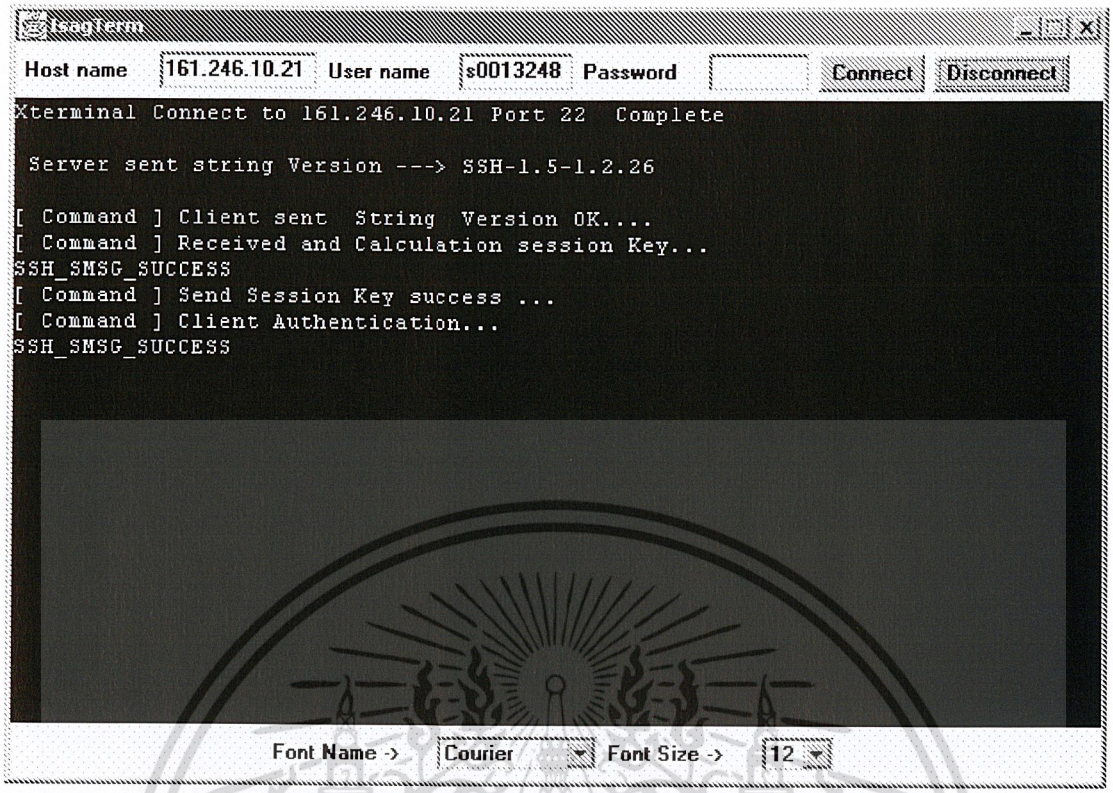
10.1. การทดลองใช้โปรแกรม IsagTerm เดิม

ในการทดลองใช้ โปรแกรม IsagTerm นั้น จะรองรับ Server ที่ให้บริการ Secure Shell เวอร์ชันไม่เกิน 2.0 โดยยังมีข้อจำกัดหลายประการในการพัฒนาได้คือ

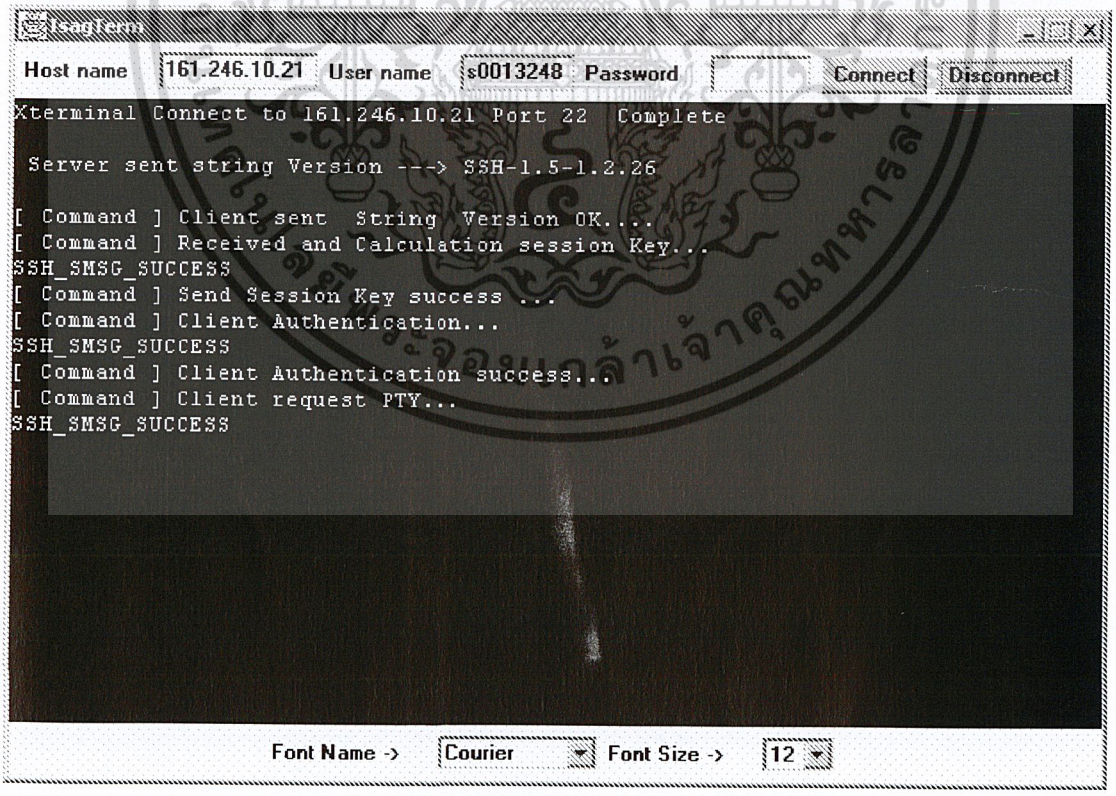
- การรองรับการเข้ารหัสในโปรแกรมยังไม่ครอบคลุมอัลกอริทึมการเข้ารหัสทั้งหมดของเซิร์ฟเวอร์ Secure Shell ที่มีให้บริการ
- อัลกอริทึมที่ใช้ในการเข้ารหัสคีย์มีความซับซ้อนมากทำให้เสียเวลาในการทำงานค่อนข้างนาน และตัวจาวาเองก็มีการทำงานค่อนข้างช้า ทำให้มีโปรแกรมทำการติดต่อสื่อสารได้ช้า



รูปที่ 10-1 แสดงการติดต่อกับเซิร์ฟเวอร์ในขั้นตอนการตรวจสอบเวอร์ชัน



รูปที่ 10-2 แสดงการแลกเปลี่ยนคีย์ที่ใช้เข้ารหัสระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์



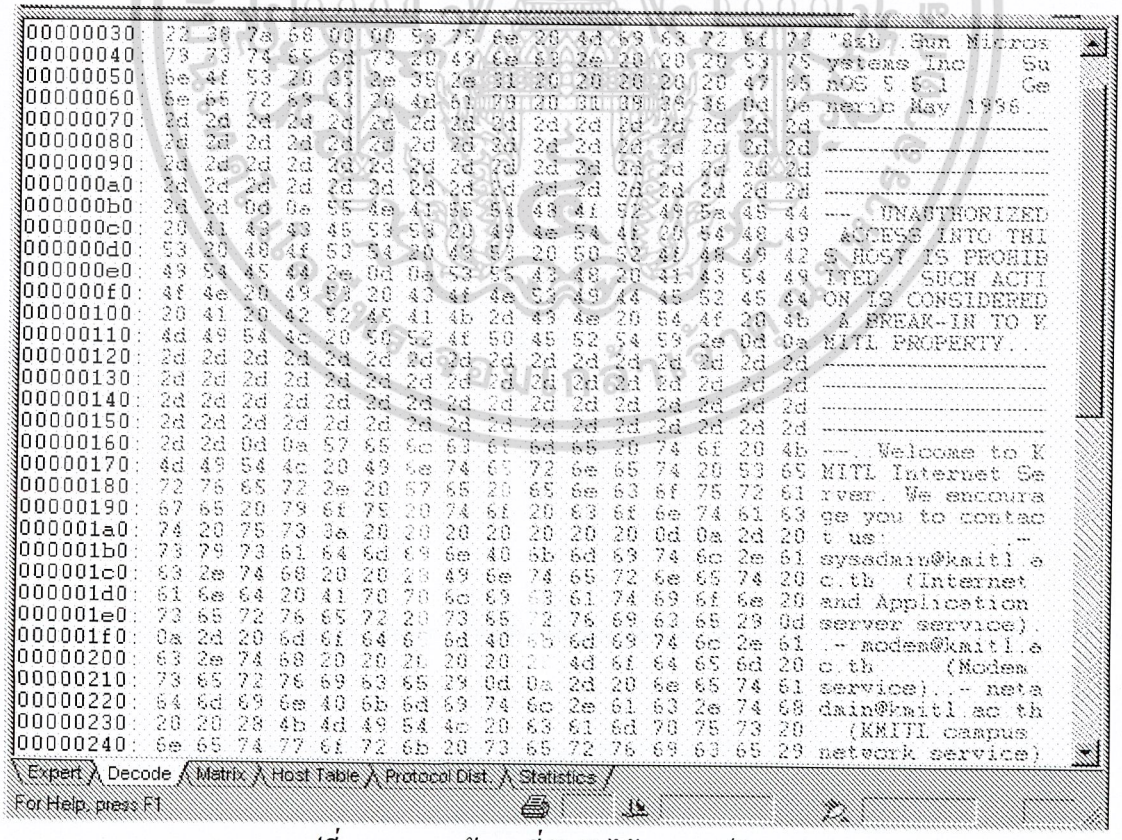
รูปที่ 10-3 แสดงการตรวจสอบและพิสูจน์สิทธิ์ผู้ใช้

ข้อมูล password ปกติ	ข้อมูลที่ได้จาก Telnet	ข้อมูลที่ได้จากโปรแกรม Secure shell
Fxicdi,g,viN	F , x , i , c , d , i , g , v , i , N	. . . . . ▮▮ÀR·▮P. ÿûx. . . . . Ù. .á®L ³»*ù Î È▮ : ² ^ ß Ùh \ ¬ | >S

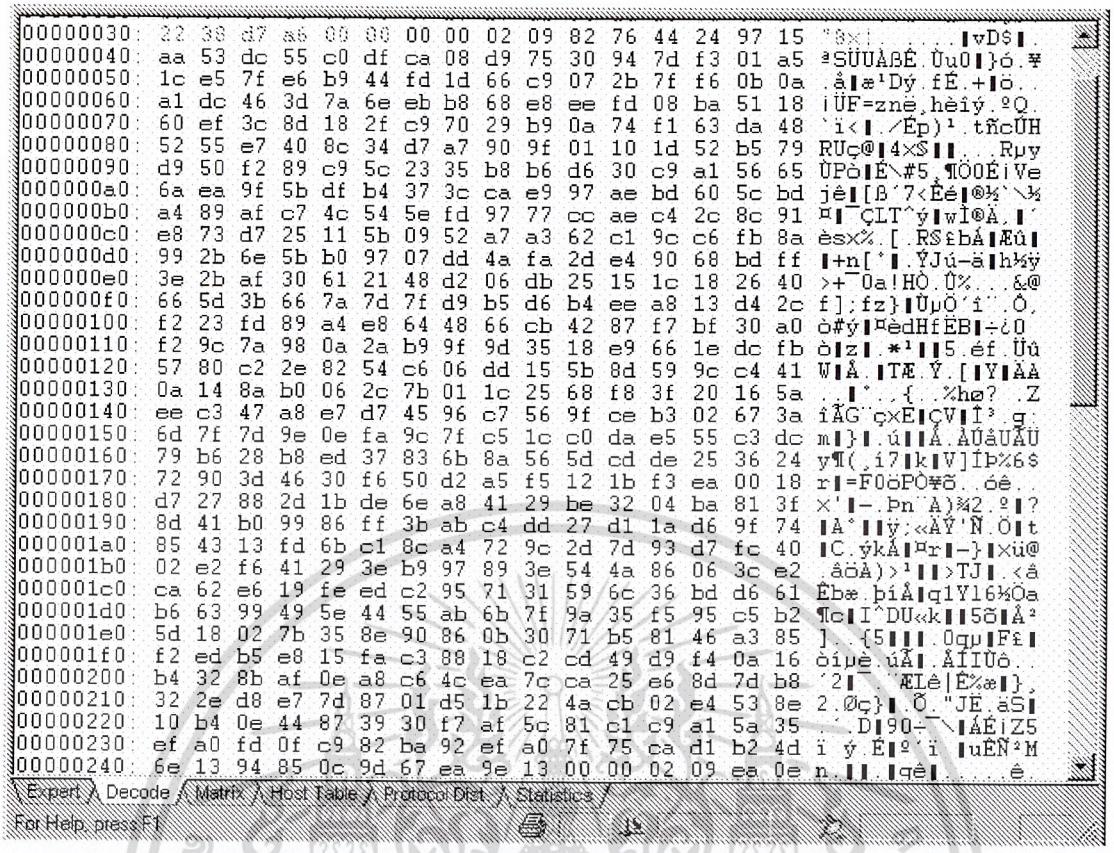
แสดงการเปรียบเทียบข้อมูลที่ดักจับได้บน NetXray ระหว่างเทลเน็ตกับ Secure Shell

ช่วงการแลกเปลี่ยนข้อมูลแบบอินเตอร์แอ็กทีฟ

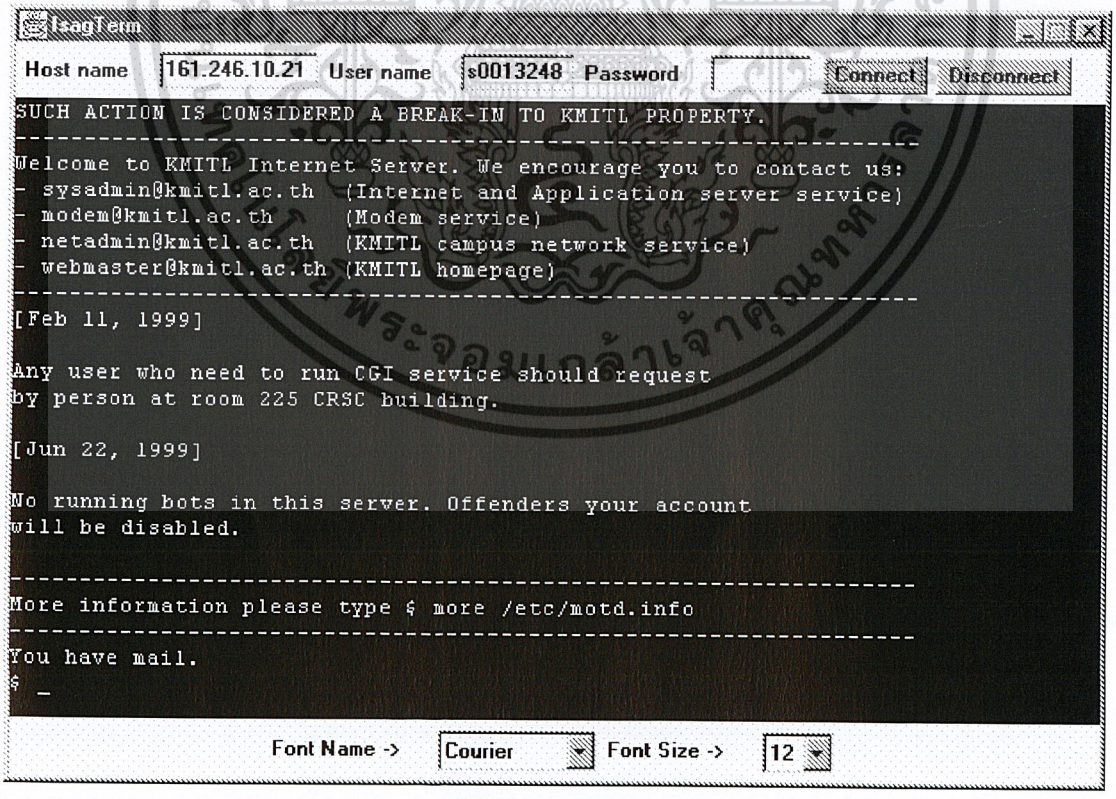
เมื่อติดต่อแลกเปลี่ยนข้อมูลและตรวจสอบสิทธิ์ผู้ใช้เรียบร้อยแล้ว จะเข้าสู่ขั้นตอนการติดต่อสื่อสารข้อมูลกันแบบอินเตอร์แอ็กทีฟ เมื่อมีการกดคีย์ใดๆ ไคลเอ็นต์จะส่งข้อมูลไปยังเซิร์ฟเวอร์ แปลความหมายแล้วส่งข้อมูลดังกล่าวกลับมา การสื่อสารข้อมูลระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้น จะต้องเข้ารหัสเพื่อไม่ให้ดักจับข้อมูลไปอ่านทำความเข้าใจได้ทันทีดังเช่นโปรแกรมเทลเน็ต การเข้ารหัสและถอดรหัสในส่วนของการแลกเปลี่ยนข้อมูลแบบอินเตอร์แอ็กทีฟนี้จะใช้การเข้าและถอดรหัสชนิด 3DES ที่มีความน่าเชื่อถือได้ และมีความเร็วพอสมควรในการเข้ารหัสข้อมูล



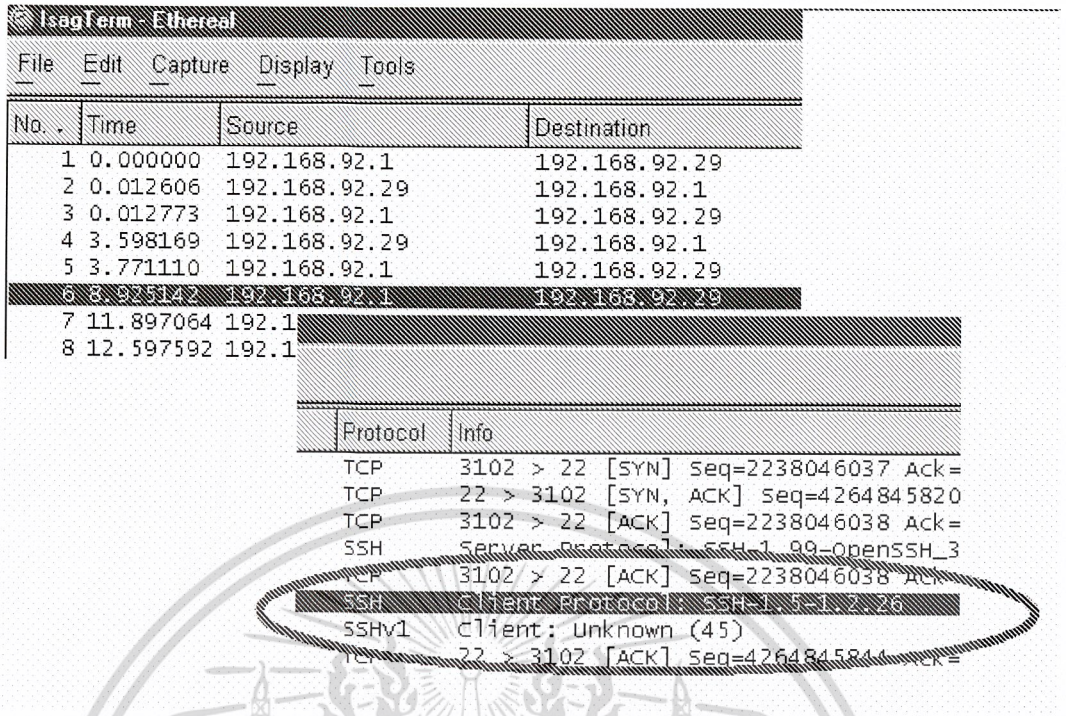
รูปที่ 10-4 แสดงข้อมูลที่ดักจับได้จากเทอร์มินอลเทลเน็ต



รูปที่ 10-5 แสดงข้อมูลที่ดักจับได้จากเทอร์มินอล Secure Shell



รูปที่ 10-6 แสดงหน้าจอและการทำงานของโปรแกรมเทอร์มินอล Secure Shell ด้วยจาวา



แสดงข้อมูลการเชื่อมต่อไปยังเซิร์ฟเวอร์ ที่ดักจับได้

จากรูป เป็นรูปการทดลอง IsagTerm 2542 ซึ่งเป็นโครงงานตัวเก่า โดยเป็นการ Connect ไปยัง ip 192.168.92.29 ซึ่งเป็น server ที่เป็น Linux และทางผู้ทดลองได้ใช้โปรแกรมในการดักจับแพ็กเก็ตคือ โปรแกรม Ethereal ซึ่งสามารถที่จะดาวโหลดไปใช้งานกันได้ทั่วไป

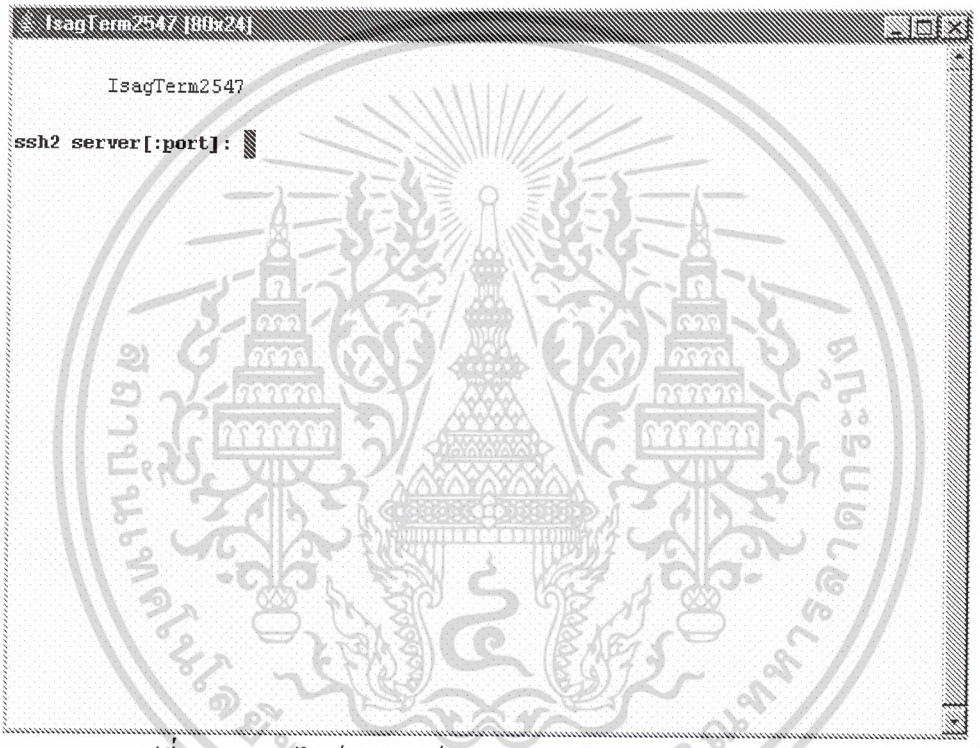
โดย เป็นรูปที่แสดงข้อมูลในการเชื่อมต่อ ซึ่งใช้โปรแกรม Ethereal ดักจับแพ็กเก็ตออกมาจะเห็นได้ว่าในส่วนของ Client นั้นยังคงสนับสนุน โพรโตคอล SSH-1 ซึ่งในปัจจุบันนี้ไม่ได้รับความนิยมในการใช้งานแล้วเนื่องจากไม่มีความยืดหยุ่นในการทำงานซึ่งได้อธิบายไว้แล้วในบทที่ 3

10.2 การทดลองใช้โปรแกรม IsagTerm 2547

โปรแกรม IsagTerm 2547 นี้ได้พัฒนามาจากโครงงานเก่าคือ IsagTerm 2542 โดยได้พัฒนาให้ใช้งานกับ โพรโตคอล SSH-2 ซึ่งมีประสิทธิภาพกว่าในการถ่ายโอนแฟ้มข้อมูล

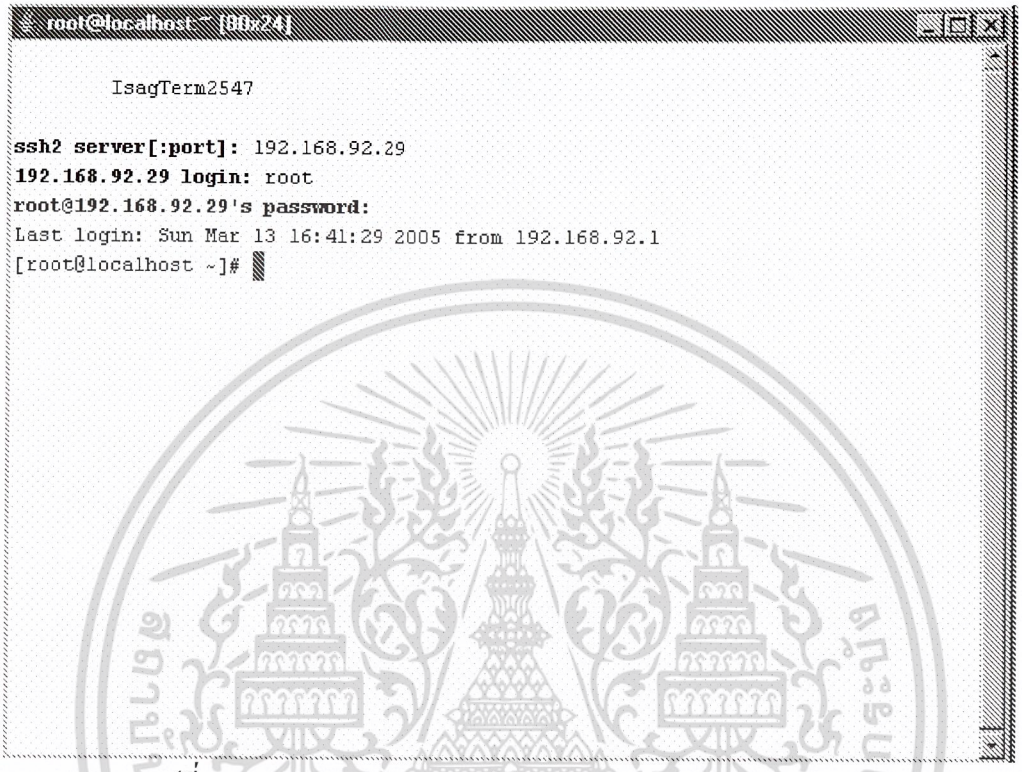
ขั้นตอนการทดลอง

1. เมื่อรันโปรแกรมก็จะพบกับตัวโปรแกรมซึ่งออกแบบมาให้ใช้งานได้ง่าย ดังรูปที่ 10-9

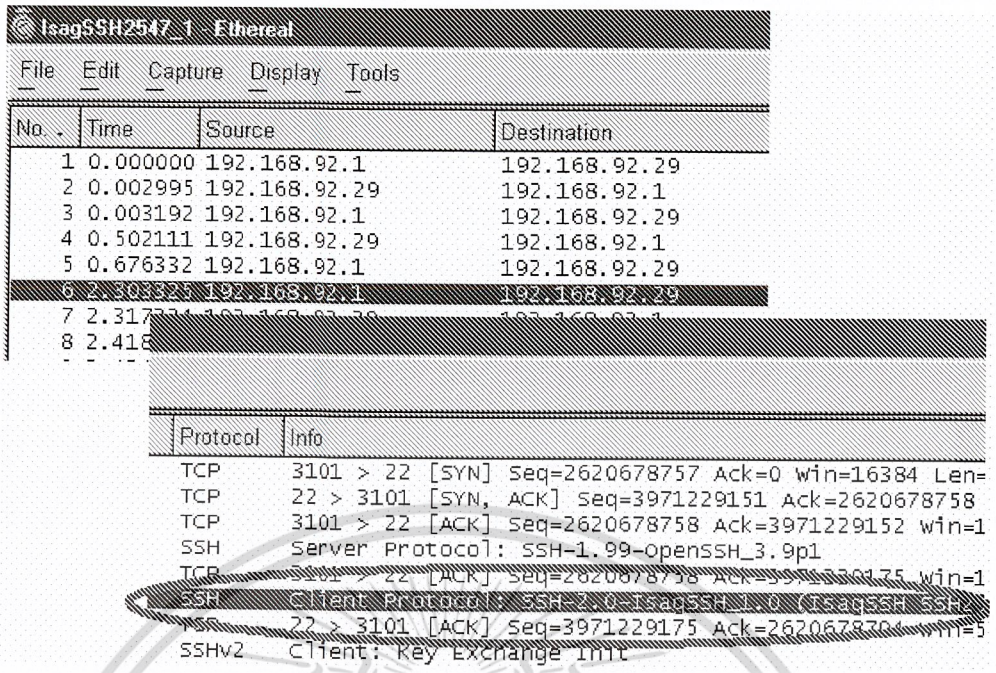


รูปที่ 10-9 แสดงไซต์เมนเจอร์ของโปรแกรม IsagTerm 2547

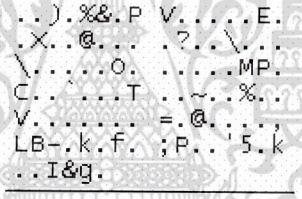
2. ป้อนค่าของ Host หรือ ip ของเซิร์ฟเวอร์ที่ต้องการติดต่อ จากนั้นก็ป้อนค่าของ Username ตามด้วยค่าของ password เมื่อทาง server (Linux Fedora core 3) พิสูจน์ตัวตนของผู้ใช้ผ่านแล้วก็จะใช้งานรับส่งข้อมูลกันได้อย่างปลอดภัย



รูปที่ 10-10 แสดงการทำงานของโปรแกรม IsagTerm 2547



รูปที่ 10-11 แสดงข้อมูลการเชื่อมต่อที่ดักจับได้

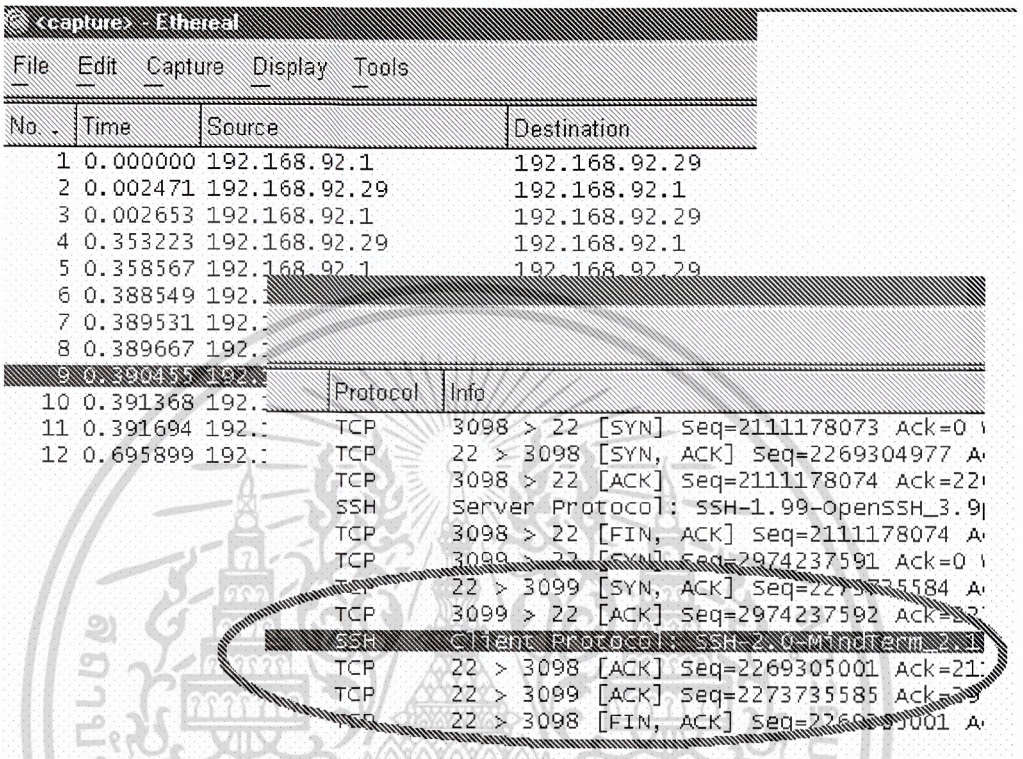


รูปที่ 10-12 แสดงการเข้ารหัสของข้อมูล

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

10.3 การทดลองใช้โปรแกรม IsagFTP

โปรแกรม IsagFTP นี้เป็นโครงงานเก่าซึ่งการทำงานนั้นยังใช้งานกับ โพรโตคอล SSH-2 ได้เช่นเดียวกับ โปรแกรม IsagTerm 2542



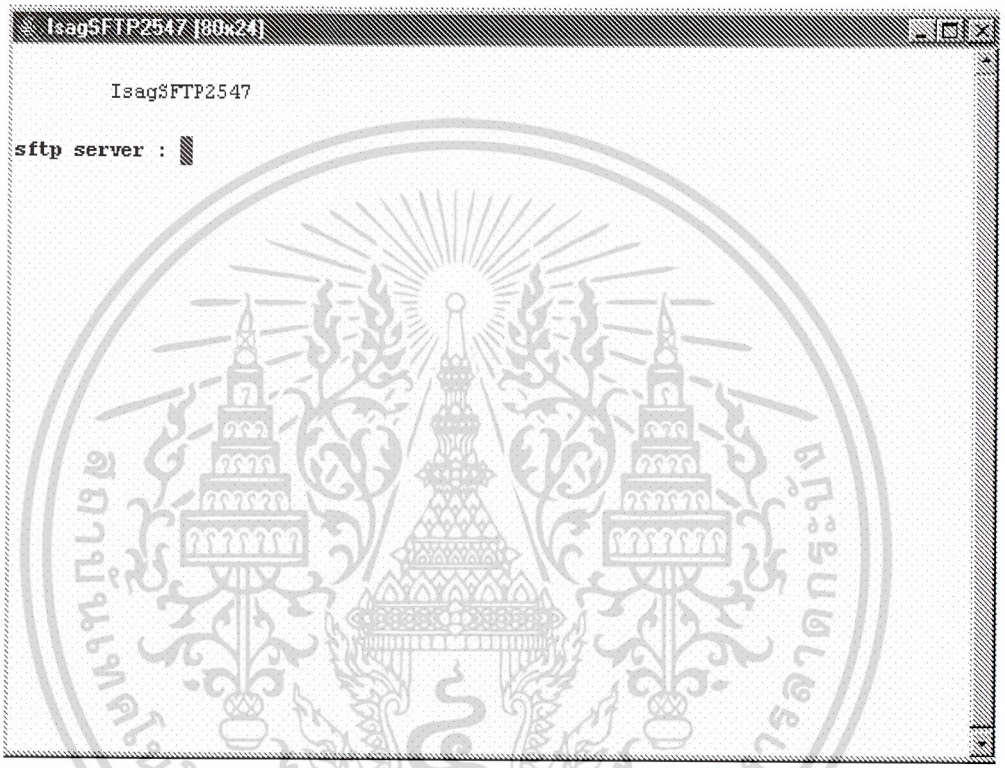
รูปที่ 10-13 แสดงข้อมูลของการเชื่อมต่อ ที่ดักจับได้

10.4 การทดลองใช้โปรแกรม IsagSFTP

ในการพัฒนาโปรแกรม IsagSFTP ขึ้นมานั้นเพื่อให้ผู้ใช้งานนั้นสามารถใช้งานในการรับส่งข้อมูลกันระหว่างเครือข่ายอย่างปลอดภัย โดยใช้ โพรโตคอล SSH-2

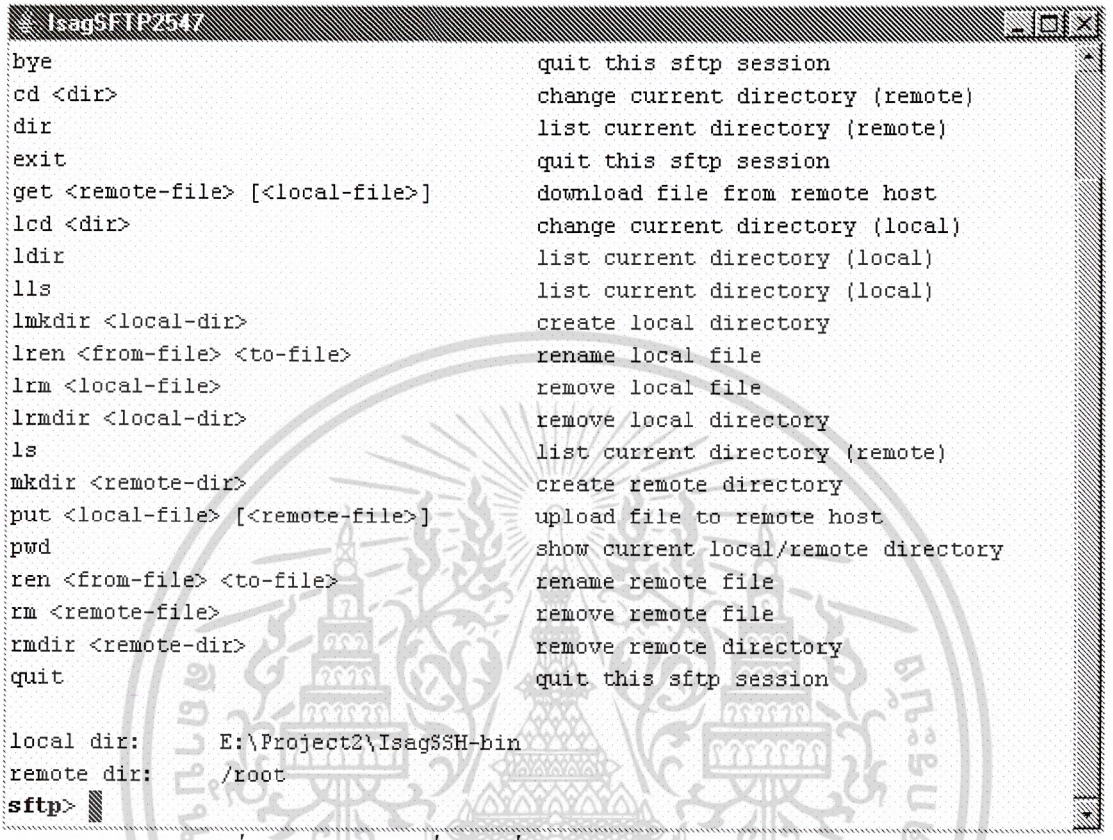
การทดลอง

1. รันโปรแกรม IsagSFTP

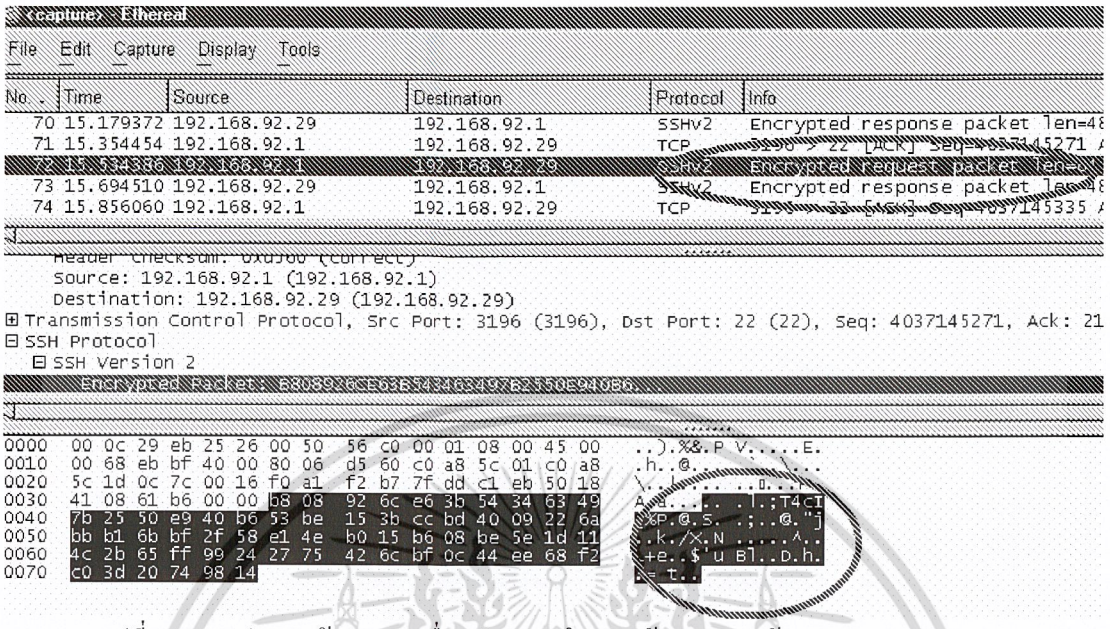


10.14 แสดงไซต์เมนเจอร์ของโปรแกรม IsagSFTP

2. ป้อน Host หรือ IP ตามขั้นตอนของโปรแกรม



รูปที่ 10-15 แสดงการเชื่อมต่อที่สมบูรณ์



รูปที่ 10-16 รูปแสดงข้อมูลการเชื่อมต่อ รวมถึง การเข้ารหัสของข้อมูล

บทที่ 11

บทวิจารณ์และสรุป

ในการพัฒนาโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัยนี้ ได้นำโพรโตคอลเอสเอสเอชมาประยุกต์ใช้ เพื่อสร้างความปลอดภัยในการติดต่อสื่อสาร โดยรองรับโพรโตคอลเอสเอสเอชเวอร์ชัน 2 ซึ่งในส่วนการรองรับเวอร์ชัน 1 นั้น จะไม่ใช้เพราะเป็นเทคโนโลยีที่ไม่มีการใช้งานแพร่หลายในปัจจุบันแล้ว โดยได้สร้างโปรแกรมในส่วนของการรับส่งข้อมูล ตามมาตรฐานของโพรโตคอลเอฟทีพี ที่ได้มีการกำหนดไว้ในอาร์เอฟซี 959 (RFC 959) ส่วนการรองรับเวอร์ชัน 2 นั้น จะพัฒนาเพื่อใช้งานร่วมกับเอสเอฟทีพี จากการศึกษาและพัฒนาโปรแกรมมาพอจะวิจารณ์และสรุปได้ดังนี้

11.1 การวิเคราะห์และสรุปมาตรฐานของโพรโตคอลเอสเอสเอช

ในปัจจุบันการรักษาความปลอดภัยของข้อมูลเป็นประเด็นสำคัญ ในการติดต่อสื่อสารของระบบเครือข่าย ถึงแม้ว่าโพรโตคอลเอสเอสเอชยังไม่ได้เป็นโพรโตคอลมาตรฐาน ทางด้านความปลอดภัย แต่ก็เป็นโพรโตคอลที่มีคนนิยมใช้เป็นจำนวนมากจนได้รับเป็นมาตรฐานดีแฟกโต (Defacto Standard) โปรแกรมประยุกต์บนทีซีพี/ไอพีที่ต้องการความปลอดภัยของข้อมูลจะเลือกใช้ และเซิร์ฟเวอร์ส่วนใหญ่มักจะติดตั้งโปรแกรมเอสเอสเอชไว้สำหรับผู้ใช้ที่ต้องการรักษาความปลอดภัยและความเป็นส่วนตัวของตน ข้อดีของโพรโตคอลเอสเอสเอช มีดังนี้

- การปลอมแปลงไอพี ไม่สามารถกระทำได้ เนื่องจากมีการตรวจสอบการเข้าใช้งานของผู้ใช้มีความปลอดภัยสูง โดยการเข้ารหัส คีย์โฮสต์และ คุกกี้ที่สุ่มขึ้นป้องกันการปลอมแปลงไอพี
- ข้อมูลผู้ใช้และรหัสผ่านของผู้ใช้นั้นจะมีความปลอดภัยในการส่งที่สูงเนื่องจากมีการเข้ารหัสที่มีประสิทธิภาพสูง ทำให้ยากต่อการถอดรหัส
- ข้อมูลที่ติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้นมีการเข้ารหัสที่หลากหลายตามความต้องการของผู้ใช้ฝั่งไคลเอ็นต์ ทำให้ยากต่อการถอดรหัสข้อมูลที่ดักจับมาได้เป็นอย่างมาก
- การโจมตีของบุคคลระหว่างกลาง (Man in the middle attack) ไม่สามารถทำได้เนื่องจากมีการใช้ คีย์โฮสต์ และคีย์เซสชัน

โพรโตคอลเอสเอสเอชนี้มีการพัฒนาต่อเนื่องโดยมี SSH Communications Security เป็นผู้ที่พัฒนาโปรแกรมและมี IETF (The Internet Engineering Task Force) เป็นผู้ดูแลเกี่ยวกับมาตรฐานอยู่ตลอด โดยในปัจจุบันมี 2 เวอร์ชัน ซึ่งถูกออกแบบเป็นอิสระต่อกัน

11.2 การวิเคราะห์และสรุปมาตรฐานของโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย (IsagSFTP)

ตัวโปรแกรมนั้นมีการทำงานเป็นแบบกราฟิก ทำให้สะดวกต่อผู้ใช้งาน และลดเวลาการเรียนรู้การใช้งานอีกด้วย อีกทั้งยังทำให้ผู้ใช้ที่ไม่มีความรู้เกี่ยวข้องกับความซับซ้อน หรือรู้เพียงเล็กน้อย สามารถใช้งานได้ดีเหมือนเช่นผู้ที่มีความรู้ การพัฒนาโปรแกรมนั้นแบ่งออกเป็น 2 ส่วนใหญ่ ๆ คือ ส่วนที่ทำตามขั้นตอนมาตรฐานของโพรโตคอลเซ็กเคียวและส่วนจัดการรับส่งข้อมูล

- การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอช 2
- การพัฒนาส่วนติดต่อกับผู้ใช้งาน เป็นแบบกราฟิก

11.2.1.ส่วนจัดการการถ่ายโอนแฟ้มข้อมูล

ได้ออกแบบตามมาตรฐานของโพรโตคอลเอฟทีพีที่ระบุไว้ในอาร์เอฟซี 959 มีความต้องการขั้นต่ำตรงกับตามที่กำหนดไว้ คือ

ชนิดของข้อมูล แอสกี

วิธีการส่ง สายข้อมูล (Stream)

โครงสร้างข้อมูล ไฟล์ , เรคอร์ด

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

และได้เพิ่มเติมคุณสมบัติต่าง ๆ เข้าไปอีก เช่น ชนิดข้อมูลไบนารี เพิ่มคำสั่งต่าง ๆ การสร้างไฟล์จดจำคอนฟิกในการติดต่อ การอัปโหลด-ดาวน์โหลดเป็นไดเรกทอรี การเลือกไฟล์เป็นกลุ่มเพื่ออัปโหลด-ดาวน์โหลด การลบทั้งไดเรกทอรี เป็นต้น

11.2.2 การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอช 2

ส่วนนี้จะทำให้โปรแกรมสามารถทำงานร่วมกับโพรโตคอลเอสเอสเอช 2 ของเซิร์ฟเวอร์ได้ ซึ่งออกแบบตามอินเทอร์เน็ต-ดราฟต์ ที่เกี่ยวข้องกับโพรโตคอลเอสเอสเอช 2 ลดข้อจำกัดหลายประการ คือ การติดเงื่อนไขตามสิทธิบัตรบนอัลกอริทึมคีย์สาธารณะ RSA ที่ใช้ในโพรโตคอลเอสเอสเอชเวอร์ชัน 1 ใช้งานด้วยเอสเอฟทีพี ซึ่งออกแบบมารองรับการเข้ารหัสข้อมูลที่มีขนาดใหญ่

11.3 แนวทางการพัฒนาต่อในอนาคต

- เพิ่มอัลกอริทึมการเข้ารหัสข้อมูลที่ยังไม่ครบถ้วน เพื่อให้ยากแก่การนำข้อมูลที่ลักลอบดักจับได้ไปทำการถอดรหัส
- สร้างฟังก์ชันในการใช้งานบนอินเทอร์เฟซแบบกราฟิกมากขึ้น เพื่อความสะดวกในการใช้งานของแต่ละผู้ใช้
- จดจำรูปแบบการใช้งานของผู้ใช้ เพื่อให้โปรแกรมเรียนรู้ และช่วยผู้ใช้งานได้
- พัฒนาให้มีการเลือกเข้ารหัสเฉพาะการเชื่อมต่อส่วนควบคุม เพื่อความรวดเร็วในการโอนย้ายข้อมูลในการเชื่อมต่อส่วนข้อมูล สำหรับผู้ที่ยอมรับความเสี่ยงในการดักจับเฉพาะเนื้อข้อมูล แต่ต้องการป้องกันการดักจับรายชื่อผู้ใช้และรหัสผ่าน