

การถ่ายโอนแฟ้มข้อมูลอย่างปลอดภัยด้วยโปรโตคอลเอสเอสเอช

SECURED FILE TRANSFERRING USING SSH



46157
152
2546

เลขหมู่.....
เลขทะเบียน.....46157
วัน, เดือน, ปี 20 ส.ค. 2546

.b.....
.i.....

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

140531761

การถ่ายโอนแฟ้มข้อมูลอย่างปลอดภัยด้วยโปรโตคอลเอสเอสเอช

SECURED FILE TRANSFERRING USING SSH



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การถ่ายโอนแฟ้มข้อมูลอย่างปลอดภัยด้วยโปรโตคอลเอสเอสเอส

SECURE FILE TRANSFERRING USING SSH

ผู้จัดทำ

1. นายภาสิ งามกมลรัตน์
- รหัสประจำตัว 41014022
2. นายกิตติศักดิ์ ถนอมธีระนันท์
- รหัสประจำตัว 41014031



การถ่ายโอนแฟ้มข้อมูลอย่างปลอดภัยด้วยโพรโตคอลเอสเอสเอสเอช

นายกาสี	งามกมลรัตน์	
นายกิตติศักดิ์	ณอมธีระนันท์	
อาจารย์ธนา	หงษ์สุวรรณ	อาจารย์ที่ปรึกษา
อาจารย์อัครเดช	วัชรภูพงษ์	อาจารย์ที่ปรึกษา
ปีการศึกษา 2544		

บทคัดย่อ

ในยุคโลกาภิวัตน์ทุกวันนี้ เทคโนโลยีสารสนเทศและอินเทอร์เน็ตมีอิทธิพลและผลกระทบอย่างยิ่งต่อชีวิตมนุษย์ในทุกๆ ด้าน การใช้งานอินเทอร์เน็ตได้เข้ามาแทนที่การติดต่อสื่อสารเดิม เรื่องที่ควรให้ความสำคัญมากที่สุดเรื่องหนึ่งคือความปลอดภัยของข้อมูลถูกส่งผ่าน แต่โปรแกรมประยุกต์ที่ทำงานบนอินเทอร์เน็ตส่วนใหญ่กลับไม่ได้คำนึงถึงความปลอดภัยเช่นนี้เท่าใดนัก

โครงการนี้ได้พัฒนาโปรแกรมเอฟทีพี ซึ่งเป็นที่นิยมใช้กันอย่างมากในการถ่ายโอนแฟ้มข้อมูลในระบบเครือข่ายให้มีความปลอดภัยมากขึ้น โดยได้ประยุกต์ใช้กับโพรโตคอลเอสเอสเอสเอชซึ่งเป็นที่ยอมรับกันอย่างแพร่หลายในเซิร์ฟเวอร์ที่เปิดให้บริการแบบระยะไกล โดยโพรโตคอลเอสเอสเอสเอชนั้นทำให้เกิดการพิสูจน์ตนที่น่าเชื่อถือ การเข้ารหัสที่ดีมีประสิทธิภาพ และการตรวจสอบความสมบูรณ์ของข้อมูลซึ่งทำให้เกิดช่องทางการติดต่อสื่อสารที่ปลอดภัย สามารถป้องกันการโจมตีได้หลายรูปแบบ

เนื่องจากความปลอดภัยเป็นประเด็นสำคัญในการพัฒนาโครงการนี้ ภาษาจาวาจึงถูกเลือกใช้ในการพัฒนาโปรแกรม อีกทั้งทำให้โปรแกรมสามารถทำงานบนระบบปฏิบัติการใดๆ ก็ได้ และเป็นภาษาเชิงวัตถุทำให้การพัฒนาเป็นไปอย่างมีประสิทธิภาพมากยิ่งขึ้น ง่ายต่อการพัฒนาต่อ โปรแกรมในโครงการนี้พัฒนาขึ้นมาเพื่อให้เกิดการใช้งานจริง โดยได้ออกแบบให้มีลักษณะการใช้งานแบบกราฟิกเพื่อให้สะดวกในการใช้งานมากยิ่งขึ้น และรองรับทั้งโพรโตคอลเอสเอสเอสเอชเวอร์ชัน 1 และเวอร์ชัน 2 ด้วย

Secured File Transferring using SSH

Kasi	Ngamkamolrat	
Kittisak	Thanomtheeranan	
Thana	Hongsuwan	Advisor
Akkradach	Watcharapupong	Advisor

ABSTRACT

Nowadays, Globalization and Internet technology become highly influenced on human life. As Internet access has become a viable replacement for traditional communications. One of the biggest challenges in using the Internet is security. But most of Internet application doesn't regard about it.

Under this project, FTP program mostly use for network file transferring is developed for more security by applying to SSH protocol, that is widely used in many servers providing remote service. Beside the SSH protocol provide strong authentication, efficiency encryption and data integrity to secure network communication defend against the attacks known today. So SSH offers a good solution for the problem of securing data sent over a public network.

Since the security is the biggest issue for this development. JAVA language is chosen to use. Additionally, it offers compatibility to support any operating system. It also provides object-oriented programming more efficient implementation. Program is developed to be more practical for all uses, which is GUI (graphic user interface). It supports both SSH1 and SSH2.

กิตติกรรมประกาศ

โครงการและปริญยานิพนธ์ฉบับนี้ เสร็จสมบูรณ์ลงด้วยดีได้ ด้วยคำแนะนำจากอาจารย์ที่ปรึกษาทั้งสองท่าน คือ อาจารย์ธนา หงษ์สุวรรณ และอาจารย์อัศรเดช วัชรภูพงษ์ ที่คอยแนะนำ เอาใจใส่และดูแลกับการทำโครงการนี้เสมอมา คณะผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษาทั้งสองท่านด้วยความเคารพเป็นอย่างสูง

ขอขอบคุณผองเพื่อนพี่น้อง ชาว ISAG ทุกท่าน ที่ได้ให้กำลังใจ ช่วยเหลือ และร่วมแก้ไขปัญหาน้อยใหญ่ต่าง ๆ ร่วมกัน ทำให้พวกเราสามารถก้าวข้ามผ่านอุปสรรคทั้งหลายด้วยดีเสมอมา และพวกเราพร้อมที่จะฝ่าฟันความยากลำบากใดๆ ด้วยกันเสมอไป

ขอขอบคุณพี่น้องที่เราได้เติบโตกลั่นแกล้งกันมา ร่วมเรียนรู้โลกกว้าง เล่นสนุก หัวเราะ ร้องไห้ อยู่ข้างกายเสมอมา เราต่างเติบโตและแข็งแกร่ง แม้จนถึงทุกวันนี้ก็ยังคงเป็นพี่น้องคนเดิมเสมอมา กำลังใจที่ให้นั้น ไม่อาจลบเลือนจากความทรงจำของข้าพเจ้าได้เลย

ขอกราบขอบพระคุณบุคคลที่สำคัญที่สุด คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง เป็นผู้ให้กำเนิดอบรมสั่งสอน ให้โอกาสในการศึกษา ให้การเอาใจใส่ ความอบอุ่น ความรัก ความหวังใยที่ประมาณมิได้ ไม่สามารถเอ่ยเป็นถ้อยคำได้เท่าเทียมเทียม ข้าพเจ้าขอระลึกพระคุณอันยิ่งใหญ่ไว้ทุกลมหายใจ และขอน้อมกราบขอบพระคุณทุกท่านมา ณ ที่นี้ด้วย

กาศิ งามกมลรัตน์

กิตติศักดิ์ วัฒนธรรณันท์

สารบัญ

บทคัดย่อ	I
ABSTRACT	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	XI
สารบัญรูปภาพ	XII
บทที่ 1 บทนำ	
1.1 ที่มาของปัญหา	1
1.2 วัตถุประสงค์ของการโครงการ	2
1.3 ขอบเขตงานโครงการ	2
1.4 ประโยชน์ที่คาดว่าจะได้รับ	2
บทที่ 2 ทีซีพี/ไอพี	3
2.1 ประวัติความเป็นมา	3
2.2 สถาปัตยกรรมของโพรโทคอล (Protocol Architecture)	3
2.2.1 ชั้นโปรแกรมประยุกต์ (Application Layer)	4
2.2.2 ชั้นทรานสปอร์ต (Transport Layer)	4
2.2.3 ชั้นอินเทอร์เน็ต (Internet Layer)	5
2.2.4 ชั้นเน็ตเวิร์กอินเทอร์เฟซ (Network Interface Layer)	5
2.3 การอ้างอิงที่อยู่ของไอพีและพอร์ต (IP Addressing and Port)	5
2.4 การห่อหุ้มแพ็กเก็ต (Packet Encapsulation)	7
2.5 อีเทอร์เน็ตเฟรม (Ethernet Frame)	8
บทที่ 3 เอฟทีพี	9
3.1 ลักษณะทั่วไป	9
3.2 แบบจำลองของเอฟทีพี	10
3.3 การแทนข้อมูล (Data Representation)	12
3.3.1 ชนิดข้อมูล (Data Types)	12
3.3.2 การควบคุมรูปแบบ (Format Control)	12
3.3.3 โครงสร้างข้อมูล (Data Structure)	12

3.3.4	วิธีการรับส่ง (Transmission Mode)	13
3.4	การฟื้นฟูจากความผิดพลาด (Error Recovery)	13
3.5	คำสั่งและการตอบกลับของเอฟทีพี	13
3.5.1	คำสั่งของเอฟทีพี (FTP Commands)	13
3.5.1.1	คำสั่งควบคุมการเข้าถึง (Access Control Commands)	13
3.5.1.2	คำสั่งพารามิเตอร์การโอนย้าย (Transfer Parameter Commands)	14
3.5.1.3	คำสั่งใช้บริการของเอฟทีพี (FTP Service Commands)	14
3.5.2	การตอบกลับของเอฟทีพี (FTP Replies)	15
3.5.2.1	รหัสการตอบกลับ	15
3.5.3	ตัวอย่างแสดงรหัสคำสั่งเอฟทีพี	17
3.5.3.1	การเชื่อมต่อ	17
3.5.3.2	การถ่ายโอน	19
3.5.3.3	การขัดจังหวะ	20
3.6	ลักษณะการทำงานของโปรแกรมเอฟทีพี	21
3.6.1	การทำงานแบบแอ็กทีฟ	21
3.6.2	การทำงานแบบพาสซีฟ	22
บทที่ 4 การเข้ารหัสและการคำนวณ		
4.1	ระบบของการเข้ารหัสข้อมูล (Cryptography System)	24
4.1.1	ระบบการเข้ารหัสข้อมูลโดยใช้คีย์เดียว (Symmetric Key)	24
4.1.2	ระบบการเข้ารหัสแบบคีย์สาธารณะ	24
4.2	รูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพี	25
4.2.1	การเข้ารหัสระดับการเชื่อมโยงข้อมูล (Link Level Encryption)	25
4.2.2	การเข้ารหัสระดับเครือข่าย (Network Level Encryption)	26
4.2.3	การเข้ารหัสระดับทรานสปอร์ต (Transport Level Encryption)	26
4.2.4	การเข้ารหัสระดับโปรแกรมประยุกต์ (Application Level Encryption)	26
4.3	การเข้ารหัสแบบ DES (Data Encryption Standard)	27
4.3.1	ประวัติและที่มาของ DES	27
4.3.2	รายละเอียดของ DES	27
4.3.3	การสลับตำแหน่งข้อมูลเริ่มต้น (Initial Permutation)	29
4.3.4	รายละเอียดของการทำฟังก์ชันแต่ละรอบ	30
4.3.5	การสร้างคีย์ (Key Generation)	35
4.3.6	การถอดรหัสข้อมูล DES	36
4.3.7	โหมด CBC (Cipher Block Chaining)	39

4.4 การเข้ารหัสแบบ 3DES (Triple DES)	40
4.4.1 Triple DES โหมด CBC (3DES CBC Mode)	41
4.5 การเข้ารหัสแบบ RSA	42
4.5.1 หลักการทำงานของ RSA	42
4.6 การคำนวณแบบ MD5	44
บทที่ 5 โพรโทคอลเอสเอสเอสเอช 1	48
5.1 ภาพรวมของโพรโทคอลเอสเอสเอสเอช 1	48
5.2 ลักษณะและโพรโทคอลของข้อมูล	49
5.2.1 ไบนารีแพ็กเก็ตโพรโทคอล (Binary Packet Protocol)	49
5.2.2 การเข้ารหัสแพ็กเก็ต (Packet Encryption)	50
5.2.3 ชนิดของข้อมูลที่เข้ารหัส (Data Type Encoding)	52
5.2.4 หมายเลขที่ซีพี/ไอพีพอร์ต (TCP/IP Port Number and Other Option)	52
5.3 รายละเอียดขั้นตอนการติดต่อ	53
5.3.1 ช่วงตรวจสอบเวอร์ชัน (Protocol Version Identification)	53
5.3.2 ช่วงการแลกเปลี่ยนคีย์ (Key Exchange)	53
5.4 ช่วงตรวจสอบชื่อและพิสูจน์ตน (Declare User Name and Authentication Phase)	56
5.4.1 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv	56
5.4.2 ตรวจสอบสิทธิ์ด้วยวิธี RSA	56
5.4.3 ตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv และ RSA	57
5.4.4 ตรวจสอบสิทธิ์จากรหัสผ่าน	57
5.5 ช่วงการเตรียมการ (Preparatory Operation)	58
5.6 เซสชันแบบอินเตอร์แอ็กทีฟและการแลกเปลี่ยนข้อมูล (Interactive Session and Exchange of Data)	58
5.7 การยกเลิกการติดต่อ (Termination of the Connection)	58
5.8 การใช้งานการฟอร์เวิร์ดพอร์ตของโพรโทคอลเอสเอสเอสเอช 1	59
5.8.1 การฟอร์เวิร์ดคิงเอกซ์ (X Forwarding)	59
5.8.2 การฟอร์เวิร์ดพอร์ตที่ซีพี/ไอพี (TCP/IP Port Forwarding)	59
บทที่ 6 โพรโทคอลเอสเอสเอสเอช 2	61
6.1 แนะนำ	61
6.2 สถาปัตยกรรม	62
6.2.1 คีย์โฮสต์ (Host Key)	62
6.2.2 ความสามารถในการขยาย (Extensibility)	63
6.2.3 การกำหนดนโยบาย (Policy Issues)	63

6.2.4 คุณสมบัติความปลอดภัย (Security Properties)	64
6.2.5 ขนาดแพ็กเก็ตและโอเวอร์เฮด (Packet Size and Overhead)	64
6.3 การแทนชนิดข้อมูล (Data Type Representation) ที่ใช้ในโพรโทคอลเอสเอสเอช	65
6.4 การตั้งชื่ออัลกอริทึม (Algorithm Naming)	67
6.5 หมายเลขเมสเสจ (Message Number)	67
6.6 การพิจารณาของ IANA	68
6.7 พิจารณาความปลอดภัย	68
 บทที่ 7 โพรโทคอลชั้นทรานสปอร์ตเอสเอสเอช 2	 70
7.1 แนะนำ	70
7.2 การก่อตั้งการเชื่อมต่อ (Connection Setup)	70
7.2.1 การใช้งานบน ทีซีพี/ไอพี (Use Over TCP/IP)	70
7.2.2 การแลกเปลี่ยนเวอร์ชันของโพรโทคอล (Protocol Version Exchange)	70
7.2.3 ความเข้ากันได้กับเวอร์ชันเอสเอสเอช 2 ที่เก่ากว่า (Compatibility With Old SSH Versions)	71
7.2.3.1 โคลเอนด์เก่า เซิร์ฟเวอร์ใหม่	71
7.2.3.2 โคลเอนด์ใหม่ เซิร์ฟเวอร์เก่า	72
7.3 โพรโทคอลแพ็กเก็ตแบบไบนารี (Binary Packet Protocol)	72
7.3.1 ความยาวแพ็กเก็ตสูงสุด (Maximum Packet Length)	73
7.3.2 การบีบอัด (Compression)	73
7.3.3 การเข้ารหัส (Encryption)	73
7.3.4 ความสมบูรณ์ของข้อมูล (Data Integrity)	75
7.3.4 วิธีการแลกเปลี่ยนคีย์ (Key Exchange Methods)	76
7.3.5 อัลกอริทึมคีย์สาธารณะ (Public Key Algorithms)	77
7.4 การแลกเปลี่ยนคีย์ (Key Exchange)	79
7.4.1 การเจรจาอัลกอริทึม (Algorithm Negotiation)	80
7.4.2 เอาท์พุทที่ได้จากการแลกเปลี่ยนคีย์ (Output from Key Exchange)	82
7.4.3 การนำคีย์ไปใช้งาน (Taking Keys Into Use)	83
7.5 การแลกเปลี่ยนคีย์ใหม่ (Key Re-Exchange)	84
7.6 การร้องขอบริการ (Service Request)	84
7.7 เมสเสจเพิ่มเติม (Additional Messages)	85
7.7.1 เมสเสจยกเลิกการเชื่อมต่อ (Disconnection Message)	85
7.7.2 เมสเสจข้อมูลที่เพิกเฉย (Ignored Data Message)	86
7.7.3 เมสเสจดีบั๊ก (Debug Message)	86
7.7.4 เมสเสจสงวน (Reserved Message)	87

7.8	สรุปหมายเลขแอสเซจ	87
7.9	พิจารณาด้านความปลอดภัย	87
บทที่ 8	โพรโทคอลการพิสูจน์ตนเอสเอสเอช 2	89
8.1	แนะนำ	89
8.2	เฟรมเวิร์กโพรโทคอลการพิสูจน์ตน (The Authentication Protocol Framework)	89
8.2.1	การร้องขอพิสูจน์ตน (Authentication Request)	90
8.2.2	การตอบสนองต่อการร้องขอพิสูจน์ตน (Responses to Authentication Requests)	90
8.2.3	การร้องขอการพิสูจน์ตนแบบ "none" (The "none" Authentication Request)	91
8.2.4	ความสำเร็จของการพิสูจน์ตนผู้ใช้ (Completion of User Authentication)	91
8.2.5	ข้อความแบนเนอร์ (Banner Message)	92
8.3	หมายเลขแอสเซจของโพรโทคอลพิสูจน์ตน (Authentication Protocol Message Numbers)	92
8.4	วิธีการพิสูจน์ตนด้วยคีย์สาธารณะ (Public Key Authentication Method : publickey)	93
8.5	วิธีการพิสูจน์ตนด้วยรหัสลับ (Password Authentication Method : password)	95
8.6	การพิสูจน์ตนบนพื้นฐานของโฮสต์ (Host-Based Authentication : hostbased)	96
8.7	พิจารณาด้านความปลอดภัย	98
บทที่ 9	โพรโทคอลเชื่อมต่อเอสเอสเอช 2	99
9.1	แนะนำ	99
9.2	การร้องขอทั่วไป (Global Requests)	99
9.3	กลไกของแชนเนล (Channel Mechanism)	100
9.3.1	การเปิดแชนเนล (Opening a Channel)	100
9.3.2	การถ่ายโอนข้อมูล (Data Transfer)	101
9.3.3	การปิดแชนเนล (Closing a Channel)	102
9.3.4	การร้องขอกำหนดลักษณะเฉพาะของแชนเนล (Channel-Specific Requests)	103
9.4	เซสชันแบบอินเทอร์แอคทีฟ (Interactive Sessions)	104
9.4.1	การเปิดเซสชัน (Opening Session)	104
9.4.2	การร้องขอเทอร์มินอลเทียม (Requesting a Pseudo-Terminal)	104
9.4.3	การฟอร์เวิร์ด X11 (X11 Forwarding)	105
9.4.3.1	การร้องขอฟอร์เวิร์ด X11 (Requesting X11 Forwarding)	105
9.4.3.2	แชนเนล X11 (X11 Channels)	106
9.4.4	การผ่านค่าตัวแปรสภาพแวดล้อม (Environment Variable Passing)	106
9.4.5	การสตาร์ทเชลล์หรือคอมมานด์ (Starting a Shell or a Command)	107
9.4.6	การถ่ายโอนข้อมูลบนเซสชัน (Session Data Transfer)	108

9.4.7	แมสเสจเปลี่ยนขนาดของวินโดว (Window Dimension Change Message)	108
9.4.8	โพล์คอนโทรลฝั่งโลคอล (Local Flow Control)	108
9.4.9	ซิกแนล (Signals)	109
9.4.10	การคืนค่าสถานะจบ (Returning Exit Status)	110
9.5	การฟอร์เวิร์ดพอร์ตที่ซีพี/ไอพี (TCP/IP Port Forwarding)	110
9.5.1	การร้องขอฟอร์เวิร์ดพอร์ต (Requesting Port Forwarding)	110
9.5.2	แชนเนลฟอร์เวิร์ดที่ซีพี/ไอพี (TCP/IP Forwarding Channels)	111
9.6	การเข้ารูปแบบโหมคของเทอร์มินอล (Encoding of Terminal Modes)	112
9.7	สรุปหมายเลขแมสเสจ	115
9.8	พิจารณาด้านความปลอดภัย	115
บทที่ 10	ภาษาจาวา	117
10.1	ประวัติของภาษาจาวา	117
10.2	คุณสมบัติของภาษาจาวา	119
10.3	คอมไพล์ชัน และอินเทอร์พรีชัน (Compilation and interpretation)	121
10.4	เครื่องจักรสมมติของจาวา (Java Virtual Machine)	124
บทที่ 11	การออกแบบและการสร้าง	127
11.1	ภาพรวมของการออกแบบ	128
11.1.1	โพรโตคอลเอพีที	128
11.1.2	โพรโตคอลเอสเอสเอส	129
11.2	การออกแบบโปรแกรม	129
11.3	Socket APIs	130
11.3.1	ภาพรวมของฟังก์ชันต่างๆ ที่ใช้สำหรับการติดต่อกับผู้ให้บริการ	134
11.4	รายละเอียดของการพัฒนา	135
11.4.1	การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอส 1	135
11.4.1.1	ขั้นตอนการสร้างส่วนโปรแกรมเอพีที	136
11.4.1.2	ขั้นตอนการสร้างส่วนที่ทำการฟอร์เวิร์ดพอร์ตด้วยโพรโตคอลเอสเอสเอส 1	137
11.4.2	การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอส 2	139
บทที่ 12	การทดลองและผลการทดลอง	142
12.1	จุดประสงค์ของการทดลอง	142
12.2	การทดลองการใช้งานของโปรแกรม	142
12.2.1	การทดลองใช้โปรแกรมเอพีทีเดิม	142

12.2.2 การทดลองใช้งานโปรแกรม IsagFTP	148
12.2.3 การทำงานในระบบปฏิบัติการต่างๆ	153
บทที่ 13 บทวิจารณ์และสรุป	154
13.1 การวิเคราะห์และสรุปมาตรฐานของโพรโตคอลเอสเอสเอสเอช	154
13.2 การวิเคราะห์และสรุปมาตรฐานของโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัย	155
13.2.1 ส่วนรองรับโพรโตคอลเอสเอสเอสเอช 1	155
13.2.1.1 ส่วนจัดการการถ่ายโอนแฟ้มข้อมูล	155
13.2.1.2 ส่วนจัดการตามขั้นตอนของโพรโตคอลเอสเอสเอสเอช 1	155
13.2.2 การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอสเอช 2	156
13.3 แนวทางการพัฒนาต่อในอนาคต	156
ภาคผนวก	
ภาคผนวก ก รายละเอียดคำสั่งเอพีทีพี	157
ภาคผนวก ข รายละเอียดการตอบกลับของเอพีทีพี	170
ภาคผนวก ค รายละเอียดของแพ็คเกจโพรโตคอลเอสเอสเอสเอช	179
บรรณานุกรม	190



สารบัญตาราง

ตารางที่ 2.1	แสดงการแบ่งคลาส (Class) ของที่อยู่ไอพี	6
ตารางที่ 4.1	แสดงการทำ Permutation ของ DES	32
ตารางที่ 4.2	แสดงขั้นตอนใน S-box ทั้ง 8 ชุด	35
ตารางที่ 4.3	แสดงการสร้างคีย์ในแต่ละรอบฟังก์ชัน	36
ตารางที่ 5.1	แสดงวิธีการเข้ารหัสที่โพรโตคอลเอสเอสเอส 1 รองรับ	51
ตารางที่ 6.1	แสดงตัวอย่างการแทนค่าด้วย mpint	66
ตารางที่ 6.2	แสดงตัวอย่างการแทนค่าด้วย name-list	66



สารบัญรูปภาพ

รูปที่ 2-1 แสดงความสัมพันธ์ของชั้นต่างๆของโพรโตคอลทีซีพี/ไอพี	4
รูปที่ 2-2 การห่อหุ้มแพ็กเก็ต	7
รูปที่ 2-3 เฟรมอีเทอร์เน็ต	8
รูปที่ 3-1 การแปลงคำสั่งเอฟทีพี	9
รูปที่ 3-2 แบบจำลองการทำงานของเอฟทีพี	10
รูปที่ 3-3 แสดงการโอนย้ายข้อมูลระหว่างเซิร์ฟเวอร์	11
รูปที่ 3-4 การสร้างส่วนเชื่อมต่อข้อมูล	11
รูปที่ 3-5 แสดงการตอบกลับแบบหลายบรรทัดของคำสั่ง HELP	17
รูปที่ 3-6 ตัวอย่างการทำงานของเอฟทีพี	18
รูปที่ 3-7 แสดงการเชื่อมต่อระหว่างไคลเอ็นต์และเซิร์ฟเวอร์	18
รูปที่ 3-8 แสดงการโอนย้ายข้อมูลในเอฟทีพี	20
รูปที่ 3-9 แสดงการขัดจังหวะการโอนย้ายจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์	20
รูปที่ 3-10 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบแอ็กทีฟ	21
รูปที่ 3-11 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบพาสซีฟ	23
รูปที่ 4-1 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์เดียว	24
รูปที่ 4-2 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์สาธารณะ	25
รูปที่ 4-3 การเข้ารหัสในระดับการเชื่อมโยงข้อมูล	25
รูปที่ 4-4 การเข้ารหัสระดับเครือข่าย	26
รูปที่ 4-5 การเข้ารหัสระดับทรานสปอร์ต	26
รูปที่ 4-6 แสดงการเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดลของทีซีพี/ไอพี	27
รูปที่ 4-7 แสดงการขั้นตอนการทำงานของ DES	29
รูปที่ 4-8 แสดงการเข้ารหัส DES ในแต่ละครั้ง(ทั้งหมดทำ 16 ครั้ง)	31
รูปที่ 4-9 แสดงการคำนวณ $f(R,K)$	33
รูปที่ 4-10 แสดงการทำ Permutation Choice	35
รูปที่ 4-11 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES	38
รูปที่ 4-12 แสดงการเข้ารหัสและถอดรหัสของ DES CBC	39
รูปที่ 4-13 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES	40
รูปที่ 4-14 แสดงการเข้ารหัสและถอดรหัสแบบ Triple DES โหมด inner CBC	41
รูปที่ 4-15 แสดงการเข้ารหัสและถอดรหัสแบบ Triple DES โหมด outer CBC	41
รูปที่ 4-16 แสดงขั้นตอนการทำ Public – Key Cryptosystem	42
รูปที่ 4-17 แสดงการเข้ารหัสแบบ RSA	43

รูปที่ 4-18 แสดงการคำนวณแบบ MD5	45
รูปที่ 4-19 กระบวนการทำในหนึ่งบล็อกของ MD5	46
รูปที่ 4-20 แสดงการทำหนึ่ง operation ของ MD5 [abcd k s i]	46
รูปที่ 5-1 แสดงขั้นตอนของโพรโตคอลเอสเอสเอช 1	49
รูปที่ 5-2 แสดงฟิลด์ต่างๆในแพ็กเก็ตของระบบ Binary Packet Protocol	50
รูปที่ 5-3 แสดงการเข้ารหัสและถอดรหัสแบบ DES โหมด CBC	51
รูปที่ 5-4 แสดงวิธีการเข้ารหัสและถอดรหัสแบบ 3DES โหมด CBC	52
รูปที่ 5-5 แสดงการตรวจสอบเวอร์ชัน Identification	53
รูปที่ 5-6 แสดงการคำนวณหาไอดีของเซสชัน	54
รูปที่ 5-7 แสดงการเข้ารหัสคีย์เซสชันก่อนส่งให้เซิร์ฟเวอร์	54
รูปที่ 5-8 แสดงตัวอย่างของข้อมูลที่จะทำการเข้ารหัส PKCS#1	55
รูปที่ 5-9 ขั้นตอนการส่งของแพ็กเก็ตต่างๆในช่วงการแลกเปลี่ยนคีย์	55
รูปที่ 5-10 แสดงการทำงานของการทำงานพอร์ตแบบโตคอล	60
รูปที่ 5-11 แสดงการทำงานของการทำงานพอร์ตแบบรีโมด	60
รูปที่ 6-1 การจัดวางตัวของโพรโตคอลหลักภายในโพรโตคอลเอสเอสเอช 2	61
รูปที่ 10-1 แสดงการการคอมไพล์โปรแกรม	122
รูปที่ 10-2 แสดงการทำงานของอินเทอร์พรีเตอร์	122
รูปที่ 10-3 การทำงานของโปรแกรม p-code	123
รูปที่ 10-4 แสดงการทำงานของเครื่องจักรสมมติ	124
รูปที่ 10-5 การทำงานด้วยเครื่องจักรสมมติที่จำลองโดยจาวาอินเทอร์พรีเตอร์	125
รูปที่ 11-1 ภาพรวมของโปรแกรมนองจากผู้ใช้	129
รูปที่ 11-2 รูปแบบของโปรแกรมนองจากการใช้งาน	130
รูปที่ 11-3 ผู้ใช้บริการทำการเปิดช็อกเก็ต	131
รูปที่ 11-4 ผู้ให้บริการทำการค้นหาพอร์ตและทำการเปิดพอร์ต	132
รูปที่ 11-5 แสดงการสร้างเส้นทางติดต่อ (Pipe line) และการสร้างบัฟเฟอร์	133
รูปที่ 11-6 แสดงการปิดช็อกเก็ต	134
รูปที่ 11-7 แสดงลำดับการจัดการกับช็อกเก็ตบนทีซีพี/ไอพี	135
รูปที่ 11-8 โครงสร้างของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น	136
รูปที่ 11-9 แสดงการทำงานของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น	136
รูปที่ 11-10 แสดงการใช้งานของพอร์ตฟอร์เวิร์ด	138
รูปที่ 11-11 แสดงการทำงานของส่วนรองรับโพรโตคอลเอสเอสเอช 1 ที่ได้พัฒนาขึ้น	139
รูปที่ 11-12 แสดงการทำงานของส่วนรองรับโพรโตคอลเอสเอสเอช 2 ที่ได้พัฒนาขึ้น	140
รูปที่ 12-1 โปรแกรมเอฟทีพีที่มากับระบบปฏิบัติการวินโดวส์	143
รูปที่ 12-2 แสดงข้อมูลในการเชื่อมต่อส่วนควบคุมของการถ่ายโอนเพิ่มข้อมูลที่ดักจับได้	144

รูปที่ 12-3 แสดงรายการไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนเพิ่มข้อมูล	145
รูปที่ 12-4 แสดงข้อมูลไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนเพิ่มข้อมูล	145
รูปที่ 12-5 โปรแกรมเอฟทีพีแบบกราฟิก CuteFTP	146
รูปที่ 12-6 แสดงข้อมูลในการเชื่อมต่อส่วนควบคุมของการถ่ายโอนเพิ่มข้อมูลที่ดักจับได้	146
รูปที่ 12-7 แสดงรายการไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนเพิ่มข้อมูล	147
รูปที่ 12-8 แสดงข้อมูลไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนเพิ่มข้อมูล	147
รูปที่ 12-9 ไซต์เมนเจอร์ของโปรแกรม IsagFTP	148
รูปที่ 12-10 วินโดว์กำหนดค่าคอนฟิกการติดต่อกับเซิร์ฟเวอร์	149
รูปที่ 12-11 แสดงอัลกอริทึมการแลกเปลี่ยนคีย์ที่สามารถเลือกใช้ได้	149
รูปที่ 12-12 แสดงอัลกอริทึมการเข้ารหัสที่รองรับ และความเป็นอิสระต่อทิศทาง	150
รูปที่ 12-13 แสดงอัลกอริทึมแฮชตรวจสอบความสมบูรณ์ข้อมูลที่รองรับ และความเป็นอิสระต่อทิศทาง	150
รูปที่ 12-14 แสดงอัลกอริทึมการบีบอัดที่รองรับ และความเป็นอิสระต่อทิศทาง	151
รูปที่ 12-15 ไซต์เมนเจอร์ที่ได้รับการกำหนดเซิร์ฟเวอร์ใหม่เพิ่ม	151
รูปที่ 12-15 วินโดว์สำหรับการถ่ายโอนเพิ่มข้อมูลของโปรแกรม IsagFTP	152
รูปที่ 12-16 ไดอะแกรมบล็อกแสดงความสำเร็จในการถ่ายโอนเพิ่มข้อมูล	152
รูปที่ 12-17 ข้อความดีบักการทำงานของโปรแกรม IsagFTP	153

บทที่ 1

บทนำ

1.1 ที่มาของปัญหา

อย่างที่ทราบกันดีแล้วว่า โพรโตคอลทีซีพี/ไอพีพัฒนาขึ้นมาโดยไม่ได้คำนึงถึงหลักความปลอดภัยของเนื้อข้อมูล ข้อมูลที่รับส่งกันในโพรโตคอลทีซีพี/ไอพีนั้นเป็นข้อมูลที่สามารถลักลอบบันทึก (tapping หรือ eavedropping) โดยเครื่องที่อยู่บนเครือข่าย โดยเฉพาะอย่างยิ่งเครือข่ายระดับข่ายท้องถิ่น เช่น อีเทอร์เน็ต (Ethernet) หรือโทกเกนริง (Token Ring) เนื่องจากข้อมูลดังกล่าวนี้มีลักษณะเป็นข้อความธรรมดา ที่ไม่ได้มีการเข้ารหัสแต่อย่างใดสามารถดักจับได้ และเนื่องจากที่เครือข่ายอินเทอร์เน็ตนั้นมีขนาดใหญ่ การส่งข้อมูลไปบนอินเทอร์เน็ตจึงต้องอาศัยเราเตอร์ ในการหาเส้นทางเพื่อให้ข้อมูลสามารถส่งต่อไปเรื่อยๆ จนถึงจุดหมายที่ต้องการได้ จะเห็นได้ว่าข้อมูลที่ส่งไปยังอาจถูกดักจับ ปลอมแปลงได้โดยง่าย

ในปัจจุบันนี้มีการขยายตัวของการใช้อินเทอร์เน็ตในชีวิตประจำวันประกอบกับความเป็นจริงที่ว่าระบบคอมพิวเตอร์มีความจำเป็นที่ต้องเกี่ยวข้องกับระบบเน็ตเวิร์กมากขึ้นเรื่อยๆ มีผลทำให้มีการเพิ่มของการละเมิดความเป็นส่วนตัวของผู้ใช้คอมพิวเตอร์เพิ่มขึ้นเป็นเงาตามตัว การแก้ไขปัญหาในจุดนี้ไม่สามารถแก้ได้ในตัวของระบบหรือจุดใหญ่ๆ หากแต่ขึ้นอยู่กับจุดเล็กๆ ที่ประกอบกันเป็นระบบ

หากมองจากภาพรวมแล้วการใช้งานกับเครื่องเซิร์ฟเวอร์จากระยะไกลผ่านทางเครือข่ายหรือที่เรียกว่ารีโมตล็อกอิน (remote login) นั้นมีการใช้งานกันอย่างแพร่หลาย มีโปรแกรมต่างๆ ที่ใช้งานกันมากเช่นโปรแกรมเอฟทีพี (FTP) , เทลเน็ต (Telnet) จากการส่งข้อมูลในโปรแกรมดังกล่าวเป็นข้อมูลชนิดเพนเท็กซ์ (Plain text) ที่ไม่มีการเข้ารหัสข้อมูลทำให้ไม่มีความปลอดภัยในการใช้งาน เนื่องจากอาจโดนผู้ที่ไม่ประสงค์ดีลักลอบดักหรือปลอมแปลงข้อมูลของเราได้ ดังนั้นเพื่อเป็นการป้องกันพฤติกรรมดังกล่าว จึงต้องมีการเข้ารหัสข้อมูลที่ส่ง ในที่นี้ใช้โพรโตคอลเอสเอสเอสเอสที่ได้มีการกำหนดเป็นมาตรฐานใช้งานกันอย่างแพร่หลาย เพราะมีวิธีการพิสูจน์ตนและมีวิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพก่อนติดต่อและส่งข้อมูล แม้ว่าผู้ที่ไม่ประสงค์ดีได้รับข้อมูลเหล่านั้นไปก็ไม่อาจทราบได้ว่าเป็นข้อมูลอะไร

ยิ่งไปกว่านั้น การใช้งานแบบคอมมานโดไลน์นั้นไม่สะดวกสำหรับผู้ใช้งาน โดยเฉพาะผู้ที่เพิ่งเริ่มใช้งานใหม่ๆ โครงการนี้จะนำโปรแกรมเอฟทีพีประยุกต์ใช้กับโพรโตคอลเอสเอสเอสเอส โดยมีรูปแบบการใช้งานแบบกราฟิก เพื่อความสะดวกของผู้ใช้งานและความปลอดภัยของข้อมูลจากการถูกลักลอบโจมตีของผู้ไม่ประสงค์ดี โดยเฉพาะอย่างยิ่งรหัสผ่านของผู้ใช้

1.2 วัตถุประสงค์ของโครงการ

- เพื่อศึกษาแนวทางในการถ่ายโอนเพิ่มข้อมูลแบบปลอดภัย
- เพื่อสร้างโปรแกรมใช้งานจริงที่สามารถถ่ายโอนเพิ่มข้อมูลที่มีความปลอดภัย

1.3 ขอบเขตโครงการ

- สร้างโปรแกรมถ่ายโอนเพิ่มข้อมูล ที่ใช้โพรโทคอลเอสเอสเอสเอชเพื่อเข้ารหัสข้อมูลที่ติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์
- พัฒนาโปรแกรมภาษาจาวาเพื่อให้โปรแกรมที่พัฒนาสามารถใช้งานได้ในทุกระบบปฏิบัติการ
- พัฒนาโปรแกรมให้มีลักษณะการใช้งานแบบกราฟิกเพื่อความสะดวกในการใช้งาน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- เพิ่มความปลอดภัยให้กับการใช้งานเอฟทีพี โดยป้องกันการตีความหมายของการดักบันทึกข้อมูล ที่รับส่งอยู่บนโปรแกรมเอฟทีพี
- เพื่อกระตุ้นให้เห็นถึงความสำคัญของการรักษาความปลอดภัย
- สร้างโปรแกรมที่จะจัดการการใช้งานเอฟทีพีให้มีความปลอดภัย



บทที่ 2

ทีซีพี/ไอพี

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

ในปี ค.ศ. 1969 หน่วยงาน ARPA (Advance Research Project Agency) ซึ่งเป็นหน่วยงานหนึ่งในกระทรวงกลาโหมของสหรัฐอเมริกาได้ทดลองเครือข่ายแบบสลับแพ็กเก็ต (packet switching network) โดยเรียกว่า เครือข่ายอาร์พานีต (Arpanet) ซึ่งเป็นเครือข่ายแบบสลับแพ็กเก็ตแบบแรกในโลก เพื่อใช้สำหรับการศึกษารวมถึงมีบทบาทสำคัญทางทหาร โดยเฉพาะอย่างยิ่งสงครามโลกครั้งที่ 1

ต่อมาในปี ค.ศ. 1975 เครือข่ายอาร์พานีตได้ถูกใช้งานเป็นเครือข่ายปฏิบัติงานจริงโดยอยู่ในความรับผิดชอบของหน่วยงานสื่อสารทางทหาร (Defense Communication Agency) ซึ่งในช่วงนี้เอง โพรโตคอลทีซีพี/ไอพี (TCP/IP protocol) ได้เริ่มถูกพัฒนาขึ้น

โพรโตคอลทีซีพี/ไอพี ถูกกำหนดให้เป็นมาตรฐานทางทหาร (Military Standards) ในปี ค.ศ.1993 โดยกำหนดให้ทุกเครื่องที่ต้องการเชื่อมโยงกับเครือข่ายนี้ จะต้องเปลี่ยนแปลงโพรโตคอลให้เป็นไปตามโพรโตคอลทีซีพี/ไอพี อาร์พานีตเดิมถูกแบ่งเป็น 2 ส่วนคือ มิลเน็ต (Milnet) สำหรับใช้งานทางทหาร และอาร์พานีตที่มีขนาดเล็กลง โดยคำว่าอินเทอร์เน็ตยังได้รวมเอาเครือข่ายต่างๆ เข้ามาเป็นส่วนหนึ่งของเครือข่ายเช่น เอ็นเอฟไอเอสเน็ต (NSFnet) เป็นต้น

หลังจากที่อินเทอร์เน็ตถูกใช้งานเป็นเครือข่ายที่ใหญ่ที่สุดในโลก ทำให้การใช้งานโพรโตคอลทีซีพี/ไอพีแพร่หลายและเป็นโพรโตคอลที่สำคัญและมีการใช้งานมากที่สุดโพรโตคอลหนึ่ง

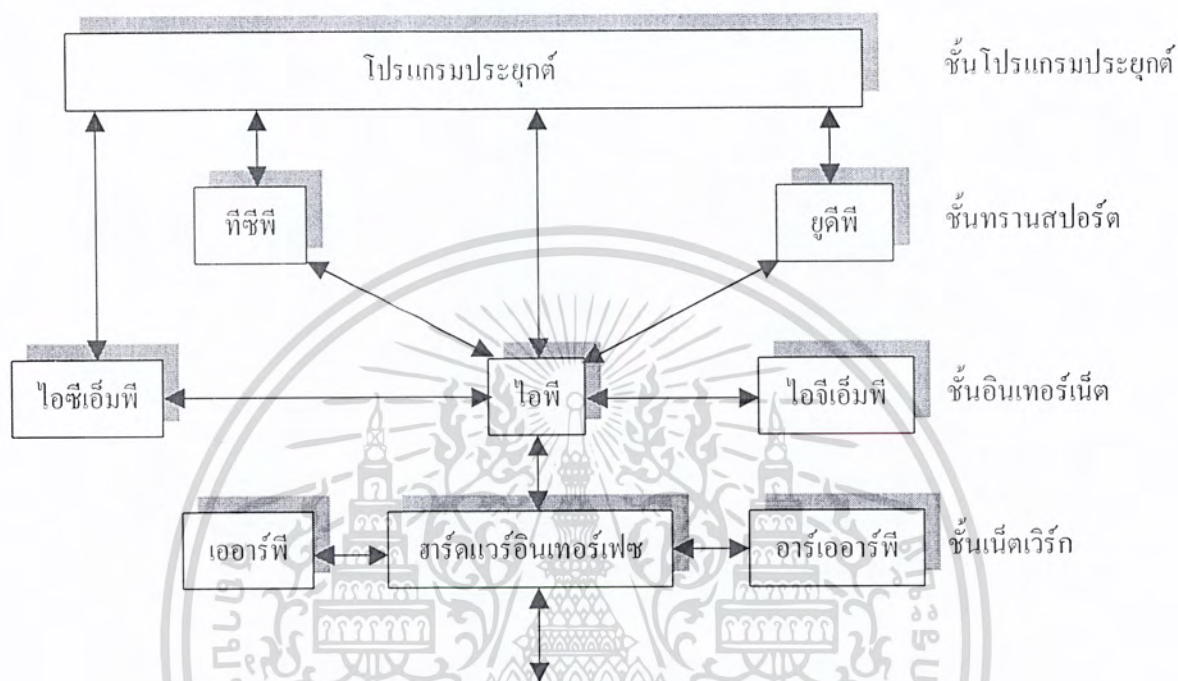
2.2 สถาปัตยกรรมของโพรโตคอล (Protocol Architecture) (Hunt,1992)

โพรโตคอลทีซีพี/ไอพี เป็นโพรโตคอลที่ได้รับความนิยมไม่ใช่ว่าอินเทอร์เน็ต หรือทางทหารสหรัฐใช้ แต่เนื่องจากคุณสมบัติที่ดีของโพรโตคอล ได้แก่

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

ชั้นเลเยอร์ของโพรโตคอลทีซีพี/ไอพี มี 4 ชั้น แบ่งจากชั้นบนลงล่างดังนี้

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



รูปที่ 2-1 แสดงความสัมพันธ์ของชั้นต่างๆของโพรโตคอลทีซีพี/ไอพี

2.2.1 ชั้นโปรแกรมประยุกต์ (Application Layer)

ชั้นโปรแกรมประยุกต์เป็นชั้นของโพรโตคอลสำหรับโปรแกรมประยุกต์บนเครือข่ายทีซีพี/ไอพี รูปแบบของโปรแกรมประยุกต์ในชั้นนี้เป็นลักษณะของผู้ให้บริการและผู้ให้บริการ (client / server model) นั่นคือการใช้งานโปรแกรมประยุกต์หนึ่งๆจะมีกระบวนการหนึ่งทำหน้าที่เป็น กระบวนการขอรับบริการ (Client process) และกระบวนการสำหรับให้บริการ (server process) ผ่านเครือข่ายโดยมีโพรโตคอลสำหรับโปรแกรมประยุกต์ร่วมกัน โปรแกรมประยุกต์ต่างๆบนโพรโตคอลทีซีพี/ไอพี เช่น เทลเน็ต (Telnet) , เอฟทีพี (FTP) , เอสเอ็มทีพี (SMTP) เป็นต้น

2.2.2 ชั้นทรานสปอร์ต (Transport Layer)

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

ทีซีพี มีชื่อเต็มว่า ทรานสมิซชันคอนโทรลโพรโตคอล (Transmission Control Protocol) เป็นโพรโตคอลที่ต้องมีการติดต่อก่อนก่อนของผู้รับส่งข้อมูล (Connection-oriented) ซึ่งทำให้เป็นโพรโตคอลที่

มีความเชื่อถือได้ (Reliable) ของการรับส่ง ยูติพี มีชื่อเต็มว่า ยูสเซอร์ดาต้าแกรมโพรโทคอล (User Datagram Protocol) เป็นโพรโทคอลที่มีการทำงานง่ายกว่าทีซีพี การรับส่งข้อมูลมีลักษณะเป็น ดาต้าแกรม (Datagram) ข้อมูลที่รับส่งไม่จำเป็นจะต้องมีการติดต่อกันก่อน (connectionless) ทำให้การรับส่งข้อมูลขาดความน่าเชื่อถือ

2.2.3 ชั้นอินเทอร์เน็ต (Internet Layer)

ชั้นอินเทอร์เน็ตมีชื่อเรียกอีกอย่างหนึ่งว่า ชั้นเครือข่าย (network layer) ในชั้นนี้โพรโทคอลที่สำคัญคือ ไอพี (IP) อันเป็นหัวใจของชุดโพรโทคอลทีซีพี/ไอพี หน้าที่สำคัญคือการส่งข้อมูล (Route) ไปตามจุดต่างๆในเครือข่าย นอกจากไอพีแล้วในชั้นนี้ยังมีโพรโทคอลอีก 2 โพรโทคอลคือ ไอซีเอ็มพี (ICMP: Internet Control Message Protocol) และไอจีเอ็มพี (IGMP : Internet Group Management Protocol)

2.2.4 ชั้นเน็ตเวิร์กอินเทอร์เฟซ (Network Interface Layer)

ชั้นเน็ตเวิร์กอินเทอร์เฟซมีชื่อเรียกอีกอย่างหนึ่งว่า ชั้นดาต้าลิงก์ (Data link layer) หรือชั้นลิงก์ (link layer) ทำหน้าที่เป็นดีไวซ์ไดรเวอร์ (Device driver) ของระบบปฏิบัติการ เพื่อให้สามารถรับส่งข้อมูลกับฮาร์ดแวร์ สำหรับการสื่อสารข้อมูลในเครือข่าย

2.3 การอ้างอิงที่อยู่ของไอพีและพอร์ต (IP Addressing and Port)

ในระบบโพรโทคอลสำหรับเครือข่ายต่างๆนั้น จำเป็นจะต้องมีวิธีการในการอ้างอิงที่อยู่ของจุด (node) ต่างๆ ซึ่งโพรโทคอลทีซีพี/ไอพีจะใช้ที่อยู่ของไอพี (IP address) ในการกำหนดที่อยู่ของจุดต่างๆในระบบเครือข่าย ที่อยู่ของไอพีจะเป็นตัวเลขขนาด 32 บิต โดยแสดงในรูปเลขฐาน 10 จำนวน 4 ตัว คั่นกลางด้วยจุด ตัวอย่างเช่น 192.1.1.1 , 148.1.2.35 เป็นต้น

ที่อยู่ของไอพี แบ่งเป็น 2 ส่วน ได้แก่

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

ที่อยู่ของเครือข่ายจะเป็นการบ่งบอกที่อยู่ของจุดนั้นๆ ว่าอยู่ที่เครือข่ายใด และที่อยู่ของเครื่องเป็นอยู่ที่บอกว่าในเครือข่ายนั้นเป็นเครื่องใด

ในโพรโทคอลทีซีพี/ไอพีแบ่งระดับ (Class) ของที่อยู่ไอพี เป็น 5 ระดับ ดังนี้

ระดับ	จำนวนหลักของที่อยู่ ของเครือข่าย	ช่วงของที่อยู่ของเครือข่าย	จำนวนหลักที่อยู่ของ ที่อยู่ของเครื่อง	ช่วงของที่อยู่ ของเครื่อง
A	1	1-127	3	0.0.0-255.255.255
B	2	128.0-191.255	2	0.0-255.255
C	3	192.0.0-222.255.255	1	0.255
D	4	224.0.0.0-239.255.255.255	-	-
E	5	240.0.0.0-255.255.255.255	-	-

ตารางที่ 2-1 แสดงการแบ่งคลาส (Class) ของที่อยู่ไอพี

ที่อยู่ของไอพีที่ถูกสำรองไว้ โดยมีความหมายอย่างอื่น ได้แก่

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

ในการทำงานของโปรแกรมประยุกต์ของโทร โคคอลที่ซีพี/ไอพี ทั้งที่ใช้โทร โคคอลที่ซีพีและยูดีพี ต้องมีกระบวนการในการระบุถึงกระบวนการที่ทำการติดต่อด้วย ซึ่งทั้งที่ซีพีและยูดีพีใช้เลข 16 บิต เรียกว่า พอร์ต(port) กระบวนการให้บริการจะต้องมีพอร์ตที่เป็นที่รู้จักกันทั่วไป (wellknown port) เพื่อให้ผู้ขอรับบริการสามารถที่จะติดต่อมาได้ เช่น กระบวนการให้บริการเทลเน็ต (Telnet server) จะใช้พอร์ต 23 หรือกระบวนการให้บริการเอฟทีพี (FTP server) จะใช้พอร์ต 21

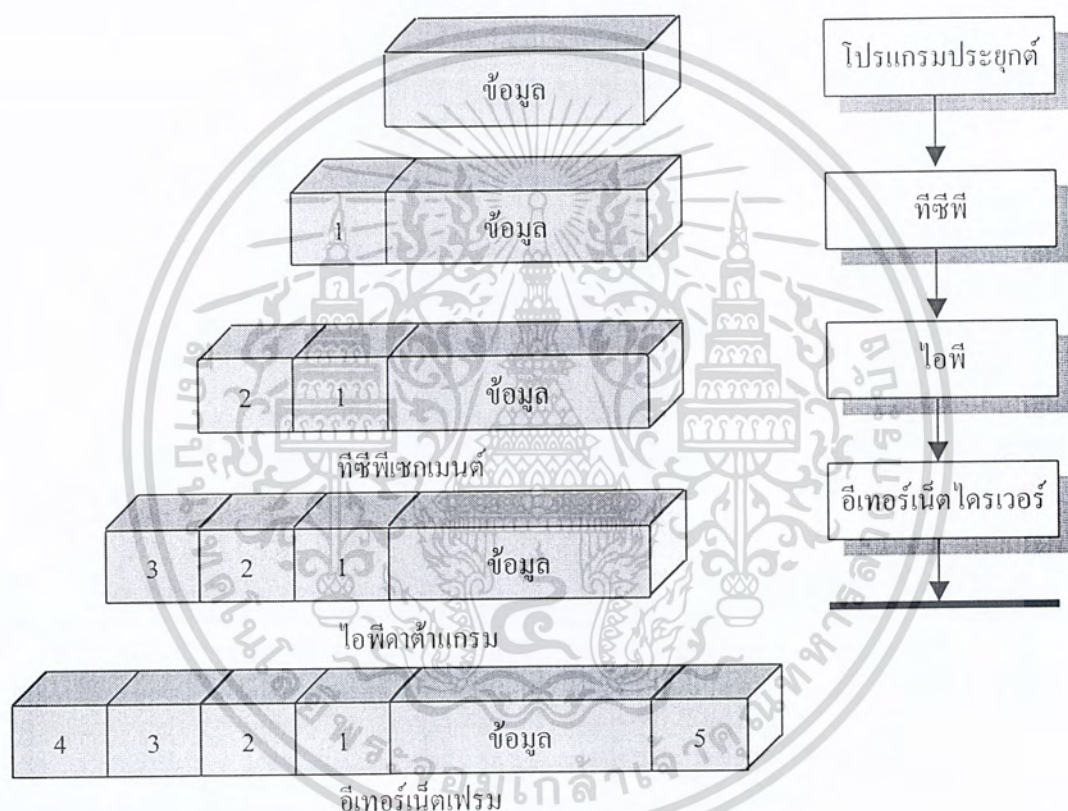
สำหรับกระบวนการขอรับบริการ จะกำหนดพอร์ตขึ้นมาพอร์ตหนึ่งเพื่อใช้ติดต่อกับพอร์ตของกระบวนการให้บริการ ดังนั้นในแต่ละการติดต่อระหว่างกระบวนการขอรับบริการและกระบวนการให้บริการจะใช้ค่าที่อ้างอิง 5 ค่า ดังต่อไปนี้

- โพรโตคอลที่ซีพีหรือยูดีพี
- ที่อยู่ไอพีของตนเอง
- พอร์ตของตนเอง
- ที่อยู่ของไอพีอีกด้านหนึ่ง
- พอร์ตของอีกด้านหนึ่ง

2.4 การห่อหุ้มแพ็กเก็ต (Packet Encapsulation)

ในการสื่อสารข้อมูลตามโพรโทคอลทีซีพี/ไอพีระหว่างการผ่านชั้นต่างๆ จากบนลงล่าง ข้อมูลจะถูกส่งเป็นรูปของแพ็กเก็ต (Packet) โดยในแต่ละระดับชั้นจะมีการเพิ่มส่วนหัว (header) ของแพ็กเก็ตเพื่อใช้สื่อสารกับโพรโทคอลระดับเดียวกันของอีกฝ่ายหนึ่ง

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



- 1) เฮดเดอร์ของโปรแกรมประยุกต์ (application header)
- 2) เฮดเดอร์ของทีซีพี (TCP header)
- 3) เฮดเดอร์ของไอพี (IP header)
- 4) เฮดเดอร์ของอีเทอร์เน็ต (Ethernet header)
- 5) เทรลเลอร์ของอีเทอร์เน็ต

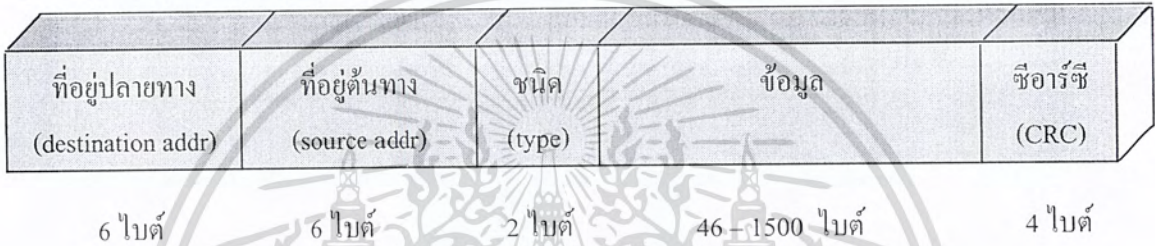
รูปที่ 2-2 การห่อหุ้มแพ็กเก็ต

2.5 อีเทอร์เน็ตเฟรม (Ethernet Frame)

โพรโทคอลทีซีพี/ไอพี เป็นโพรโทคอลที่สามารถใช้งานได้กับชนิดของการรับส่งข้อมูลหลายรูปแบบเช่น อีเทอร์เน็ต (Ethernet) โทกเกนริง (Token ring) หรือเฟดดีไอ (FDDI) เป็นต้น

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

ลักษณะของเฟรมของอีเทอร์เน็ตเป็นดังนี้



รูปที่ 2-3 เฟรมอีเทอร์เน็ต

บทที่ 3

เอฟทีพี

3.1 ลักษณะทั่วไป

เอฟทีพีเป็นโพรโทคอลสำหรับถ่ายโอนแฟ้มข้อมูลระหว่างคอมพิวเตอร์สองระบบ เอฟทีพีนั้นอาศัยโพรโทคอลที่ซีพีในการลำเลียงข้อมูล คุณสมบัติของเอฟทีพีคือช่วยให้การถ่ายโอนแฟ้มข้อมูลข้ามเครือข่ายได้โดยง่ายถึงแม้ว่าระบบแฟ้มของคอมพิวเตอร์ทั้งสองด้านนั้นจะแตกต่างกัน

การถ่ายโอนแฟ้มข้อมูลข้ามระหว่างโฮสต์เป็นความต้องการหลักอย่างหนึ่ง ในการใช้งานระบบเครือข่ายทั่วไป การสำเนาแฟ้มขึ้นใหม่หรือการสำเนาแฟ้มข้ามไดเรกทอรีเป็นงานปกติที่ทำได้โดยตรงและเรียบง่าย แต่การสำเนาแฟ้มข้ามเครื่องผ่านระบบเครือข่ายมีข้อระมัดระวังและต้องคำนึงหลายประการ ทั้งนี้เพราะโฮสต์ที่ทำงานภายใต้ระบบปฏิบัติการที่ต่างกันมักมีระบบจัดเก็บแฟ้มที่แตกต่างกัน นับตั้งแต่วิธีตั้งชื่อแฟ้มและไดเรกทอรี วิธีเข้าสู่ไดเรกทอรีย่อย หรือข้อกำหนดเพื่อป้องกันการใช้แฟ้มโดยไม่ได้รับอนุญาต เป็นต้น

ภายใต้สภาวะที่ปัจจุบันมีระบบปฏิบัติการที่แตกต่างกันเป็นจำนวนมาก การสร้างระบบปฏิบัติการหนึ่งๆ ให้รู้จักกับวิธีการจัดเก็บแฟ้มข้อมูลของระบบปฏิบัติการอื่นทั้งหมดเป็นสิ่งที่ทำได้ยาก ผู้ออกแบบที่ซีพี/ไอพีแก้ปัญหานี้โดยสร้างโพรโทคอลประยุกต์เอฟทีพี (FTP : File Transfer Protocol) เพื่อทำหน้าที่เป็นตัวกลางแลกเปลี่ยนคำสั่ง เอฟทีพีแต่ละด้านจะเชื่อมโยงเข้ากับระบบแฟ้มข้อมูลของตนเองและทำงานตามชุดคำสั่งมาตรฐานทำให้โฮสต์หนึ่งไม่ต้องทราบอีกโฮสต์หนึ่งนั้นมีการจัดเก็บข้อมูลอย่างไร



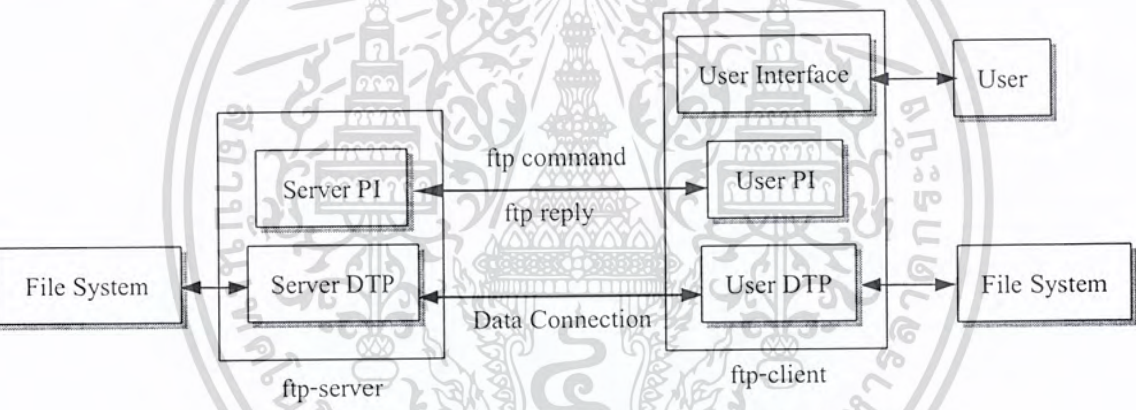
รูปที่ 3-1 การแปลงคำสั่งเอฟทีพี

จากรูปที่ 3-1 แสดงตัวอย่างวินโดวส์ไคลเอ็นต์ขอบริการถ่ายโอนด้วยเอฟทีพีจากยูนิกซ์เซิร์ฟเวอร์ เมื่อผู้ใช้ป้อนคำสั่ง dir คำสั่งจะถูกส่งไปยังเซิร์ฟเวอร์และแปลงไปสู่คำสั่ง ls การใช้เอฟทีพีจึงช่วยการจัดการไฟล์ข้ามเครือข่ายได้โดยง่าย และกำจัดความไม่เข้ากันในการจัดเก็บแฟ้มของระบบปฏิบัติการออกไป

3.2 แบบจำลองของเอฟทีพี

อาร์เอฟซี 959 นิยามแบบจำลองการทำงานของเอฟทีพีซึ่งทำงานตามแบบไคลเอ็นต์-เซิร์ฟเวอร์ เอฟทีพีเป็นโพรโทคอลที่ค่อนข้างซับซ้อน เนื่องจากการสถาปนาที่ซับซ้อนขึ้นสองส่วนคือ ส่วนรับส่งคำสั่ง และส่วนโอนย้ายข้อมูล

เอฟทีพีแต่ละด้านจะมีตัวแปลโพรโทคอล (Protocol Interpreter : PI) ต่อเชื่อมผ่านการเชื่อมต่อส่วนควบคุม (Control Connection : CC) ซึ่งเป็นช่องทางส่งคำสั่ง ตัวแปลโพรโทคอลมีหน้าที่แปลคำสั่งและดำเนินการตามคำสั่งนั้น ส่วนเชื่อมต่อที่สองคือ การเชื่อมต่อส่วนข้อมูล (Data Connection :DC) เป็นช่องทางส่งข้อมูลซึ่งอยู่ภายใต้การดูแลของโมดูลถ่ายโอนข้อมูล (Data Transfer Process : DTP) แบบจำลองของเอฟทีพีนั้นแสดงได้ดังรูปที่ 3-2



รูปที่ 3-2 แบบจำลองการทำงานของเอฟทีพี

ข้อสังเกต

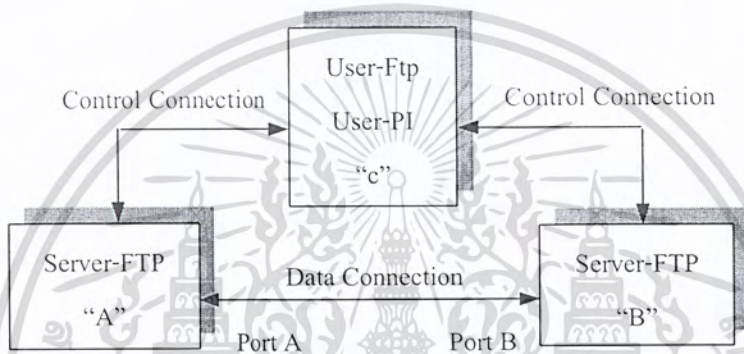
- 1. การเชื่อมต่อส่วนข้อมูล (data connection) นั้นอาจจะไม่ถูกใช้แบบในรูปที่แสดงนี้ได้
- 2. การเชื่อมต่อส่วนข้อมูล (data connection) ไม่จำเป็นจะต้องมีอยู่ตลอดเวลา

จากแบบจำลองที่แสดงดังรูปที่ 3-2 ตัวแปลโพรโทคอลจะเป็นส่วนที่เริ่มสร้างการเชื่อมต่อส่วนควบคุม (control connection) ซึ่งในส่วนนี้จะมีการทำงานเหมือนกับโพรโทคอลเทลเน็ต ตอนเริ่มต้นนั้น คำสั่งเอฟทีพีมาตรฐานจะถูกสร้างโดยยูสเซอร์-พีไอ และส่งไปยังเซิร์ฟเวอร์ผ่านการเชื่อมต่อส่วนควบคุม การตอบกลับมาตรฐานนั้นถูกส่งมาจากเซิร์ฟเวอร์-พีโอมาหา ยูสเซอร์-พีไอ เป็นการตอบรับของคำสั่งที่ส่งไป

คำสั่งของเอฟทีพีจะระบุพารามิเตอร์สำหรับการเชื่อมต่อส่วนข้อมูลไปด้วย (พอร์ตข้อมูล, โหมดของการถ่ายโอนข้อมูล, ประเภทของข้อมูล, โครงสร้าง) และลักษณะของการจัดการระบบไฟล์ ยูสเซอร์-ดีทีพี ควรจะรอรับข้อมูลบนพอร์ตข้อมูลที่ได้ระบุมา และเซิร์ฟเวอร์ก็จะเริ่มสร้างการเชื่อมต่อส่วนข้อมูล

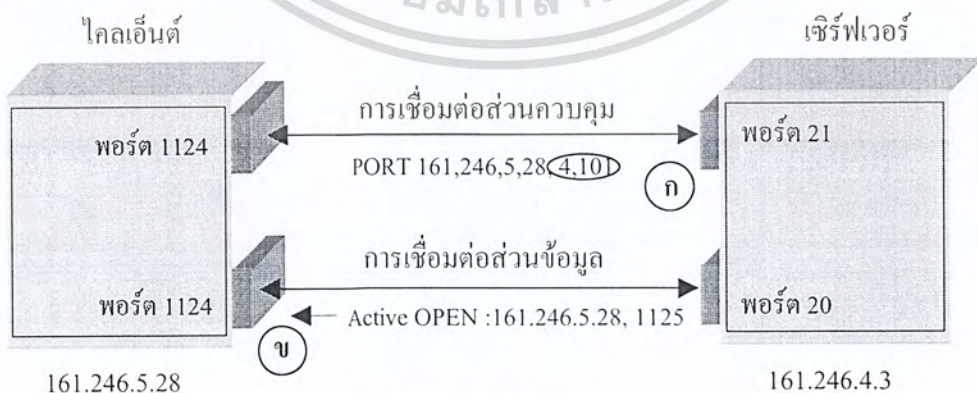
และการโอนย้ายข้อมูลด้วยพารามิเตอร์ที่ตรงกันกับฝั่งผู้ใช้ เป็นข้อสังเกตว่าพอร์ตข้อมูลนั้นไม่จำเป็นต้องอยู่ในโฮสต์เดียวกับที่เริ่มสร้างคำสั่งเอฟทีพี แต่เพียงผู้ใช้หรือยูสเซอร์-เอฟทีพีที่ต้องแน่ใจว่าได้ ‘รอรับข้อมูล’ บนพอร์ตข้อมูลที่ได้ระบุไว้ และต้องเป็นที่สังเกตด้วยเหมือนกันว่าการเชื่อมต่อส่วนข้อมูลนั้นจะถูกใช้สำหรับการส่งและการรับข้อมูล

ในอีกสถานการณ์หนึ่งผู้ใช้นั้นต้องการที่จะส่งไฟล์ระหว่างโฮสต์ 2 แห่ง ซึ่งทั้งสองนั้นไม่ใช่โพลีโฮสต์ ผู้ใช้นั้นต้องทำการสร้างการเชื่อมต่อส่วนควบคุม (Control Connection) ไปยังเซิร์ฟเวอร์ทั้งสอง ต่อจากนั้นก็จัดการส่วนการเชื่อมต่อข้อมูลให้กับโฮสต์ทั้ง 2 ในการจัดการส่วนนี้ ข้อมูลในส่วนการเชื่อมต่อส่วนควบคุมจะถูกส่งไปยังยูสเซอร์-เอฟทีพี แต่ข้อมูลจะถูกส่งระหว่างกระบวนการโอนย้ายข้อมูล จากโมเดลนี้จะเป็นการติดต่อกันระหว่างเซิร์ฟเวอร์



รูปที่ 3-3 แสดงการโอนย้ายข้อมูลระหว่างเซิร์ฟเวอร์

โพรโตคอลต้องการให้ การเชื่อมต่อส่วนควบคุมถูกเปิดอยู่ในระหว่างการส่งข้อมูลกำลังดำเนินอยู่ มันเป็นหน้าที่ของผู้ใช้ที่จะต้องร้องขอเพื่อที่จะปิดการเชื่อมต่อในส่วนนี้เมื่อเสร็จสิ้นการให้บริการของโปรแกรมเอฟทีพี ในขณะที่เซิร์ฟเวอร์ที่กำลังทำงานอยู่นั้น เซิร์ฟเวอร์ควรจะยกเลิกการส่งข้อมูลถ้าการเชื่อมต่อถูกปิด โดยไม่ต้องมีคำสั่งใดๆ



รูปที่ 3-4 การสร้างส่วนเชื่อมต่อข้อมูล

จากรูปแสดงการสร้างส่วนเชื่อมต่อข้อมูลระหว่างไคลเอนต์และเซิร์ฟเวอร์ ในที่นี้สมมติให้พอร์ตประจำการเชื่อมต่อส่วนควบคุมของไคลเอนต์คือ 1124 และเตรียมพอร์ต 1125 รอไว้สำหรับการเชื่อมต่อส่วนข้อมูล

ไคลเอนต์จะขอเปิดการเชื่อมต่อส่วนข้อมูลตามตำแหน่ง ก. โดยส่งรหัสคำสั่ง PORT ตามด้วยอาร์กิวเมนต์หกตัวแยกด้วยจุลภาคคือโอพีแอคเดรส (161,246,5,28) และหมายเลขพอร์ต 4 , 101 ซึ่งแสดงถึงพอร์ต 1125 (เลขพอร์ตเป็นรหัส 16 บิต สองชุดติดกัน ดังนั้นตัวเลข 4 , 101 คือ $4 \times 256 + 101 = 1125$)

ต่อจากนั้นเซิร์ฟเวอร์จะสร้างการเชื่อมต่อจากพอร์ต 20 ไปยังไคลเอนต์ที่พอร์ต 1125 ตามตำแหน่ง ข.

3.3 การแทนข้อมูล (Data Representation)

เอฟทีพีมีรูปแบบการแทนข้อมูลเพื่อกำหนดวิธีการโอนย้ายและการจัดเก็บข้อมูล 4 ประเภทคือ ชนิดแฟ้มข้อมูล , พอร์เมตควบคุม, โครงสร้างข้อมูล , วิธีการส่งข้อมูล

3.3.1 ชนิดข้อมูล (Data Type)

คุณสมบัติที่สำคัญของเอฟทีพี คือความสามารถในการแปลงประเภทของข้อมูลให้ถูกต้องตามความเหมาะสม มีประโยชน์กรณีที่เครื่องเซิร์ฟเวอร์และเครื่องไคลเอนต์เป็นคอมพิวเตอร์ที่ต่างระบบกัน ซึ่งเอฟทีพีก็จะแปลงข้อมูลให้ถูกต้องตามความเหมาะสมได้โดยจะมีคำสั่ง TYPE เพื่อกำหนดประเภทของข้อมูลซึ่งประกอบด้วย

- เอ็นวีที-แอสกี (NVT-ASCII (Network Virtual Terminal ASCII)) เป็นข้อมูลทั่ว ๆ ไปที่ใช้รหัสแอสกี
- เอ็บซีดีค (EBCDIC) เป็นข้อมูลที่ใช้ในเครื่องคอมพิวเตอร์เมนเฟรมของไอบีเอ็ม
- อิมจ (IMAGE) เป็นข้อมูลที่เป็นไบนารี อาจเป็นโปรแกรมหรือรูปภาพโดยมีข้อมูล 8 บิต
- แบบเฉพาะโฮสต์ (LOCAL) เป็นข้อมูลที่เป็นไบนารีเช่นเดียวกับอิมจ แต่จำนวนบิตต่อหนึ่งไบต์อาจจะแตกต่างกันไปตามประเภทของเครื่องคอมพิวเตอร์

3.3.2 การควบคุมรูปแบบ (Format Control)

- ไม่มีฟอร์เมต (NonPrint) ไม่บรรจุรหัสควบคุม
- ฟอร์เมตเทลเน็ต (Telnet format control) มีรหัสควบคุมตามแบบเทลเน็ตสำหรับกำหนดการทำงานของเครื่องพิมพ์
- ฟอร์เมตฟอร์แทรน (Fortran carriage control) แบบภาษาฟอร์แทรน

3.3.3 โครงสร้างข้อมูล (Data Structure)

เอฟทีพีสามารถกำหนดโครงสร้างของไฟล์ได้หลายแบบ เพื่อรองรับการใช้งานร่วมกับเครื่องคอมพิวเตอร์ชนิดที่ต่างกัน โดยแยกเป็น 3 ประเภทได้ดังนี้

- แฟ้ม (file) ไม่มีโครงสร้างพิเศษใด ข้อมูลถือว่าเป็นไบนารีเรียงต่อกัน
- เรคอร์ด (record) ข้อมูลจัดแบ่งออกเป็นเรคอร์ด
- หน้า (page) ข้อมูลมีหมายเลขหน้ากำกับหน้า และสามารถถ่ายโอนโดยไม่ต้องเรียงลำดับหน้า

3.3.4 วิธีการรับส่ง (Transmission Mode)

- สายข้อมูล (stream) เป็นวิธีที่จะรับส่งข้อมูลเรียงลำดับไบนารีส่งต่อกันไปเรื่อยๆ ดังนั้นการรับส่งวิธีนี้จึงสามารถใช้ได้กับไฟล์ทุกประเภท ส่วนไฟล์ที่มีโครงสร้างต้องมีรหัสตัวอักษรพิเศษที่กำหนดการสิ้นสุดเรคอร์ด (End of Record หรือ EOR) และจบไฟล์ (End of File หรือ EOF) ด้วย ซึ่งการรับส่งข้อมูลจะสิ้นสุดโดยการตรวจสอบจากค่าของ EOF นี้
- บล็อก (block) เป็นโหมดการรับส่งข้อมูลที่เป็นบล็อก ในแต่ละบล็อกจะมีส่วนหัวที่ระบุขนาดและรายละเอียดต่างๆ ของบล็อก เช่น สถานะแสดงบล็อกสุดท้ายในไฟล์หรือในเรคอร์ด ขนาดบิตที่ใช้ตรวจสอบว่าข้อมูลที่รับส่งในบล็อกนั้นถูกต้องหรือไม่ รวมทั้งข้อมูลที่เรียกว่า Restart Marker ซึ่งเป็นข้อมูลที่ใช้อ้างอิงเพื่อโอนข้อมูลต่อจากข้อมูลของเดิมที่โอนไม่สำเร็จ
- บีบอัด (compress) เป็นวิธีเพิ่มประสิทธิภาพการรับส่งข้อมูล โดยใช้เทคนิคการบีบอัดข้อมูลให้เล็กลงโดยมีวิธีการง่ายๆ คือ ข้อมูลซ้ำๆ ที่เรียงต่อกันจะถูกลดขนาดเหลือ 3 ไบนารี ซึ่ง 2 ไบนารีแรกจะบอกให้รู้ว่าข้อมูลชุดดังกล่าวมีค่าซ้ำจำนวนกี่ตัว

3.4 การฟื้นฟูจากความผิดพลาด (Error Recovery)

ในการรับส่งไฟล์แบบสายข้อมูล (Stream Mode) การแก้ไขกรณีข้อมูลสูญหายระหว่างการรับส่ง จะใช้ความสามารถของโปรโตคอลที่ซีที แต่ไม่มีการส่งข้อมูลซ้ำ แต่ถ้าหากเป็นแบบบล็อกโหมดหรือโหมดบีบอัด ข้อมูลสามารถส่งใหม่เฉพาะส่วนที่ตรวจพบข้อผิดพลาดได้

3.5 คำสั่งและการตอบกลับของเอฟทีพี

3.5.1 คำสั่งของเอฟทีพี (FTP Command)

คำสั่งของเอฟทีพีใช้รหัสแอสกีเอ็นวีทีเช่นเดียวกับเทลเน็ต ซึ่งจะปิดท้ายด้วย CR LF ชื่อคำสั่งในเอฟทีพีอยู่ในรูปรหัสอักขระตัวใหญ่ บางคำสั่งอาจมีอาร์กิวเมนต์เป็นค่าเพื่อเลือกประกอบอยู่ด้วย จำนวนคำสั่งในเอฟทีพีมีมากกว่า 30 คำสั่ง ซึ่งแบบออกเป็นประเภทได้ดังนี้

3.5.1.1 คำสั่งควบคุมการเข้าถึง (Access Control Command)

คำสั่งดังต่อไปนี้จะเป็นรายละเอียดของคำสั่งที่ใช้ในการควบคุมการเข้าถึง

- USER NAME (USER) ตามด้วยพารามิเตอร์ที่บ่งบอกถึงชื่อผู้ใช้งานในระบบเพื่อเป็นการตรวจสอบสิทธิ์ผู้ใช้งานว่ามีผู้ใช้งานในระบบหรือไม่
- PASSWORD (PASS) ตามด้วยพารามิเตอร์ที่บ่งบอกถึงรหัสลับของผู้ใช้งานในระบบ เป็นการพิสูจน์ตนของผู้ใช้งานในระบบ

- ACCOUNT (ACCT) เป็นวิธีการเข้าใช้งานอีกรูปแบบหนึ่งนอกเหนือจากการใช้คำสั่ง USER
- CHANGE WORKING DIRECTORY (CWD) เป็นคำสั่งที่ใช้ในการเปลี่ยนไดเรกทอรีปัจจุบัน
- CHANGE TO PARENT DIRECTORY (CDUP) เป็นคำสั่งที่ใช้เพื่อเข้าถึงไดเรกทอรีที่อยู่ก่อนไดเรกทอรีปัจจุบัน
- STRUCTURE MOUNT (SMNT) จะทำให้ผู้ใช้สามารถใช้ระบบอื่นที่ผูกติดอยู่ได้โดยไม่ต้องใส่ชื่อผู้ใช้กับรหัสผ่านอีก
- REINITIALIZE (REIN) เป็นการรีเซตค่าใหม่ทั้งหมดแต่ไม่หยุดการโอนย้ายข้อมูลของการเชื่อมต่อส่วนข้อมูล
- LOGOUT (QUIT) เป็นการยกเลิกการติดต่อ

3.5.1.2 คำสั่งพารามิเตอร์การโอนย้าย (Transfer Parameter Commands)

ค่าพารามิเตอร์ในการโอนย้ายข้อมูลทั้งหมดนั้นจะต้องมีค่าดีฟอลท์ และคำสั่งจะระบุค่าพารามิเตอร์การโอนย้ายข้อมูลก็ต่อเมื่อค่าพารามิเตอร์ดีฟอลท์มีการเปลี่ยนแปลง ค่าดีฟอลท์อาจจะจะเป็นค่าสุดท้ายที่มีการระบุไว้หรือไม่ได้มีการระบุไว้ ค่าดีฟอลท์มาตรฐานยังคงเป็นค่าเดิม

- DATA PORT (PORT) จะส่งพารามิเตอร์ (แอดเดรสกับพอร์ตที่เปิดรอการติดต่อ) ไปยังเซิร์ฟเวอร์
- PASSIVE (PASV) จะสั่งให้เซิร์ฟเวอร์เปลี่ยนไปอยู่ในโหมดพาสซีฟ
- REPRESENTATION TYPE (TYPE) ใช้สำหรับตั้งค่าการรับส่งเพิ่มข้อมูลมี 2 โหมด คือโหมดแอสกีกับโหมดไบนารี
- FILE STRUCTURE (STRU) เป็นการส่งไฟล์ตามลักษณะของไฟล์ที่จะส่งมีอยู่ 3 ชนิด คือ แฟ้ม, เรคอร์ด, เพจ โดยปกติจะถูกตั้งค่าเป็นแฟ้ม
- TRANSFER MODE (MODE) เป็นการส่งไฟล์ที่มีหลายรูปแบบในการส่ง มีรูปแบบในการส่งอยู่ 3 ชนิด คือ สายข้อมูล, บล็อก, บีบอัด โดยปกติจะถูกตั้งค่าเป็นสายข้อมูล

3.5.1.3 คำสั่งให้บริการของเอฟทีพี (FTP Service Commands)

คำสั่งที่ใช้บริการของเอฟทีพีที่จะอธิบายการโอนย้ายไฟล์ หรือการร้องขอจากผู้ใช้งาน ส่วนใหญ่แล้วการระบุค่าอาร์กิวเมนต์ของคำสั่งจะเป็นชื่อพาธ

- RETRIEVE (RETR) ใช้เมื่อต้องการไฟล์ที่อยู่บนเซิร์ฟเวอร์
- STORE (STOR) ใช้เมื่อต้องการเก็บไฟล์ไว้ในบนเซิร์ฟเวอร์
- STORE UNIQUE (STOU) เป็นคำสั่งใช้เหมือนกัน STOR แต่จะสร้างเก็บไว้ในไดเรกทอรีที่มีชื่อเดียวกับไฟล์นั้น
- APPEND (with create) (APPE) ใช้เมื่อต้องการส่งไฟล์ไปต่อกับไฟล์ที่มีอยู่แล้ว
- ALLOCATE (ALLO) เป็นคำสั่งไว้สำหรับจองพื้นที่บนเซิร์ฟเวอร์ เพื่อเก็บไฟล์
- RESTART (REST) คำสั่งนี้จะส่งค่าตำแหน่งของไฟล์ที่ต้องการจะดาวน์โหลดมา

- RENAME FROM (RNFR) จะเปลี่ยนชื่อไฟล์โดยระบุชื่อไฟล์ที่จะเปลี่ยน โดยจะทำหลังคำสั่ง RNT0
- RENAME TO (RNT0) จะเปลี่ยนเป็นชื่อไฟล์ใหม่ โดยจะทำก่อนคำสั่ง RNFR
- ABORT (ABOR) เป็นคำสั่งยกเลิกการติดต่อ
- DELETE (DELE) เป็นคำสั่งลบไฟล์ที่อยู่บนเซิร์ฟเวอร์
- REMOVE DIRECTORY (RMD) เป็นคำสั่งลบไดเรกทอรีที่อยู่บน เซิร์ฟเวอร์
- MAKE DIRECTORY (MKD) เป็นคำสั่งสร้างไดเรกทอรีบนเซิร์ฟเวอร์
- PRINT WORKING DIRECTORY (PWD) แสดงไดเรกทอรีปัจจุบันบน เซิร์ฟเวอร์
- LIST (LIST) แสดงรายชื่อไฟล์ที่อยู่บนเซิร์ฟเวอร์ในไดเรกทอรีปัจจุบัน
- NAME LIST (NLST) แสดงรายชื่อไฟล์แบบมีรายละเอียดที่อยู่บนเซิร์ฟเวอร์ ในไดเรกทอรีปัจจุบัน
- SITE PARAMETERS (SITE) เป็นการส่งพารามิเตอร์ที่เป็นคำสั่งที่สร้างขึ้นเฉพาะ
- SYSTEM (SYST) เป็นคำสั่งที่จะแสดงชนิดของระบบปฏิบัติการที่เป็นเซิร์ฟเวอร์เอฟทีพี
- STATUS (STAT) จะแสดงสถานะทั้งหมดที่เกี่ยวข้องกับการรับส่งแฟ้มข้อมูล
- HELP (HELP) แสดงวิธีการใช้ของคำสั่งที่ต้องการจะใช้
- NOOP (NOOP) ไม่มีผลอะไรกับเซิร์ฟเวอร์แต่ส่งไปเพื่อไม่ให้เซิร์ฟเวอร์ตัดการติดต่อ

3.5.2 การตอบกลับของเอฟทีพี (FTP Replies)

3.5.2.1 รหัสการตอบกลับ

การตอบกลับของเอฟทีพีประกอบด้วยรหัสตัวเลขแอสกี 3 หลัก พร้อมด้วยคำอธิบายซึ่งอาจมีหรือไม่ก็ได้ ซอฟต์แวร์ด้านไคลเอนต์เพียงแต่นำรหัสตัวเลขไปตรวจสอบการทำงาน ส่วนข้อความนั้นมีเพื่อให้ผู้ใช้เข้าใจถึงผลการทำงานด้วย แต่ละหลักของตัวเลข 3 หลัก จะมีความหมายเฉพาะตัว

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

- 1 การร้องขอการกระทำกำลังเกิดขึ้น (initiated)
- 2 ตอบสนองการร้องขอเสร็จเรียบร้อยแล้ว เซิร์ฟเวอร์พร้อมที่จะรับการร้องขอครั้งใหม่
- 3 ได้รับคำสั่งแล้ว แต่ในการร้องขอการปฏิบัติการต้องการข้อมูลมากกว่านี้
- 4 ไม่ยอมรับคำสั่ง การร้องขอการปฏิบัติล้มเหลว แต่เงื่อนไขที่ล้มเหลวสามารถที่จะร้องขอได้อีกครั้ง
- 5 ไม่ยอมรับคำสั่ง การร้องขอการปฏิบัติล้มเหลว แต่เงื่อนไขที่ล้มเหลวมีท่าทางว่าจะกลับมาเกิดอีกครั้ง

ตัวเลขในหลักที่สองแสดงข้อมูลในส่วนการทำงาน ได้แก่

- 0 Syntax แสดงถึง ไวยากรณ์ผิดพลาด (syntax error)
- 1 Information แสดงถึงการร้องตอบการร้องขอข้อมูล
- 2 Connection หมายความว่าข้อความนั้นได้มีการตอบกลับสำหรับการร้องขอในการควบคุมและการรับส่งข้อมูล
- 3 Authentication and accounting แสดงถึงการล็อกอินหรือมีการปฏิบัติการ
- 4 ยังไม่ระบุการใช้งาน
- 5 File System แสดงข้อมูลเกี่ยวกับสถานะของระบบไฟล์ของเซิร์ฟเวอร์

เลขหลักที่สาม เป็นตัวกำหนดความหมายเพิ่มเติมโดยเฉพาะกับรหัสการตอบกลับนั้น ๆ

ตัวอย่างรหัสและการตอบกลับ

- 110 restart marker reply. โดย yyyy = mmmm เมื่อ yyy เป็น user process data stream marker และ mmmm เป็น ftpd equivalent marker
- 120 พร้อมทั้งจะให้บริการ ใน mnn นาที
- 200 คำสั่งถูกต้อง
- 211 สถานะของระบบ หรือ reply help ของระบบ
- 212 สถานะของไดเรกทอรี
- 230 ยูล์เซอรล็อกอิน
- 250 การร้องขอที่จะกระทำกับไฟล์ ถูกต้อง
- 331 ชื่อผู้ใช้ถูกต้อง ต้องการรหัสผ่าน
- 425 ไม่สามารถเปิดการเชื่อมต่อส่วนข้อมูล (Data Connection)
- 451 การร้องขอบริการถูกยกเลิก เกิดข้อผิดพลาดในการทำงาน
- 500 ไวยากรณ์ผิดพลาด (Syntax error) ไม่สามารถแปลความหมายได้หรือคำสั่งนั้นมีความยาวมากเกินไป
- 530 ไม่ได้ล็อกอิน
- 550 การร้องขอให้เกิดขึ้น, ไม่มีไฟล์นี้ปรากฏอยู่, ไม่พบ, ไม่สามารถเข้าถึงได้

สำหรับคำสั่งโดยทั่วไปแล้วเอฟทีพีจะตอบกลับเพียงบรรทัดเดียวโดยมีรหัสตอบกลับนำหน้า เช่น

- 226 การโอนย้ายสมบูรณ์

แต่ถ้ามีการตอบกลับหลายบรรทัด เอฟทีพีจะแสดงรหัสพร้อมทั้งกำกับรหัสบรรทัดแรกด้วยเครื่องหมาย “-” ตัวอย่างเช่นเมื่อใช้คำสั่ง HELP จะมีการตอบกลับหลายบรรทัด ดังนี้

220 diamond FTP server (Version 1.7.212.3 Wed Jul 14 10:24:05 GMT 1999) ready.

HELP

214-The following commands are recognized (* =>'s unimplemented).

USER	PORT	STOR	MSAM*	RNTO	NLST	MKD	CDUP	PBSZ*
PASS	PASV	APPE	MRSQ*	ABOR	SITE	XMKD	XCUP	PROT*
ACCT*	TYPE	MLFL*	MRCP*	DELE	SYST	RMD	STOU	SIZE
SMNT*	STRU	MAIL*	ALLO	CWD	STAT	XRMD	AUTH*	MDTM
REIN*	MODE	MSND*	REST	XCWD	HELP	PWD	ADAT*	
QUIT	RETR	MSOM*	RNFR	LIST	NOOP	XPWD	CCC*	

214 Direct comments to ftp-bugs@diamond.

รูปที่ 3-5 แสดงการตอบกลับแบบหลายบรรทัดของคำสั่ง HELP

3.5.3 ตัวอย่างแสดงรหัสคำสั่งเอฟทีพี

เพื่อแสดงการส่งคำสั่งและการตอบกลับของเอฟทีพี จะขอยกตัวอย่างการใช้เอฟทีพีที่เซิร์ฟเวอร์ของภาควิชาวิศวกรรมคอมพิวเตอร์

3.5.3.1 การเชื่อมต่อ

ตัวอย่างแรกเป็นการเชื่อมต่อไปยังเอฟทีพีเซิร์ฟเวอร์โดยใช้ชื่อพจน -d (debug) เพื่อให้ไคลเอ็นต์แสดงคำสั่งที่ใช้ติดต่อกับเซิร์ฟเวอร์ บรรทัดที่ขึ้นต้นด้วยสัญลักษณ์ “--->” หมายถึงไคลเอ็นต์ส่งคำสั่งไปเซิร์ฟเวอร์ และบรรทัดที่ขึ้นต้นด้วยตัวเลขถือรหัสที่เซิร์ฟเวอร์ตอบกลับ

จากตัวอย่างนี้ไคลเอ็นต์จะส่งคำสั่ง USER และ PASS สำหรับการล็อกอิน ไคลเอ็นต์บางตัวเช่น

```
root@cherry:/home/kit# ftp -d 161.246.4.3
Connected to 161.246.4.3.
220 diamond FTP server (Version 1.7.212.3 Wed Jul 14 10:24:05 GMT 1999) ready.
Name (161.246.4.3:kit): s1014031
---> USER s1014031
331 Password required for s1014031.
Password:
---> PASS XXXX
230 User s1014031 logged in.
---> SYST
215 UNIX Type: L8
```

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> dir .ssh
---> PORT 161,246,5,215,7,17
200 PORT command successful.
---> LIST .ssh
150 Opening ASCII mode data connection for /usr/bin/ls.
total 12
drwxr-xr-x      2 s1014031  std4d      1024 Jan 30 01:12 .
drwxrwxrwx     10 s1014031  std4d      1024 Mar 14 05:10 ..
-rw-r-----    1 s1014031  std4d        17 Jan 30 01:12 authorized_keys
-rw-r--r--     1 s1014031  std4d       665 Mar  5 01:27 known_hosts
-rw-----     1 s1014031  std4d      1024 Mar 15 16:32 prng_seed
-rw-----     1 s1014031  std4d      727 Jan 30 01:12 s1014031.pub
226 Transfer complete.
ftp> quit
---> QUIT
221 Goodbye.
```

รูปที่ 3-6 ตัวอย่างการทำงานของเอฟทีพี

ในตัวอย่างนี้จะส่งคำสั่ง SYST เพื่อสอบถามระบบของเซิร์ฟเวอร์

คำสั่ง dir เป็นคำสั่งขอรายละเอียดของไฟล์หรือไดเรกทอรี เนื่องจากการส่งรายชื่อไฟล์จะส่งทางการเชื่อมต่อส่วนข้อมูล โคลเอ็นต์จึงขอเปิดการเชื่อมต่อโดยคำสั่ง PORT หลังจากนั้นจึงส่งคำสั่ง LIST เพื่อขอรายชื่อไฟล์ เมื่อต้องการยกเลิกการเชื่อมต่อโดยพิมพ์คำสั่ง quit โคลเอ็นต์จะส่งคำสั่ง QUIT และเซิร์ฟเวอร์ตอบกลับด้วยรหัส 221 จากนั้นจึงปิดการเชื่อมต่อ

ในระหว่างที่มีการเชื่อมต่อนี้หากใช้คำสั่ง netstat ตรวจสอบสถานะของทีซีพีจะพบสถานะดังนี้

คือ

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

รูปที่ 3-7 แสดงการเชื่อมต่อระหว่างโคลเอ็นต์และเซิร์ฟเวอร์

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

3.5.3.2 การโอนย้าย

ตัวอย่างต่อไปนี้แสดงการโอนย้ายไฟล์ข้อมูลด้วยคำสั่ง get จะพบว่าไคลเอ็นต์เริ่มติดต่อการกำหนดแบบการถ่ายโอนแบบไบนารี (I) และสร้างการเชื่อมโยงข้อมูลและส่งคำสั่ง RETR เพื่อขอเพิ่มข้อมูล Search.class

```
ftp> cd /home/std4d/s1014031/www/spider
---> CWD /home/std4d/s1014031/www/spider
250 CWD command successful.
ftp> dir
---> PORT 161,246,5,215,7,246
200 PORT command successful.
---> LIST
150 Opening ASCII mode data connection for /usr/bin/l.
total 60
-rwxr-xr-x 1 s1014031 std4d 2090 Feb 20 00:23 MakeIndex.class
-rwxr-xr-x 1 s1014031 std4d 2692 Feb 20 00:23 MakeIndex.java
-rwxr-xr-x 1 s1014031 std4d 2929 Feb 20 00:23 Search.class
-rwxr-xr-x 1 s1014031 std4d 600 Feb 20 00:23 Search.html
-rwxr-xr-x 1 s1014031 std4d 4443 Feb 20 00:23 Search.java
-rwxr-xr-x 1 s1014031 std4d 2160 Feb 20 00:23 Spider.class
-rwxr-xr-x 1 s1014031 std4d 5334 Feb 20 00:23 Spider.java
-rwxr-xr-x 1 s1014031 std4d 1356 Feb 20 00:23 WebSpider.class
-rwxr-xr-x 1 s1014031 std4d 3867 Feb 20 00:23 WebSpider.java
226 Transfer complete.
ftp> get Search.class
local: Search.class remote: Search.class
---> TYPE I
200 Type set to I.
---> PORT 161,246,5,215,7,247
200 PORT command successful.
```

```

---> RETR Search.class
150 Opening BINARY mode data connection for Search.class (2929 bytes).
226 Transfer complete.
2929 bytes received in 0.142 secs (20 Kbytes/sec)
ftp>

```

รูปที่ 3-8 แสดงการโอนย้ายข้อมูลในเอฟทีพี

3.5.3.3 การขัดจังหวะ

ระหว่างการโอนย้ายหากผู้ใช้ขัดจังหวะด้วยการกด Ctrl-C การยกเลิกจะเกิดขึ้น อย่างไรก็ตามนั้นขึ้นอยู่กับทิศทางการถ่ายโอน หากไคลเอ็นต์ถ่ายโอนข้อมูลไปยังเซิร์ฟเวอร์การขัดจังหวะย่อมทำได้ง่ายเนื่องจากไคลเอ็นต์เพียงแต่หยุดส่งข้อมูลและส่งคำสั่ง ABOR ไปยังเซิร์ฟเวอร์ แต่ในกรณีการขัดจังหวะเมื่อโอนย้ายข้อมูลจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์จะมีขั้นตอนเพิ่มขึ้น เนื่องจากไคลเอ็นต์ต้องส่งคำสั่งไปบอกให้เซิร์ฟเวอร์หยุดการโอนย้าย

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

```

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

```

รูป 3-9 แสดงการขัดจังหวะการโอนย้ายจากเซิร์ฟเวอร์ไปยังไคลเอ็นต์

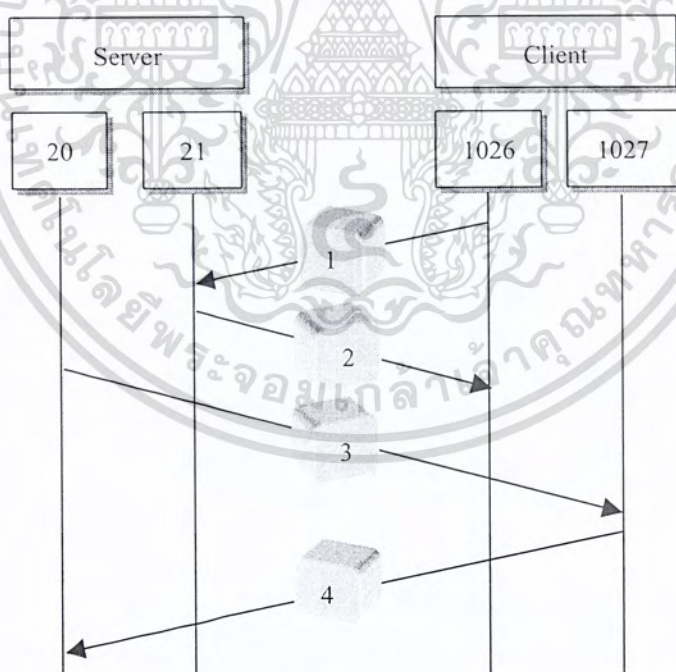
3.6 ลักษณะการทำงานของโปรแกรมเอฟทีพี

3.6.1 การทำงานแบบแอ็กทีฟ

ในการทำงานของเอฟทีพีแบบแอ็กทีฟ ฟังโคลเอ็นต์จะเป็นฝ่ายที่ไปสร้างการเชื่อมต่อจากค่าพอร์ตที่สุ่มขึ้นมา ($N > 1024$) ไปยังพอร์ต 21 ของเซิร์ฟเวอร์ ค่อยจากนั้นจะเริ่มรอรับข้อมูลที่พอร์ต $N+1$ และส่งคำสั่งเอฟทีพี PORT $N+1$ ไปยังเอฟทีพีเซิร์ฟเวอร์ เครื่องเซิร์ฟเวอร์จะติดต่อเพื่อทำการเชื่อมต่อกลับไปยังพอร์ตของโคลเอ็นต์ที่ได้รับนามกับพอร์ต 20 ของเซิร์ฟเวอร์

ไฟล์วอลล์ที่ฟังของเซิร์ฟเวอร์ที่จะสนับสนุนการทำงานแบบแอ็กทีฟจะต้องมีการเปิดช่องจากการติดต่อสื่อสารดังนี้

- การติดต่อมายังพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (การเริ่มการใช้งานของเครื่องฟังโคลเอ็นต์)
- การติดต่อจากพอร์ต 21 ของเอฟทีพีเซิร์ฟเวอร์ไปยังพอร์ต $N > 1024$ (การติดต่อกลับไปยังพอร์ตในส่วนควบคุมของโคลเอ็นต์)
- การติดต่อจากพอร์ต 20 ของเอฟทีพีเซิร์ฟเวอร์ไปยังพอร์ต $N > 1024$ (การติดต่อไปยังพอร์ตในส่วนข้อมูลของโคลเอ็นต์เพื่อสร้างการเชื่อมต่อส่วนข้อมูล)
- การติดต่อมายังพอร์ต 20 ของเอฟทีพีเซิร์ฟเวอร์จากพอร์ต $N > 1024$ (การส่งสัญญาณ ACKs ไปยังพอร์ตข้อมูลของเซิร์ฟเวอร์)



รูปที่ 3-10 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบแอ็กทีฟ

จากรูปที่ 3-10 ในขั้นตอนที่ 1 เป็นการติดต่อกันจากพอร์ตควบคุมของโคลเอ็นต์กับพอร์ตส่วนควบคุมของเซิร์ฟเวอร์ และส่งคำสั่ง PORT 1027 แล้วเซิร์ฟเวอร์ก็ส่งสัญญาณ ACK ตอบกลับไปยังพอร์ต

ควบคุมของไคลเอนต์ ในขั้นที่ 2 ต่อมาในขั้นตอนที่ 3 เซิร์ฟเวอร์จะสร้างเชื่อมต่อจากพอร์ตข้อมูลของเซิร์ฟเวอร์ไปยังพอร์ตของไคลเอนต์ที่ได้รับรูไว้ในข้างต้น สุดท้ายไคลเอนต์ก็จะส่งสัญญาณ ACK กลับมายังเซิร์ฟเวอร์เพื่อยืนยันการเชื่อมต่อ

ปัญหาหลักของการทำงานแบบแอ็กทีฟ คือการที่โปรแกรมเอพีทีพีของฝั่งไคลเอนต์นั้นไม่จำเป็นต้องสร้างการเชื่อมต่อจริงๆ ไปยังพอร์ตข้อมูลของเซิร์ฟเวอร์ มันเพียงแค่ไปบอกเซิร์ฟเวอร์ให้ทราบว่า มีพอร์ตอะไรที่กำลังรอรับข้อมูลอยู่ และเซิร์ฟเวอร์ก็จะติดต่อกลับมายังพอร์ตที่ระบุไว้ของเครื่องไคลเอนต์เอง แต่ถ้าไพล์วอลล์อยู่ฝั่งของไคลเอนต์แล้ว จะมีการบล็อกการติดต่อจากภายนอกไปยังเครื่องไคลเอนต์ที่อยู่ภายใน

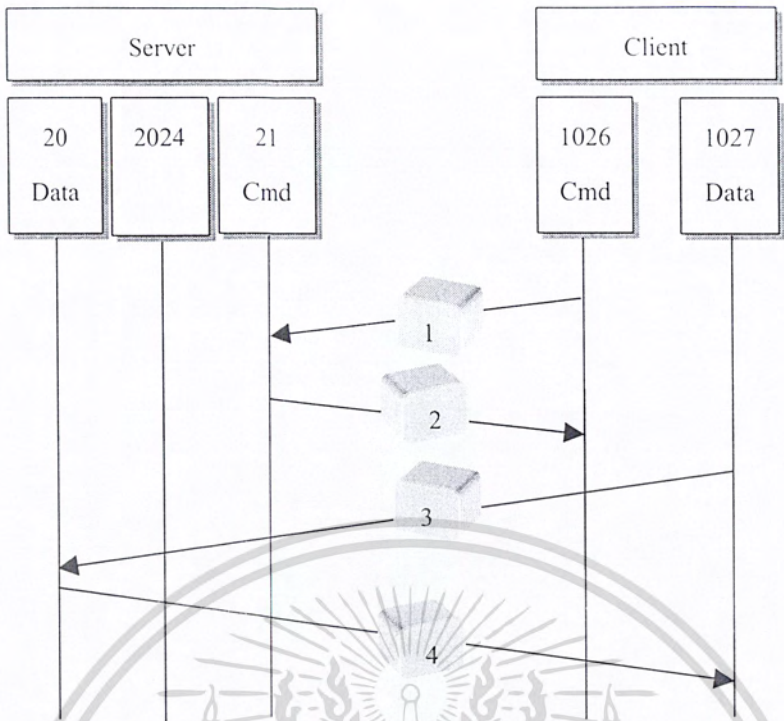
3.6.2 การทำงานแบบพาสซีฟ

ในการทำงานแบบพาสซีฟนี้ ไคลเอนต์จะเป็นฝ่ายที่เริ่มการติดต่อทั้ง 2 ส่วน (ส่วนควบคุมและส่วนข้อมูล) ซึ่งเป็นการแก้ปัญหาที่ไพล์วอลล์จะบล็อกการติดต่อส่วนข้อมูลที่มาจากเซิร์ฟเวอร์ เมื่อมีการเชื่อมต่อของเอพีทีพีไคลเอนต์จะเปิด 2 พอร์ตที่สุ่มขึ้นมา ($N > 1024$ และ $N+1$) พอร์ตแรกจะใช้ติดต่อกับเซิร์ฟเวอร์บนพอร์ต 21 แต่แทนที่จะใช้คำสั่ง PORT ตามด้วยเลขพอร์ตข้อมูลที่จะให้เซิร์ฟเวอร์ติดต่อกลับมา ไคลเอนต์จะส่งคำสั่ง PASV ไปแทน ผลที่ได้ก็คือ เซิร์ฟเวอร์จะเปิดพอร์ต $P > 1024$ และส่งคำสั่ง PORT P กลับมายังไคลเอนต์ และฝั่งไคลเอนต์ก็จะติดต่อจากพอร์ต $N+1$ ไปยังพอร์ต P ของเครื่องเซิร์ฟเวอร์เพื่อการโอนย้ายข้อมูล

ไพล์วอลล์ที่ฝั่งของเซิร์ฟเวอร์ที่สนับสนุนการทำงานแบบพาสซีฟจะต้องมีการเปิดช่องจากการติดต่อสื่อสารดังนี้

- การติดต่อมายังพอร์ต 21 ของเอพีทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (การเริ่มการใช้งานของเครื่องฝั่งไคลเอนต์)
- การติดต่อจากพอร์ต 21 ของเอพีทีพีเซิร์ฟเวอร์ไปยังพอร์ต $N > 1024$ (การติดต่อไปยังพอร์ตในส่วนควบคุมของไคลเอนต์)
- การติดต่อมายังพอร์ต $P > 1024$ ของเอพีทีพีเซิร์ฟเวอร์จากที่ใดก็ได้ (ไคลเอนต์ติดต่อไปยังพอร์ตที่ระบุมาให้จากเซิร์ฟเวอร์เพื่อสร้างการเชื่อมต่อส่วนข้อมูล)
- การติดต่อจากพอร์ต $P > 1024$ ของเอพีทีพีเซิร์ฟเวอร์ไปยังพอร์ต $N > 1024$ (การส่งสัญญาณ ACKs ไปยังพอร์ตข้อมูลของไคลเอนต์)

ในขั้นตอนแรกนั้น ฝั่งไคลเอนต์จะติดต่อไปยังเซิร์ฟเวอร์ผ่านพอร์ตส่วนควบคุมและส่งคำสั่ง PASV ไป เซิร์ฟเวอร์จะตอบกลับในขั้นตอนที่ 2 ด้วยคำสั่ง PORT 2024 เป็นการบอกให้ไคลเอนต์ทราบถึงพอร์ตที่ได้เปิดไว้รอรับข้อมูล สำหรับการเชื่อมต่อส่วนข้อมูล ในขั้นที่ 3 ไคลเอนต์จะสร้างการเชื่อมต่อจากพอร์ตของไคลเอนต์ไปยังพอร์ตที่ได้รับมาของเซิร์ฟเวอร์ ในตอนสุดท้ายเซิร์ฟเวอร์ก็จะส่งสัญญาณ ACK กลับมายังพอร์ตข้อมูลของไคลเอนต์เพื่อยืนยันการเชื่อมต่อ



รูปที่ 3-11 แสดงลักษณะการเชื่อมต่อที่เกิดขึ้นในการทำงานแบบพาสซีฟ

บทที่ 4

การเข้ารหัสและการคำนวณ

4.1 ระบบของการเข้ารหัสข้อมูล (Cryptography System)

การเข้ารหัสข้อมูล คือ การทำให้ข้อมูลที่ต้องการ เป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูล ปัจจุบัน โดยอาศัยหลักการของการเข้ารหัส (Encryption) และการถอดรหัส (Decryption)

การเข้ารหัสเป็นการเปลี่ยนรูปข้อมูล โดยผ่านรูปแบบและกระบวนการแปรรูปข้อมูล ทำให้ข้อมูลที่ส่งมีรูปแบบที่ไม่เหมือนเดิม เพื่อทำให้ข้อมูลเป็นความลับ

การถอดรหัสเป็นการแปลงข้อมูลที่ได้อ่านรหัสให้กลับมาเป็นข้อมูลเดิม ซึ่งการเข้ารหัสและการถอดรหัส จะถูกควบคุมโดยคีย์ (Key)

4.1.1 ระบบการเข้ารหัสโดยใช้คีย์เดียว (Symmetric Key)

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



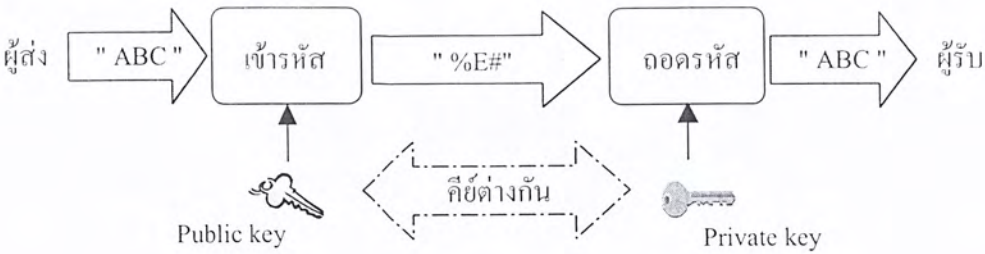
รูปที่ 4-1 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์เดียว

4.1.2 ระบบการเข้ารหัสแบบคีย์สาธารณะ

การเข้ารหัสข้อมูลระบบนี้จะประกอบด้วยคีย์ 2 อัน คือ

1. คีย์ส่วนตัว (private key) เป็นคีย์ที่จะต้องเก็บเป็นความลับ
2. คีย์สาธารณะ (public key) เป็นคีย์ที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งคีย์ทั้งสองนี้จะเป็นคีย์ที่ต่างกัน การมีวิธีการในการเข้าและถอดรหัส ดังรูป



รูปที่ 4-2 แสดงการเข้ารหัสและถอดรหัสโดยใช้คีย์สาธารณะ

4.2 รูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโทคอลทีซีพี/ไอพี

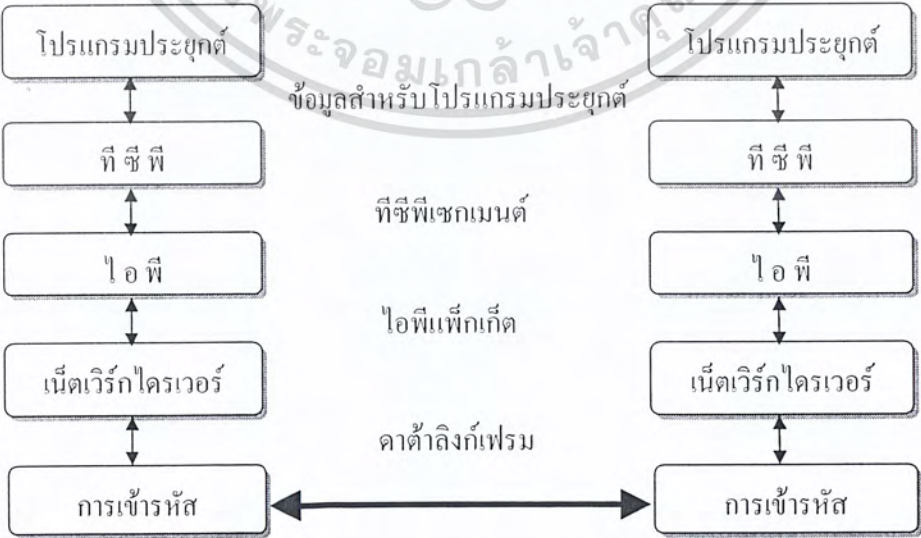
สำหรับรูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโทคอลทีซีพี/ไอพี แบ่งออกเป็น 4 ระดับ ได้แก่

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

4.2.1 การเข้ารหัสระดับการเชื่อมโยงข้อมูล (Link Level Encryption)

เป็นการเข้ารหัสในระดับต่ำสุด โดยทั่วไปมักจะเป็นการใช้อุปกรณ์พิเศษที่เรียกว่า กล่องเข้ารหัส (encryption box) ซึ่งวิธีการนี้ถูกใช้ในระบบไซเฟอร์หรือการใช้ดีไวซ์ไครเวอร์

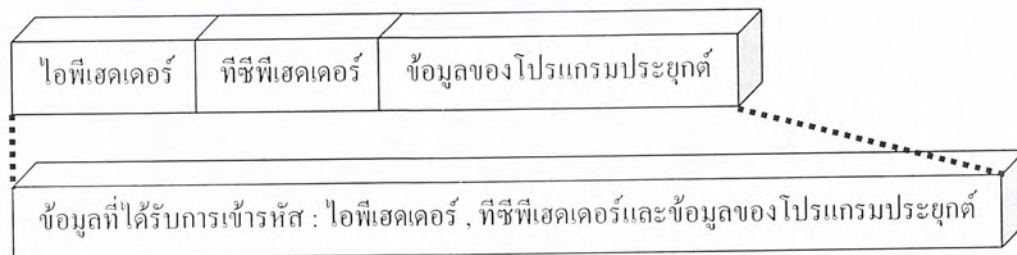
วิธีการนี้เป็นวิธีการที่ปลอดภัยที่สุด เนื่องจากข้อมูลถูกเข้ารหัสทั้งหมด นั่นคือทั้งข้อมูลที่ใช้งานและข้อมูลที่เป็นโครงสร้างของโพรโทคอล แต่ด้วยวิธีการนี้เป็นวิธีการที่ค่อนข้างมีค่าใช้จ่ายที่สูงเนื่องจากอุปกรณ์สำหรับการสื่อสารข้อมูลทั้งหมด จะต้องสนับสนุนการเข้ารหัสนี้ด้วย



รูปที่ 4-3 การเข้ารหัสในระดับการเชื่อมโยงข้อมูล

4.2.2 การเข้ารหัสระดับเครือข่าย (Network Level Encryption)

เป็นการเข้ารหัสในไอพีแพ็คเกจ (IP packet) มีลักษณะดังภาพที่ 4-4

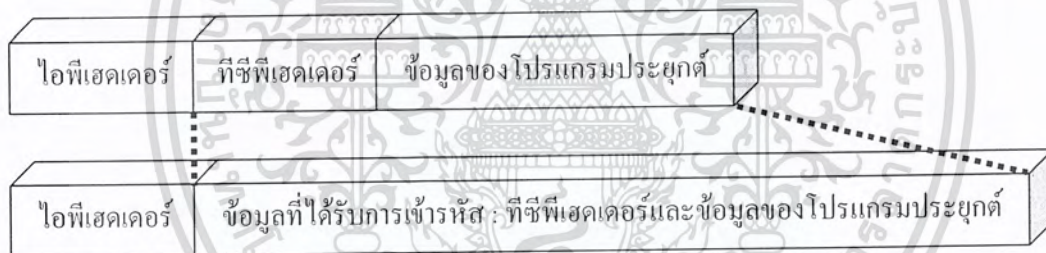


รูปที่ 4-4 การเข้ารหัสระดับเครือข่าย

4.2.3 การเข้ารหัสระดับทรานสปอร์ต (Transport Level Encryption)

การเข้ารหัสระดับทรานสปอร์ตเป็นการเข้ารหัสในส่วนของทีซีพีเซกเมนต์ ซึ่งได้แก่ ทีซีพีเฮดเดอร์และ ข้อมูลของโปรแกรมประยุกต์

การเข้ารหัสในรูปแบบนี้ทำให้การส่งข้อมูลไปตามเครือข่ายสามารถทำได้โดยไม่จำเป็นต้องเปลี่ยนแปลงอุปกรณ์ใด เนื่องจากโครงสร้างในส่วนของไอพีไม่มีการเปลี่ยนแปลง

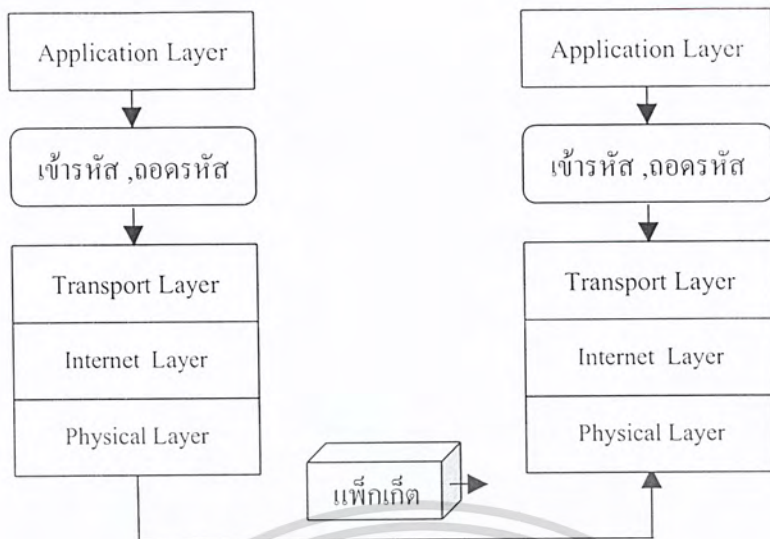


รูปที่ 4-5 การเข้ารหัสระดับทรานสปอร์ต

4.2.4 การเข้ารหัสระดับโปรแกรมประยุกต์ (Application Level Encryption)

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

สำหรับโครงการนี้ จะทำการเข้ารหัสข้อมูลในช่วงระหว่างชั้นแอปพลิเคชันและชั้นทรานสปอร์ตในระบบโอเอสไอโมเดลของโพรโตคอลทีซีพี/ไอพี ดังภาพที่ 4-6



รูปที่ 4-6 แสดงการเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดลของทีซีพี/ไอพี

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

4.3 การเข้ารหัสแบบ DES (Data Encryption Standard)

4.3.1 ประวัติและที่มาของ DES

ในปลายทศวรรษที่ 1960 บริษัท IBM ได้จัดตั้งโครงการวิจัยทางด้าน Computer Cryptography ซึ่งนำโดย Horst Feistel ซึ่งโครงการนี้เสร็จสิ้นในปี 1971 ซึ่งผลงานวิจัยของโครงการนี้คือ 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

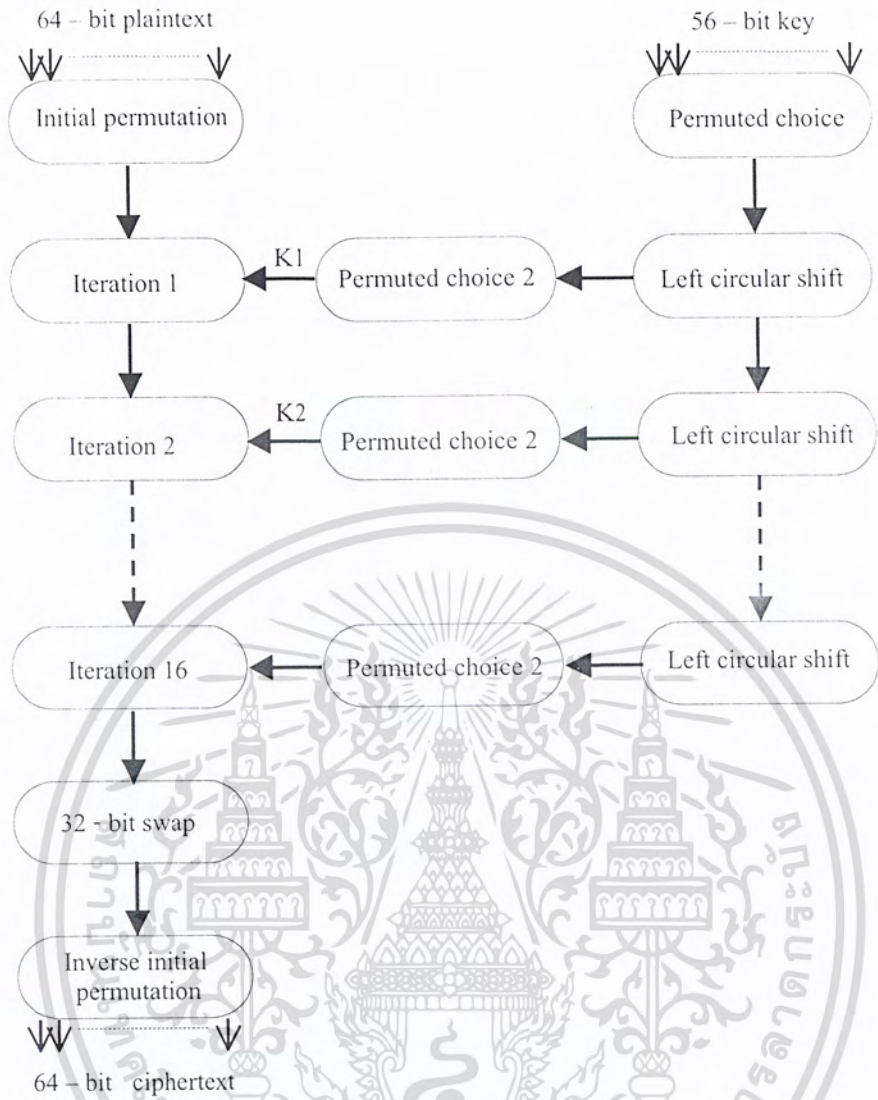
4.3.2 รายละเอียดของ DES

เป็นวิธีการเข้ารหัสที่ใช้อย่างแพร่หลายที่เป็นพื้นฐานบน 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) แต่ก็ยังเป็นข้อถกเถียงกันอย่างมากถึงเรื่อง DES นั้นจะปลอดภัยหรือไม่และมีความปลอดภัยมากน้อยแค่ไหน แต่จนถึงปัจจุบันเรายังไม่พบช่องโหว่ของ 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 อย่างไรก็ดีตาม 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) ซึ่งจะทำการเรียงเรียงบิตใหม่เพื่อผลิต “permuted input” ข้อมูลที่มีการสลับตำแหน่ง
- เฟสที่ 2 จะทำฟังก์ชันเดียวกัน 16 ครั้ง ซึ่งฟังก์ชันนี้รวมการทำ permutation และ substitution ซึ่งผลลัพธ์ที่ได้จากการทำทั้งหมด 16 ครั้งนี้จะได้ข้อมูลขนาด 64 บิตโดยใช้ทั้งข้อมูลที่ไม่ได้ผ่านการเข้ารหัสและคีย์ในการทำ ซึ่งข้อมูลที่มีขนาด 64 บิตที่ได้แบ่งเป็น 2 ด้านคือซ้ายและขวา ทั้งหมดจะถูกสวอป เพื่อผลิต 64 บิตที่เป็น preoutput
- เฟสที่ 3 จะนำ preoutput ผ่านเข้าไปยังส่วนที่เรียกว่า “Inverse initial permutation” : IP^{-1} ซึ่งทำหน้าที่ inverse ตัว initial permutation function ซึ่งทั้งหมดจะผลิต 64 บิตที่เรียกว่า ข้อมูลที่ผ่านการเข้ารหัสแล้ว (Ciphertext)



รูปที่ 4-7 แสดงการขั้นตอนการทำงานของ DES

4.3.3 การสลับตำแหน่งข้อมูลเริ่มต้น (Initial Permutation)

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

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

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

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

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

4.3.4 รายละเอียดของการทำฟังก์ชันในแต่ละรอบ

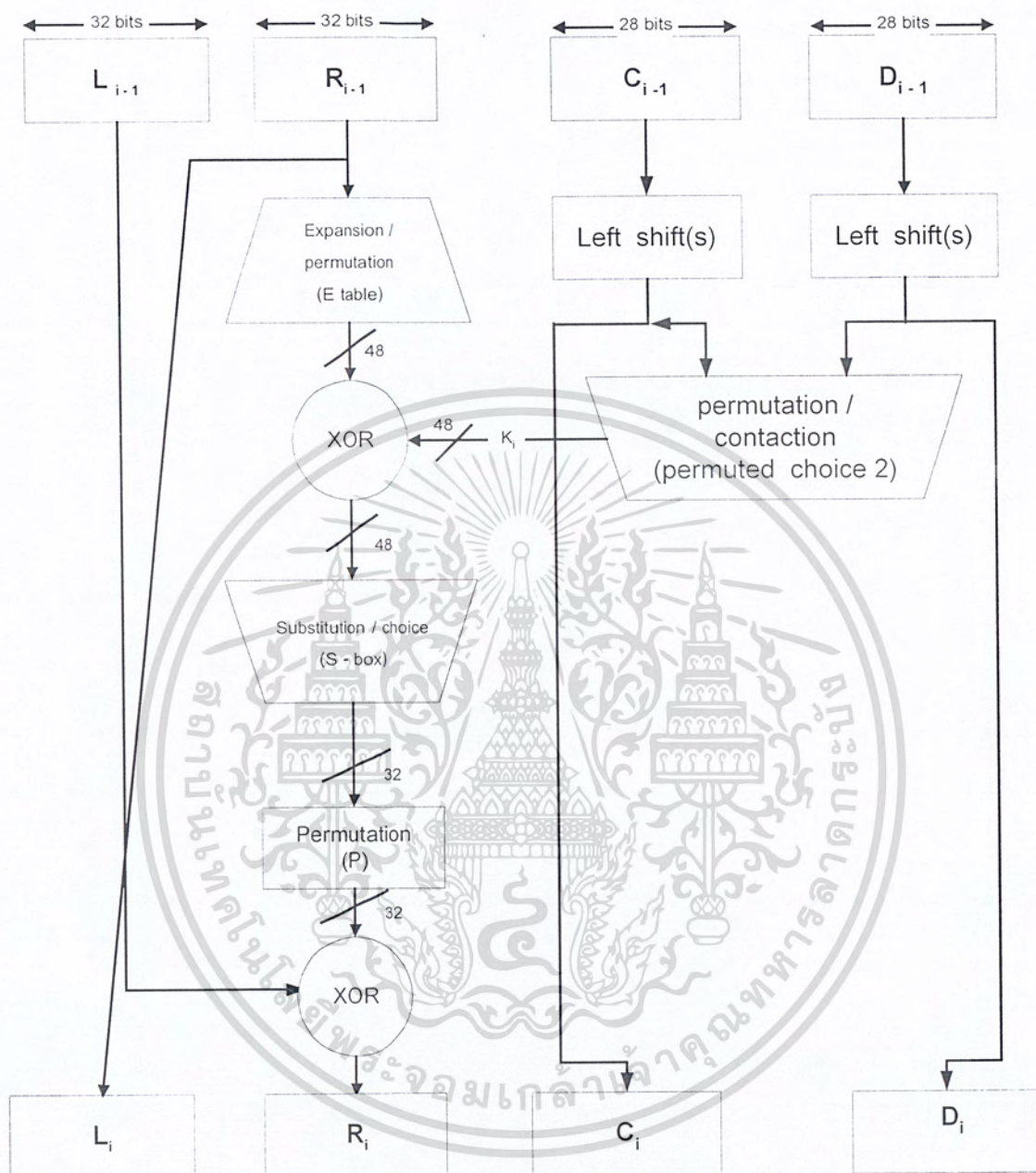
ข้อมูลที่เข้ามามีขนาด 64 บิต โดยจะทำการแบ่งข้อมูลเป็น 2 ส่วนด้วยกันขนาดละ 32 บิตเท่ากัน (แบ่งเป็นซ้ายกับขวา) ซึ่งกระบวนการทำในแต่ละครั้งสามารถสรุปเป็นสูตรได้ดังนี้

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

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

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 4.8



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

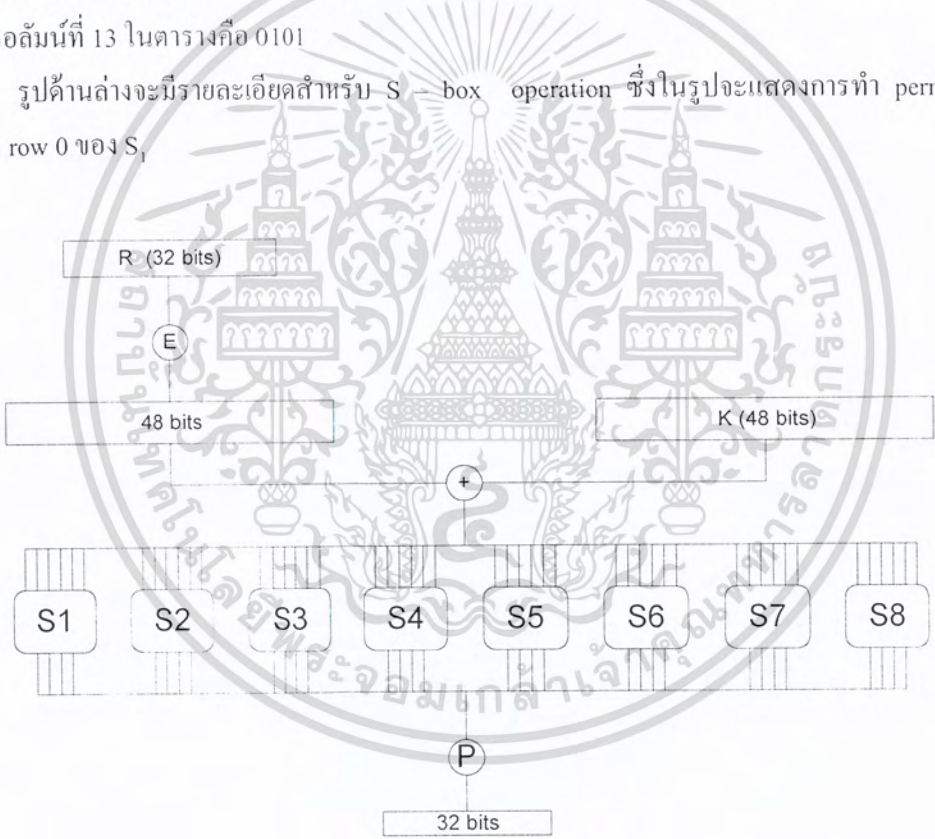
(a) Initial Permutation (IP)																
Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Form input bit	58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Form input bit	62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
Output bit	33	34	35	36	37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

ตารางที่ 4-1 แสดง Permutation ของ DES

โดยคีย์ K_i ที่ใช้ในแต่ละรอบจะมีขนาด 48 บิตและอินพุตด้านขวา(R)มีขนาด 32 บิต ดังนั้นจึงต้องมีการขยายขนาดจาก 32 บิตให้เป็น 48 บิต โดยใช้ตารางที่ได้มีการกำหนด permutation และ expansion 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 บิตและผลิตเอาต์พุตขนาด 4 บิต ซึ่งตารางด้านล่างจะแสดง DES S – box โดยมีวิธีการในการแปลงอินพุตขนาด 6 บิตให้กลายเป็นเอาต์พุตขนาด 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 จะได้ตำแหน่งของคอลัมน์ก็คือ คอลัมน์ที่ 13 ดังนั้นค่าที่ตำแหน่งแถวที่ 1 และคอลัมน์ที่ 13 ในตารางคือ 0101

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



รูปที่ 4-9 แสดงการคำนวณ $f(R,K)$

Column Number

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

Row

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
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

S1

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

S2

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
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

S3

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
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

S4

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
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

S5

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

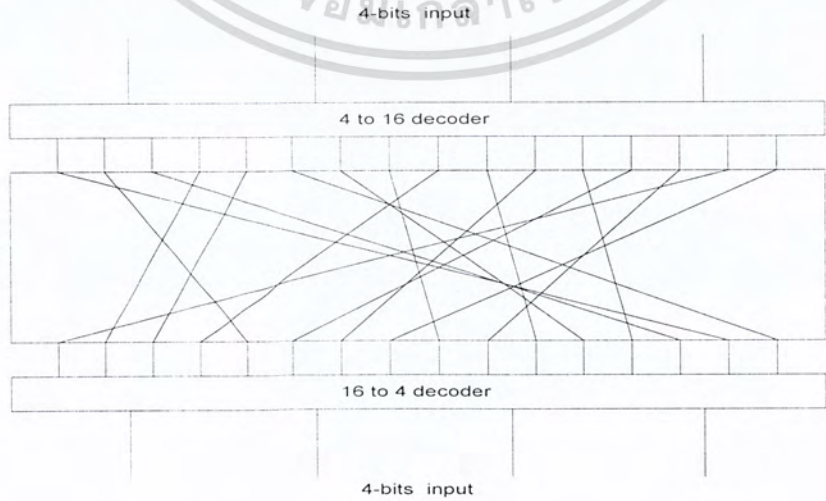
S6

Column Number																	
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15																	
Row																	Box
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S7
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	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S8
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	

ตารางที่ 4-2 แสดงขั้นตอนใน S-box ทั้ง 8 ชุด

4.3.5 การสร้างคีย์ (Key Generation)

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



รูปที่ 4-10 แสดงการทำ Permuted Choice

(a) Permuted Choice One(PC-1)

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

(b) Permuted Choice Two (PC-2)

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

(c) Schedule of Left Shifts

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

ตารางที่ 4-3 แสดงการสร้างคีย์ในแต่ละรอบฟังก์ชัน

4.3.6 การถอดรหัสข้อมูล DES

กระบวนการถอดรหัสโดยใช้ DES นั้นเหมือนกับขั้นตอนในการเข้ารหัส ซึ่งมีขั้นตอนในการทำงานดังนี้ คือ นำข้อมูลผ่านการเข้ารหัสแล้ว (Ciphertext) มาเป็นอินพุตแต่จะมีการใช้คีย์ (K_1) ที่มีลำดับย้อนกลับกับคีย์ที่ใช้ในการเข้ารหัสเช่น K_{16} , K_{15} เป็นคีย์แรกและคีย์ถัดไปในการเข้ารหัสแทนดังรูป ด้านซ้ายมือเป็นขั้นตอนการเข้ารหัสและด้านขวามือเป็นขั้นตอนในการถอดรหัส

เราจะแสดงถึงผลลัพธ์ของขั้นตอนแรกในการกระบวนการถอดรหัส ซึ่งจะเท่ากับ 32 บิตที่ถูกสวอปจากอินพุตของการทำทั้งหมด 16 รอบของการเข้ารหัส เริ่มจาก

$$L_{16} = R_{15}$$
$$R_{16} = L_{15} \oplus f(R_{15}, K_{16})$$

ในด้านการถอดรหัส

$$L_{d1} = R_{d0} = L_{16} = R_{15}$$
$$R_{d1} = L_{d0} \oplus f(R_{d0}, K_{16})$$

$$\begin{aligned} &= R16 \oplus f(R15,K16) \\ &= [L15 \oplus f(R15,K16)] \oplus f(R15,K16) \end{aligned}$$

ซึ่งคุณสมบัติของ XOR ที่สำคัญคือ

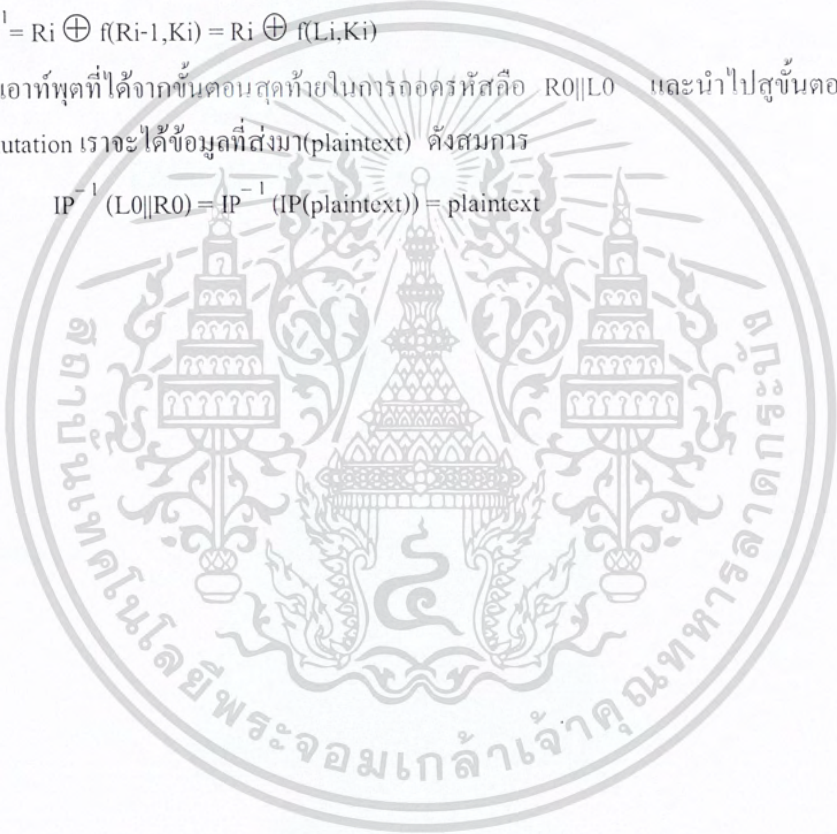
$$\begin{aligned} [A \oplus B] \oplus C &= A \oplus [B \oplus C] \\ D \oplus D &= 0 \\ E \oplus 0 &= E \end{aligned}$$

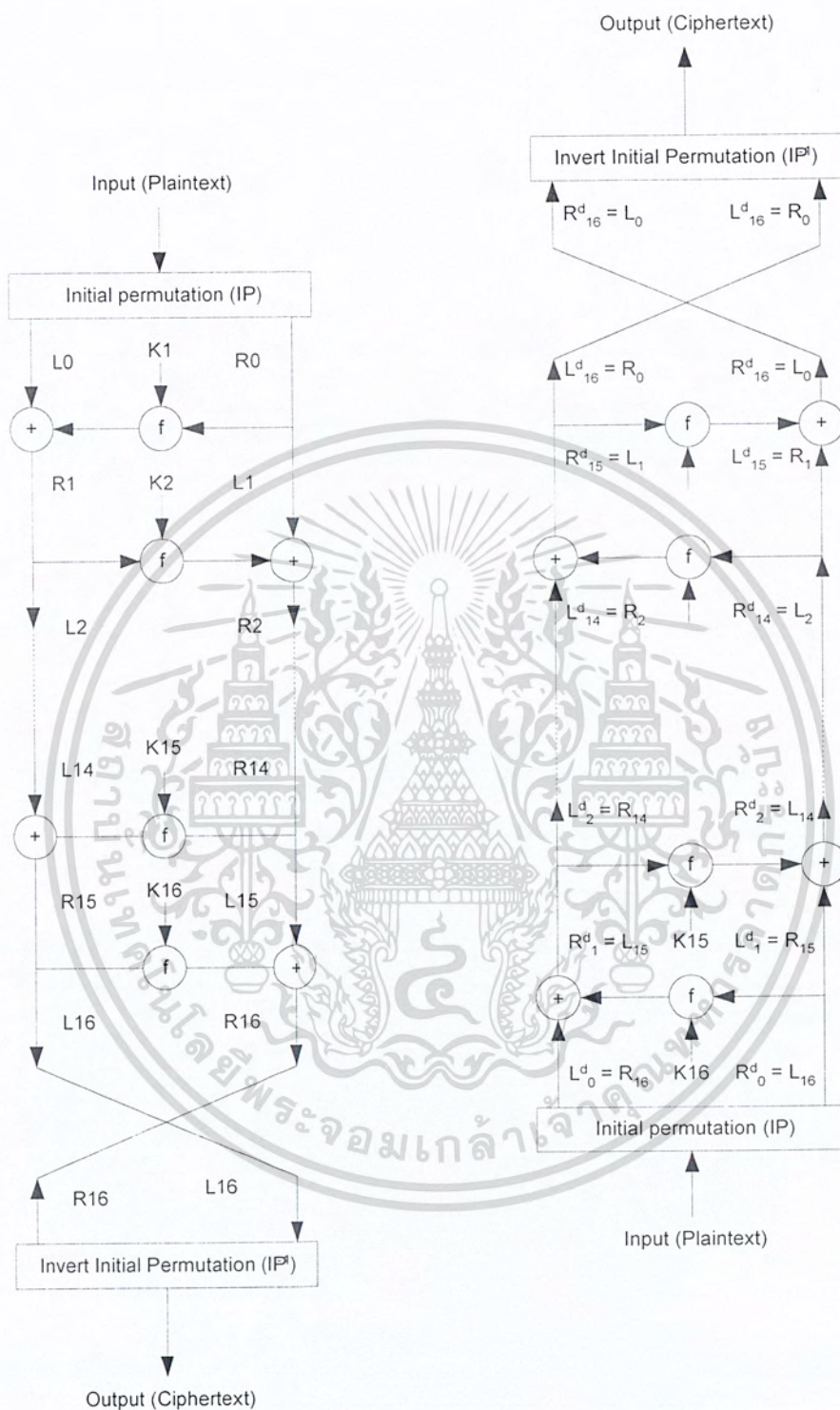
ดังนั้นเรามี $Ld1 = R15$ และ $Rd1 = L15$ จะได้เอาท่พุดของขั้นตอนแรกในการถอดรหัสคือ $L15||R15$ เราสามารถเขียนเป็นสมการในการถอดรหัสได้ดังนี้คือ

$$\begin{aligned} R_i^{-1} &= L_i \\ L_i^{-1} &= R_i \oplus f(R_{i-1},K_i) = R_i \oplus f(L_i,K_i) \end{aligned}$$

ซึ่งผลสุดท้ายเอาท่พุดที่ได้จากขั้นตอนสุดท้ายในการถอดรหัสคือ $R0||L0$ และนำไปสู่ขั้นตอนในการทำ Inverse Permutation เราจะได้ข้อมูลที่ส่งมา(plaintext) ดังสมการ

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

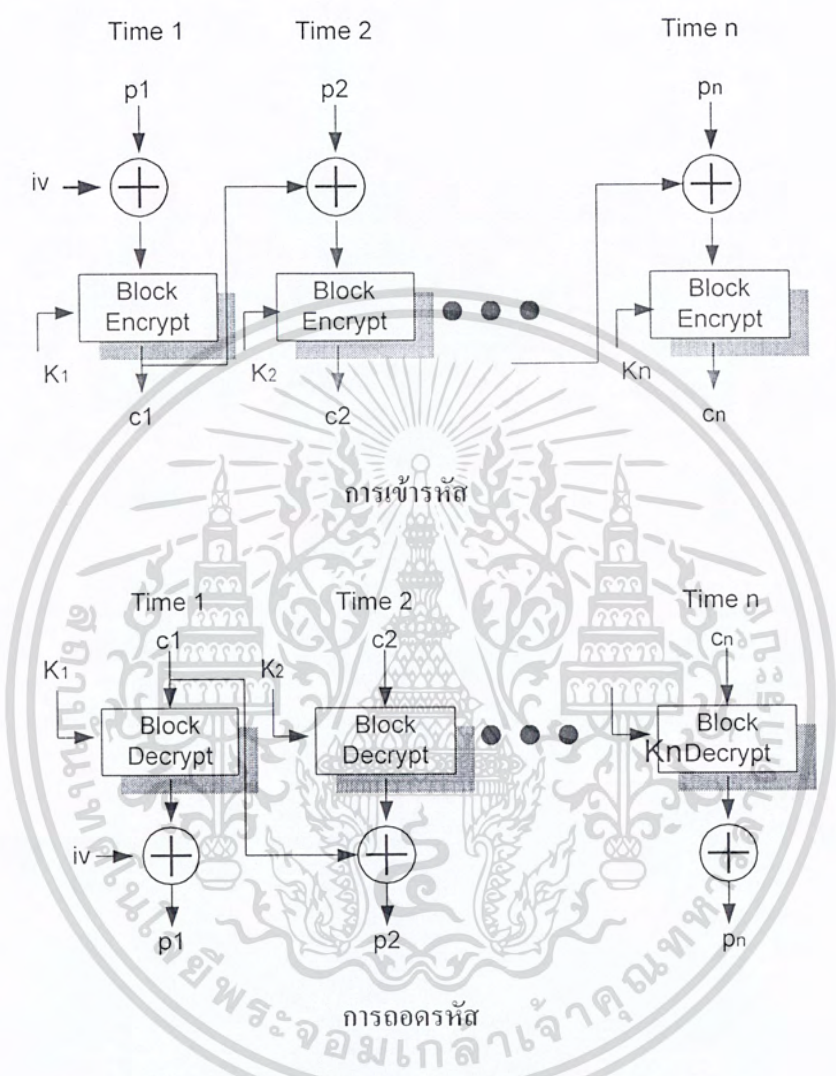




รูปที่ 4-11 แสดงคีย์และขั้นตอนการเข้ารหัสและถอดรหัส DES

4.3.7 โหมด CBC (Cipher Block Chaining)

เป็นวิธีการเข้ารหัสที่พัฒนามาจาก DES ช่วยทำให้ข้อมูลที่ส่งยังมีความปลอดภัยมากยิ่งขึ้น โดยมีวิธีการคือ จะนำข้อมูลที่ผ่านการเข้ารหัสของข้อมูลตัวก่อนมา XOR กับข้อมูลที่ยังไม่ได้เข้ารหัสของตัวถัดไปก่อนจะทำการเข้ารหัสแบบ DES ตามปกติดังรูป



รูปที่ 4-12 แสดงการเข้ารหัสและถอดรหัสของ DES CBC

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

$$C_n = E_k[C_{n-1} \oplus P_n]$$

สมการในการถอดรหัส

$$D_k[C_n] = D_k[E_k(C_{n-1} \oplus P_n)]$$

$$Dk[C_n] = C_n^{-1} \oplus P_n$$

$$C_n^{-1} \oplus Dk[C_n] = C_n^{-1} \oplus C_n^{-1} \oplus P_n = P_n$$

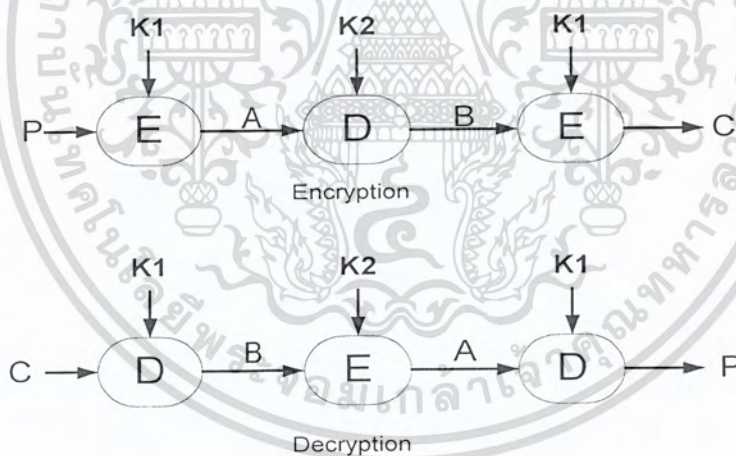
จะสังเกตเห็นได้ว่าบล็อกแรกของข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) จะมีการนำ iv (Initialization Vector) มาทำการ XOR กับข้อมูลที่ไม่ได้เข้ารหัส (plaintext) ก่อนจะมีการเข้ารหัส DES ตามปกติ และในส่วนของการถอดรหัสข้อมูลก็จะใช้ iv ในการถอดรหัสข้อมูลเช่นเดียวกัน ดังนั้นจึงจำเป็นต้องมีข้อตกลงกันระหว่างผู้ส่งกับผู้รับก่อนว่าจะใช้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

4.4 การเข้ารหัสแบบ 3DES (Triple DES)

เป็นที่ทราบกันว่ายี่สิบมีความยาวมากขึ้น การถอดรหัสยิ่งทำได้ยากยิ่งขึ้น นอกจากนี้เราก็สามารถเพิ่มสมรรถนะของการเข้ารหัสให้มากขึ้นได้โดย การเข้ารหัสหลายๆ ครั้ง

อย่างไรก็ตามการเข้ารหัสครั้งที่สองโดยใช้คีย์ที่ต่างจากครั้งแรกนั้น มิได้หมายความว่าข้อมูลจะมีความปลอดภัยเพิ่มขึ้นเป็น 2 เท่า ถ้าการเข้ารหัสดังกล่าวเป็นการเข้ารหัสโดยเทคนิคทางคณิตศาสตร์

DES มิได้อยู่ในกลุ่มดังกล่าว แต่การที่จะทำให้ DES มีความปลอดภัยมากขึ้นเป็น 2 เท่า นั้นจำเป็นที่เราจะต้องเข้ารหัสถึง 3 ครั้ง



รูปที่ 4-13 แสดงการเข้ารหัสและถอดรหัสแบบ 3DES

ขั้นตอนของการเข้ารหัสแบบ 3 ครั้งนั้นมีดังนี้

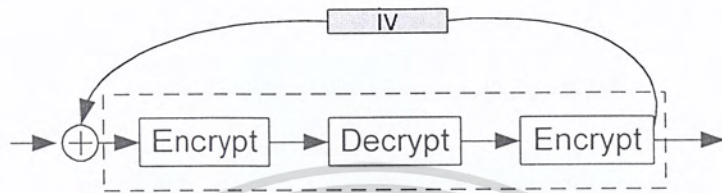
1. เข้ารหัสโดยใช้คีย์ตัวแรก
2. ถอดรหัสโดยใช้คีย์ตัวที่สอง
3. เข้ารหัสโดยใช้คีย์ตัวที่สาม

(อาจใช้คีย์ตัวที่ 1 และ 3 เป็นตัวเดียวกันก็ได้ แต่ประสิทธิภาพของการเข้ารหัสก็จะลดลง)

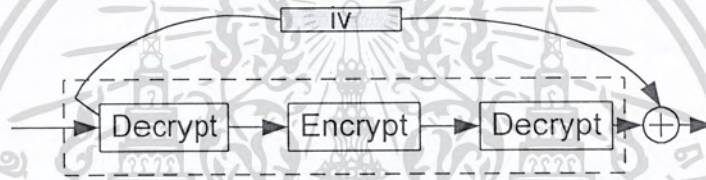
4.4.1 Triple DES โหมด CBC (3DES CBC Mode)

เนื่องจากการเข้ารหัสแบบ DES นั้นมีโหมคในการเข้ารหัสแบบ CBC ซึ่งการเข้ารหัส แบบ 3DES นี้ได้มีการประยุกต์มาจาก DES จึงได้มีการพัฒนา 3DES นี้ให้มีโหมค CBC ด้วย ซึ่งในการเข้ารหัสแบบ 3DES นี้ได้พัฒนาโหมคนี้ออกเป็น 2 แบบ

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



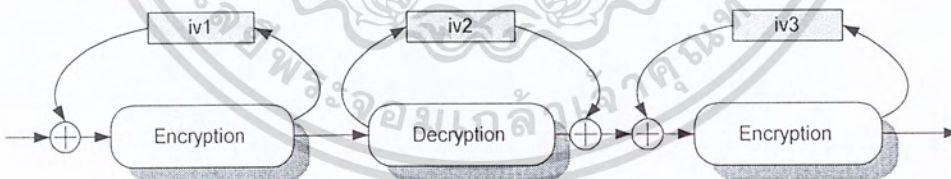
การเข้ารหัส



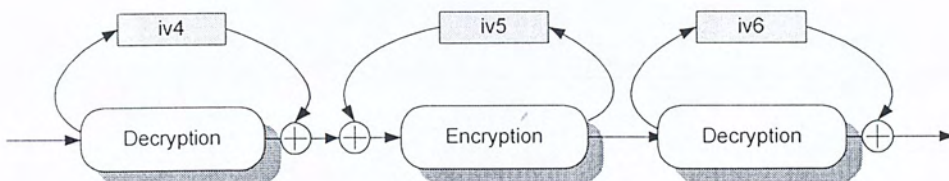
การถอดรหัส

รูปที่ 4-14 แสดงการเข้ารหัสและถอดรหัสแบบ Triple DES โหมด inner CBC

- แบบ outer CBC การเข้ารหัสแบบนี้แต่ละโหมคในการเข้ารหัสหรือถอดรหัสจะมี iv เฉพาะของแต่ละโหมค ดังรูป 4-15



การเข้ารหัส

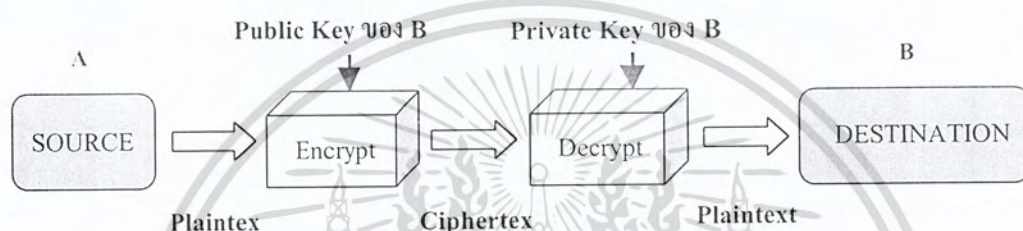


การถอดรหัส

รูปที่ 4-15 แสดงการเข้ารหัสและถอดรหัสแบบ Triple DES โหมด outer CBC

4.5 การเข้ารหัสแบบ RSA

RSA ถูกคิดค้นขึ้นในปี 1977 โดยที่ชื่อ RSA ได้มาจากอักษรตัวแรกของนามสกุลของผู้ร่วมกันคิดค้นคือ Ron Rivest, Adi Shamir และ Leonard Adleman เป็นวิธีการเข้ารหัสแบบคีย์สาธารณะที่เรียกว่า พับลิคคีย์ (Public-Key Cryptosystem) เพื่อแก้ไขปัญหการทำให้คีย์เป็นความลับ (Secret-Key Cryptosystem) โดยสมาชิกแต่ละคนจะต้องมีคีย์ 2 ชนิดคือ คีย์ส่วนตัวหรือไพรเวตคีย์ (Private Key) และคีย์สาธารณะหรือพับลิคคีย์ (Public Key) โดยคีย์ส่วนตัวจะถูกเก็บไว้เป็นความลับ ส่วนคีย์สาธารณะนั้นจะเปิดเผยให้ใครก็ได้ที่ต้องการส่งเอกสารให้แก่นั้น หลักการทำงานของ การเข้ารหัสแบบ RSA คือ ข้อมูลที่ถูกเข้ารหัสด้วยคีย์สาธารณะของผู้ใด จะถูกถอดรหัสได้ด้วยคีย์ส่วนตัวของผู้นั้นเท่านั้น การทำงานของระบบพับลิคคีย์สามารถอธิบายได้ดังต่อไปนี้



รูปที่ 4-16 แสดงขั้นตอนการทำ Public – Key Cryptosystem

4.5.1 หลักการทำงานของ RSA

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

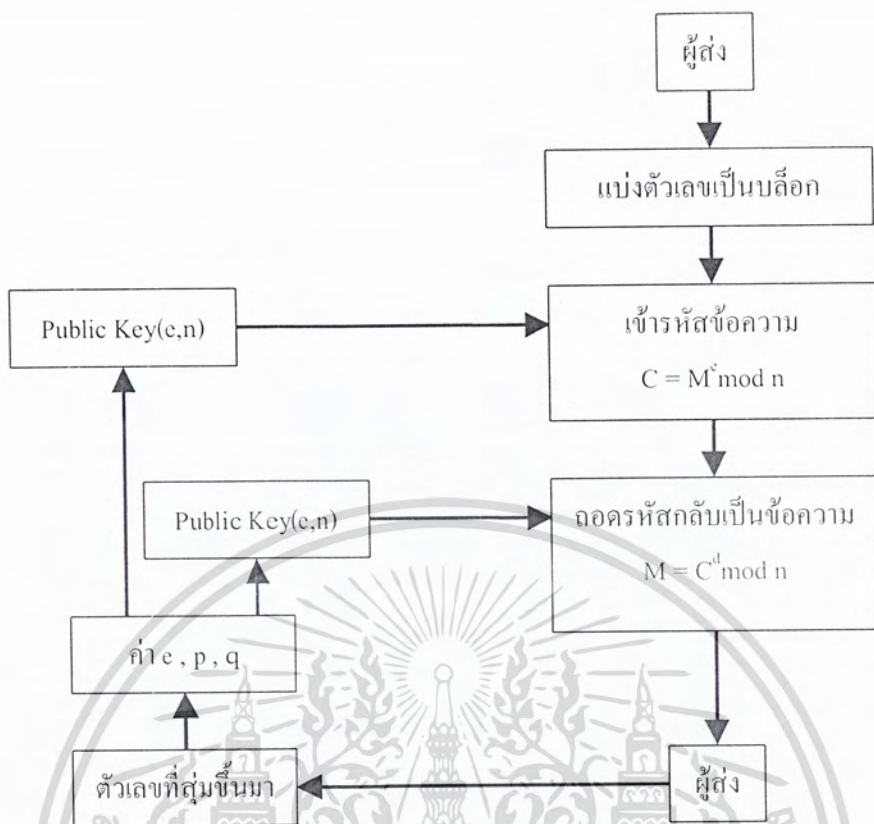
$$e \cdot d \bmod (p-1)(q-1) = 1 \quad \dots\dots\dots (1)$$

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

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

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

และเพราะมีแต่นุญมีเท่านั้นที่รู้ค่า d นุญมีเท่านั้นที่จะถอดรหัสได้ คุณสมบัติสำคัญประการหนึ่งของ RSA คือจากสมการ (2) ในทางกลับกันถ้าเราเข้ารหัสด้วยไพรเวตคีย์ เราก็สามารถถอดรหัสด้วยพับลิคคีย์ได้ด้วยเช่นกัน คุณสมบัติข้อนี้เองที่ทำให้ RSA มีประโยชน์มาก เพราะใช้ในการเซ็นระบบดิจิทัลได้ด้วย



รูปที่ 4-17 แสดงการเข้ารหัสแบบ RSA

ตัวอย่างขั้นตอนการเข้ารหัสแบบ RSA ซึ่งมีขั้นตอนในการหาคีย์ ดังนี้

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

$$77 * 3 = 231 = 2 * 96 + 3$$

ดังนั้น Public Key คือ $\{3, 119\}$ และมีค่า Private Key คือ $\{77, 119\}$

วิธีทำลาอริส RSA ที่รู้จักกันดีที่สุดคือการหาค่า d นั้นเองจากสมการที่ (1) เราอาจหาค่า d ได้หากรู้ค่า p และ q แต่เนื่องจาก p และ q เป็น prime number ที่ $p * q = n$ ดังนั้นการทำลาอริส RSA ขึ้นอยู่กับการแยกตัวประกอบของ n นั้นเอง แต่วิธีการแยกตัวประกอบของ n นั้นไม่ง่ายเลยหาก n มีค่ามาก R.Rivest ได้คำนวณไว้ในปี 1992 ว่าจะต้องใช้เงินประมาณ 8.3 ล้านเหรียญสหรัฐในการแยกตัวประกอบของ n ที่มีความยาว 512 บิต ค่านี้อาจลดลงได้ในอนาคต ดังนั้นสำหรับข้อมูลที่มีความสำคัญมากกว่า อาจจำเป็นต้องใช้ n ที่มีความมากถึง 700 หรือ 1000 ก็ได้

จะเห็นได้ว่าไม่ว่าการเข้ารหัสหรือถอดรหัส RSA ก็จำเป็นต้องใช้การยกกำลังในระบบมอดุโล ซึ่งการยกกำลังนั้นสามารถทำได้โดยใช้การใช้อัลกอริทึมระบบมอดุโลมาต่ออนุกรมกัน ดังนั้นความเร็วของทั้งการเข้ารหัสและถอดรหัสจึงขึ้นกับความเร็วของวงจรระบบมอดุโลเป็นอย่างมาก นี่เองเป็นจุดอ่อนข้อหนึ่งของ RSA เมื่อเปรียบเทียบกับคีย์เดียว เช่น DES เพราะปัจจุบันการคูณในระบบมอดุโลยังค่อนข้างยุ่งยากเมื่อเปรียบเทียบกับ การแทนค่าหรือสลับตำแหน่งใน DES ได้มีการประมาณกันว่าสำหรับ n ที่มีความยาว 512 บิต การเข้ารหัสแบบ DES จะเร็วกว่า RSA ประมาณ 100 เท่า ถ้าเราทำการเข้ารหัสด้วยซอฟต์แวร์ และอาจเร็วกว่าถึง 1000 ถึง 10000 แล้วแต่ลักษณะของวงจร หากทำการเข้ารหัสด้วยฮาร์ดแวร์ โดยทั่วไปแล้วเราต้องการให้การเข้ารหัสเร็วกว่าการถอดรหัส ดังนั้นเราจึงมักเลือกให้ e มีค่าน้อยกว่า d และยิ่งกว่านั้นเรามักให้ e ของสมาชิกทุกคนมีค่าเดียวกันเพื่อให้ฮาร์ดแวร์ของวงจรเข้ารหัสสำหรับสมาชิกแต่ละคนมีลักษณะคล้ายกัน

เนื่องจาก DES และ RSA มีข้อดีข้อเสียที่แตกต่างกัน จึงไม่จำเป็นว่ารหัสชนิดใดชนิดหนึ่งจะเหมาะสมในทุกสถานการณ์ โดยทั่วไปแล้ว DES จะถูกใช้ในการเข้ารหัสข้อมูลที่มีขนาดใหญ่เพราะรวดเร็ว ในขณะที่ RSA จะถูกใช้ในระบบสื่อสารที่ไม่ยาวนานแต่ต้องการความปลอดภัยสูงในบางครั้ง RSA ยังถูกใช้ร่วมกับ DES เพื่อเสริมจุดเด่นซึ่งกันและกัน เช่น ตัวเอกสารจริงจะถูกเข้ารหัสด้วย DES โดยที่คีย์รหัส DES จะถูกเข้ารหัสด้วย RSA แล้วส่งไปด้วยกันหรือส่งไปก่อนแต่ในบางครั้ง DES อย่างเดียวก็นพอแล้วหากการแลกเปลี่ยนคีย์สามารถทำได้อย่างปลอดภัยเพียงพอ หรือในกรณีที่ผู้ส่งและผู้รับเป็นบุคคลเดียวกัน เช่น ฮาร์ดดิสก์ในคอมพิวเตอร์ส่วนตัวหรือข้อมูลส่วนตัวในสมาร์ตการ์ด

4.6 การคำนวณแบบ MD5

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

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

ขั้นที่ 2 : เติมความยาว ใส่ขนาดของข้อความโดยจะมีค่าไม่เกิน 2^{64}

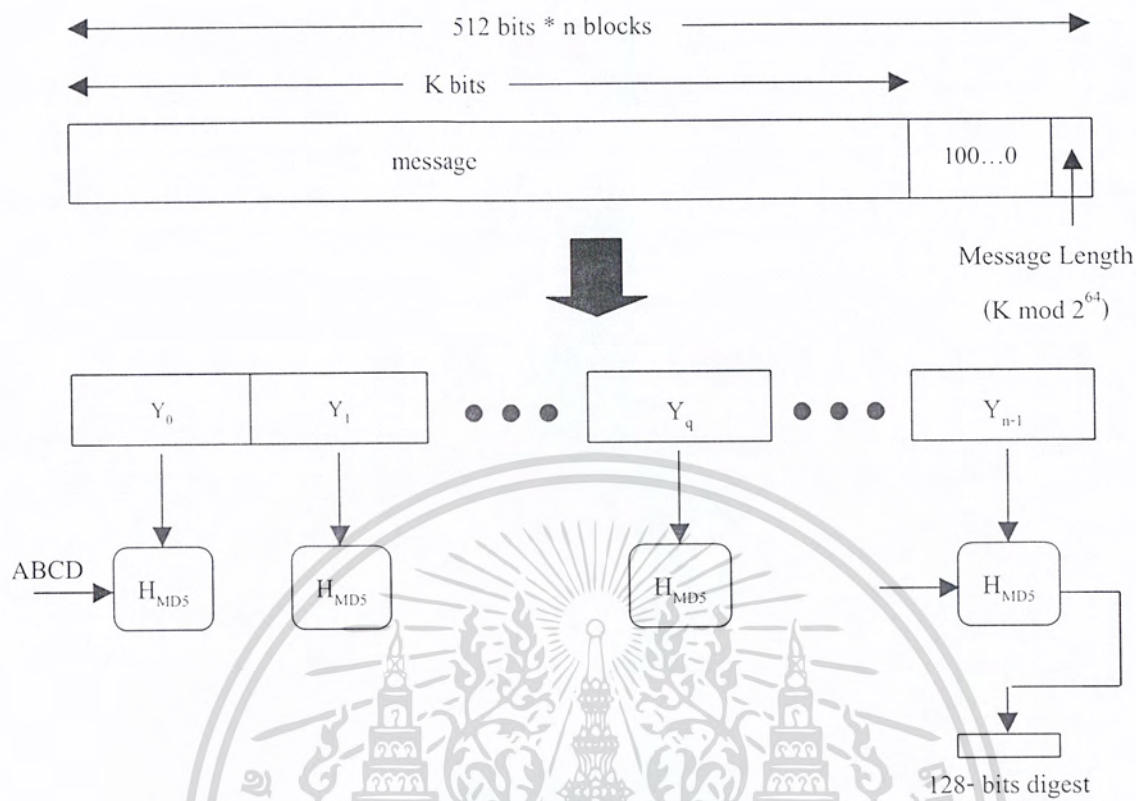
ขั้นที่ 3 : กำหนดค่าเริ่มของ MD บัฟเฟอร์ บัฟเฟอร์จะมีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการ hash โดยบัฟเฟอร์เริ่มต้นจะประกอบด้วย 32 บิต ไบต์ 4 ตัว (A,B,C,D) ซึ่งมีค่าต่อไปนี้

A = 01234567

B = 89ABCDEF

C = FEDCBA98

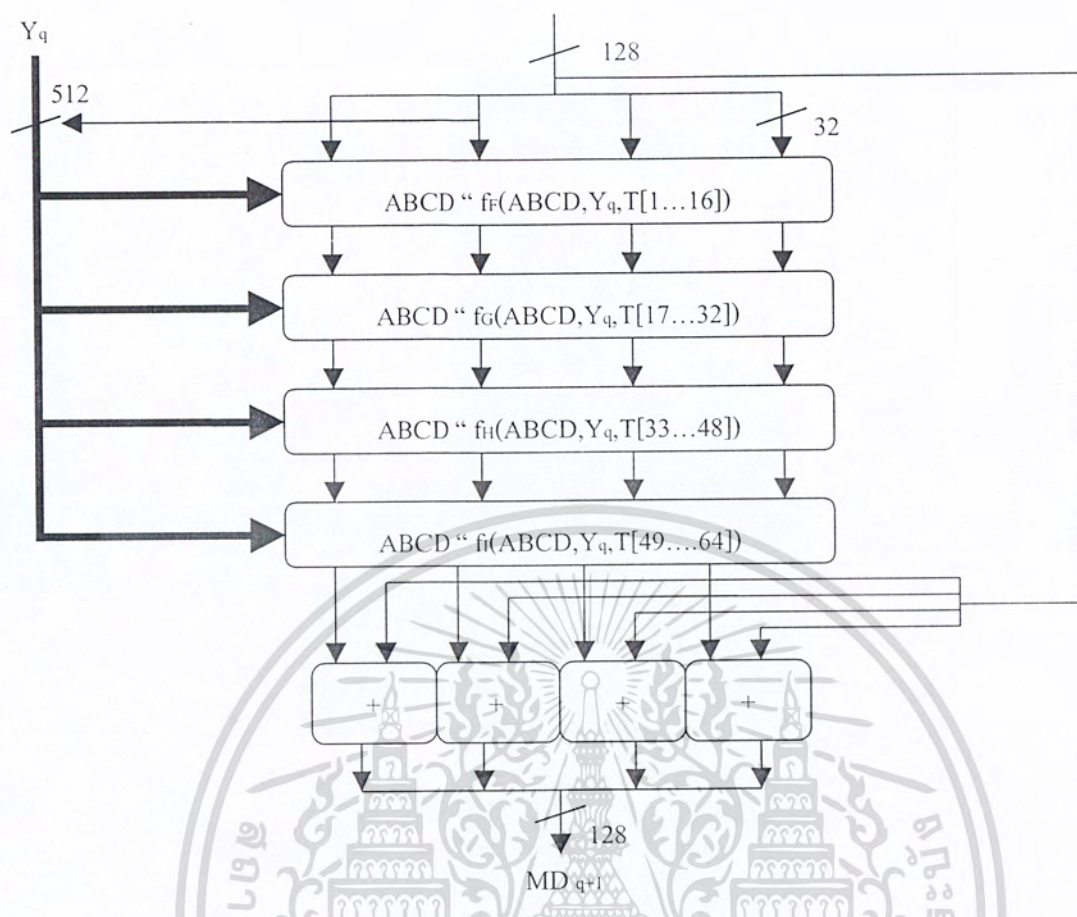
D = 76543210



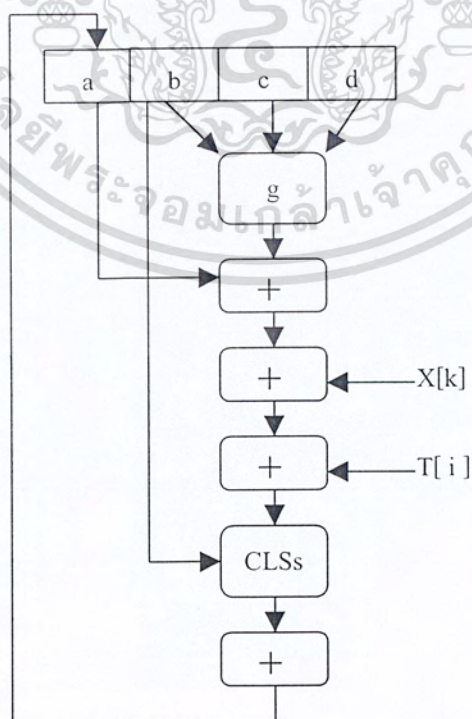
รูปที่ 4-18 แสดงการคำนวณแบบ MD5

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

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



รูปที่ 4-19 กระบวนการทำในหนึ่งบล็อกของ MD5



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

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

นำเอาท่สุดท้ายที่ได้จากการทำครั้งแรก (128 บิต) ซึ่งจะได้เป็น A,B,C,D ใหม่มาทำการโพรเซสกับบิตที่ถัดไปจนครบทั้งหมด ซึ่งจะได้ผลลัพธ์ขนาด 128 บิตด้วย



บทที่ 5

โพรโทคอลเอสเอสเอสเอช 1

โพรโทคอลเอสเอสเอสเอช 1 เป็นโปรแกรมประยุกต์โปรแกรมหนึ่งที่ใช้สามารถลือคอินเข้าใช้บริการจากเครื่องให้บริการที่อยู่ห่างไกลบนระบบเครือข่าย และสามารถทำการประมวลผล คำสั่งในเครื่องให้บริการที่อยู่ห่างไกลได้ (remote machine) พร้อมทั้งสามารถแลกเปลี่ยนข้อมูลบนระบบเครือข่ายอินเทอร์เน็ตได้อย่างปลอดภัย เนื่องจากมีวิธีการพิสูจน์ความเป็นเจ้าของที่มีประสิทธิภาพ (Strong Authentication) และระบบการติดต่อที่ปลอดภัย (Secure Communication) บนระบบเครือข่าย

5.1 ภาพรวมของโพรโทคอลเอสเอสเอสเอช 1

โพรโทคอลเอสเอสเอสเอช 1 ประกอบด้วยโปรแกรมในส่วนเครื่องที่ให้บริการ (เซิร์ฟเวอร์) และอีกส่วนหนึ่งคือส่วนเครื่องใช้บริการ (ไคลเอนต์) โดยการติดต่อระหว่างเครื่องให้บริการและเครื่องใช้บริการจะติดต่อกันโดยใช้หมายเลขไอพี (IP Address) และ ทีซีพี/ไอพีซ็อกเก็ต (TCP/IP socket) ซึ่งเป็นการติดต่อสองทิศทาง (bi-directional communication)

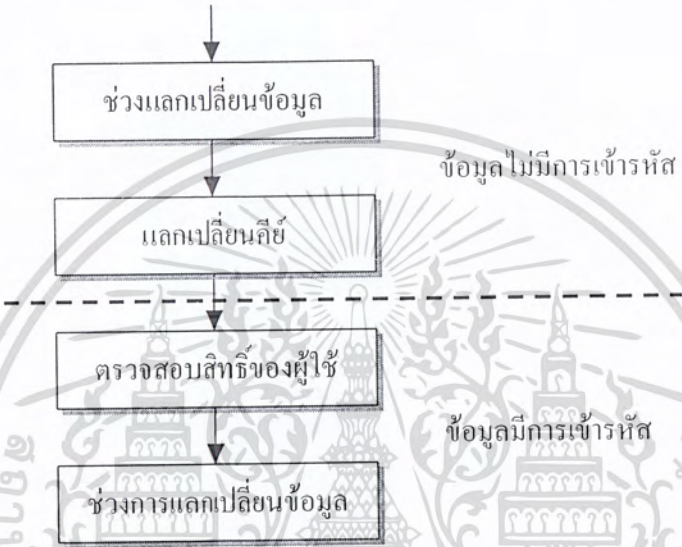
การติดต่อกันจะเริ่มการติดต่อจากเครื่องใช้บริการ โดยเครื่องให้บริการจะรอฟังจากพอร์ตที่กำหนดไว้เฉพาะ ว่ามีเครื่องที่จะใช้บริการติดต่อเข้ามายังพอร์ตนั้นหรือไม่ ซึ่งเครื่องให้บริการสามารถติดต่อกับเครื่องใช้บริการได้หลายเครื่องพร้อมๆ กัน

เมื่อเครื่องใช้บริการทำการติดต่อกับเครื่องผู้ให้บริการ หากเครื่องผู้ให้บริการยอมรับการติดต่อก็จะส่งข้อความเพื่อตรวจสอบเวอร์ชันของกันและกัน ที่เรียกว่า *Version Identification String* ของตนกลับไป เมื่อเครื่องใช้บริการได้รับก็จะทำการแปลข้อความของผู้ให้บริการ และส่งข้อความในการตรวจสอบเวอร์ชันของตนเองกลับไปเช่นกัน ซึ่งหน้าที่ของ *Version Identification String* คือ เป็นข้อความที่ใช้ในการตรวจสอบการติดต่อว่าเวอร์ชันของโปรแกรมและโพรโทคอลนั้นรองรับกันหรือไม่

หลังจากทำการตรวจสอบเวอร์ชันของกันและกันและกันแล้ว ข้อมูลในการติดต่อจะเปลี่ยนมาเป็นรูปแบบของแพ็กเก็ตซึ่งเรียกโพรโทคอลในการติดต่อนี้ว่า *Binary packet protocol* โดยเครื่องผู้ให้บริการจะเริ่มส่งการติดต่อโดยส่งคีย์สาธารณะที่เรียกว่า Host Public Key, Server Key (สำหรับรายละเอียดเกี่ยวกับคีย์ต่างๆ จะอยู่ในบทที่ 4) และข้อมูลอื่นๆไปยังเครื่องผู้ให้บริการ ตัวผู้ให้บริการจะทำการผลิตคีย์เซสชัน (Session Key) ซึ่งใช้ในการเข้ารหัสข้อมูลที่มีขนาด 256 บิต ซึ่งจะถูกเข้ารหัสแบบ RSA 2 ครั้ง และทำการส่งไปให้เครื่องผู้ให้บริการพร้อมทั้งชนิดของการเข้ารหัสข้อมูล (cipher) ที่ใช้ หลังจากนั้นทั้ง 2 ฝ่ายจะเปิดการติดต่อกันโดยข้อมูลที่ติดต่อกันนี้จะมีการเข้ารหัสโดยใช้คีย์และวิธีการที่ได้กำหนดไว้ข้างต้น

ต่อมาเครื่องผู้ให้บริการจะทำการตรวจสอบสิทธิ์ของตน โดยทำการส่งหมายเลขเพื่อป้องกันวิธีการพิสูจน์ตนและทำการพิสูจน์ตนตามแบบที่ได้ส่งไปเซิร์ฟเวอร์

เมื่อทำการตรวจสอบและพิสูจน์ตนแล้วว่าถูกต้อง เครื่องผู้ให้บริการจะทำการส่งตัวเลขเพื่อทำการเตรียมตัวสำหรับการเรียกใช้เชลล์ (Shell) และทำการเรียกใช้เชลล์เพื่อที่จะเข้าไปทำการติดต่อในโหมดเซสชันแบบอินเตอร์แอ็กทีฟ (Interactive Session Mode) ซึ่งการทำงานในโหมดนี้เป็นโหมดในการแลกเปลี่ยนข้อมูลกันและกันระหว่างผู้ที่ให้บริการกับเครื่องผู้ให้บริการและจะสิ้นสุดการติดต่อเมื่อเครื่องผู้ให้บริการส่งคำสั่งเลิกบริการ (Exit Status) ของโปรแกรมไปยังเครื่องผู้ให้บริการ



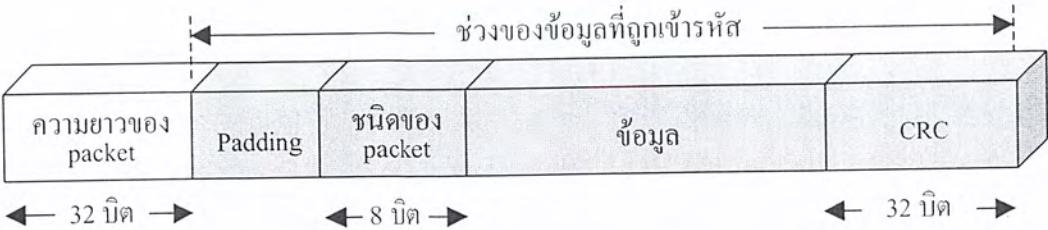
รูปที่ 5-1 แสดงขั้นตอนของโปรโตคอลเอสเอสเอส 1

5.2 ลักษณะและโปรโตคอลของข้อมูล

จากภาพรวมของโปรโตคอลที่ได้กล่าวมาข้างต้น ทำให้เราทราบวิธีการทำงาน ขั้นตอนการติดต่อและรูปแบบข้อความที่ใช้ติดต่อกัน แต่ก่อนที่จะศึกษาขั้นตอนการทำงานอย่างละเอียด เราจะทำความรู้จักกับคำที่ได้กล่าวมาข้างต้นก่อนว่าแต่ละตัวคืออะไร ทำหน้าที่อย่างไร และข้อความที่ใช้มีอะไรบ้าง

5.2.1 ไบนารีแพ็กเก็ตโปรโตคอล (Binary Packet Protocol)

ภายหลังจากการทำการตรวจสอบเวอร์ชันของกันและกัน ทั้งเครื่องให้บริการและเครื่องผู้ให้บริการจะทำการส่งแพ็กเก็ตที่มีรูปแบบเฉพาะซึ่งเรียกว่า Binary Packet Protocol รูปแบบของแพ็กเก็ตเป็นดังนี้



รูปที่ 5-2 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) ได้ง่ายขึ้น
ชนิดของแพ็กเก็ต	จะมีขนาด 8 บิต ค่า 255 กันไว้สำหรับใช้ในอนาคต
ข้อมูลที่จะส่ง	ข้อมูล (มีลักษณะเป็นแบบเลขฐานสอง) ที่จะส่ง โดยขึ้นกับชนิดของแพ็กเก็ต จำนวนของข้อมูลมีขนาดเท่ากับความยาวของแพ็กเก็ตลบ 5
CRC	เป็นข้อมูลที่ใช้ตรวจสอบ โดยมีขนาด 32 บิต โดยนำเอา padding, ชนิดของแพ็กเก็ตและฟิลด์ข้อมูล มาทำ CRC (Cyclic Redundancy Check) ด้วยโพลีโนเมียล 0xadb88320 โดย CRC นี้จะถูกคำนวณก่อนการเข้ารหัส

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

5.2.2 การเข้ารหัสแพ็กเก็ต (Packet Encryption)

แพ็กเก็ตของโปรโตคอลนี้รองรับการเข้ารหัสได้หลายชนิด โดยตั้งแต่เริ่มต้นการติดต่อ เครื่องผู้ให้บริการจะส่งบิตมาร์ค (bitmask) ของวิธีการเข้ารหัสแบบต่างๆที่เครื่องผู้ให้บริการรองรับ ตัวเครื่องผู้ให้บริการจะเลือกวิธีการในการเข้ารหัสข้อมูลและสร้างเซสชันคีย์ ขนาด 256 บิต จากนั้นทำการส่งไปให้เครื่องผู้ให้บริการ

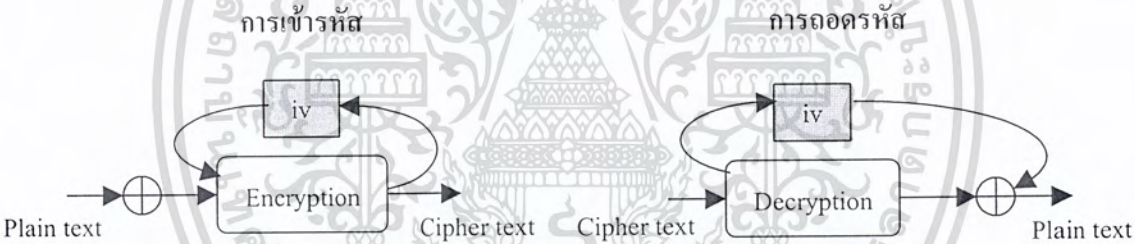
วิธีการเข้ารหัสที่รองรับและโค้ดของวิธีการเข้ารหัสแต่ละแบบคือ

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 5-1 แสดงวิธีการเข้ารหัสที่โพรโทคอลเอสเอสเอช 1รองรับ

SSH_CIPHER_DES

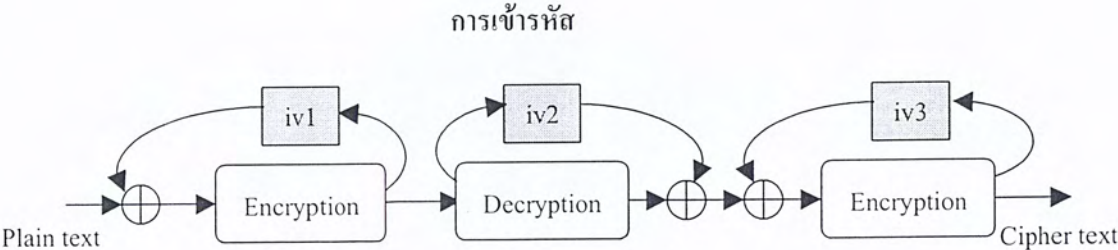
กุญแจในการเข้ารหัสนำมาจาก 8 บิตแรกของเซสชันคีย์ และบิตที่มีค่าน้อยสุดของแต่ละไบต์จะถูกตัดทิ้งไป กุญแจที่ได้จะมีขนาด 56 บิต การเข้ารหัสแบบ DES ที่ใช้นี้จะเป็นโหมด CBC ซึ่ง iv (initialization vector) ในการเริ่มต้นถูกกำหนดให้เป็น 0 ทั้งหมด (ซึ่งได้อธิบายละเอียดในบทการเข้ารหัสข้อมูล)



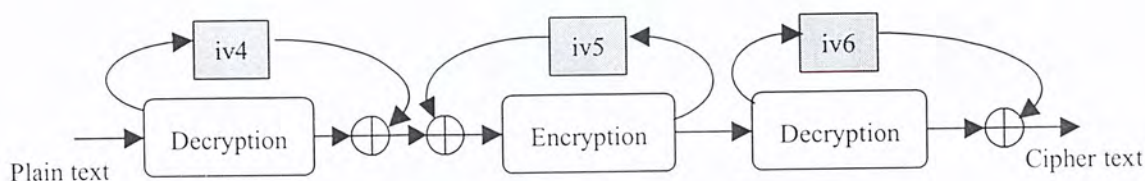
รูปที่ 5-3 แสดงการเข้ารหัสและถอดรหัสแบบ DES โหมด CBC

SSH_CIPHER_3DES

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



การถอดรหัส



รูปที่ 5-4 แสดงวิธีการเข้ารหัสและถอดรหัสแบบ 3DES โหมด CBC

สำหรับการเข้ารหัส ข้อมูลจะถูกเข้ารหัสด้วยไชเฟอร์แรกและถอดรหัสด้วยไชเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกเข้ารหัสอีกครั้งด้วยไชเฟอร์ 3 ส่วนการถอดรหัส ข้อมูลจะถูกถอดรหัสด้วยไชเฟอร์แรกและเข้ารหัสด้วยไชเฟอร์ที่ 2 และครั้งสุดท้ายจะถูกถอดรหัสอีกครั้งด้วยไชเฟอร์ 3 โดยคีย์ที่ใช้สำหรับไชเฟอร์แรกจะถูกนำมาจาก 8 ไบต์แรกของคีย์เซชัน และคีย์สำหรับไชเฟอร์ที่ 2 มาจาก 8 ไบต์ถัดไปของคีย์เซชัน และคีย์สำหรับไชเฟอร์ที่ 3 จะเป็น 8 ไบต์ถัดไปของคีย์เซชันเช่นกัน (ค่า iv จะไม่เกี่ยวข้องกัน)

5.2.3 ชนิดของข้อมูลที่ถูกเข้ารหัส (Data Type Encoding)

ฟิลด์ข้อมูลของแต่ละแพ็กเก็ตจะเก็บข้อมูลที่ถูกเข้ารหัสตามวิธีที่ได้อธิบายไป ซึ่งจะประกอบรายการของข้อมูลหลายๆตัว แต่ละรายการจะมีรหัสในแต่ละแบบและถูกนำมาใช้โดยการนำแต่ละรายการมาต่อกัน

ข้อมูลแต่ละชนิดจะถูกเก็บตามนี้คือ

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

5.2.4 หมายเลขที่ซีพี/ไอพีพอร์ต (TCP/IP Port Number and Other Option)

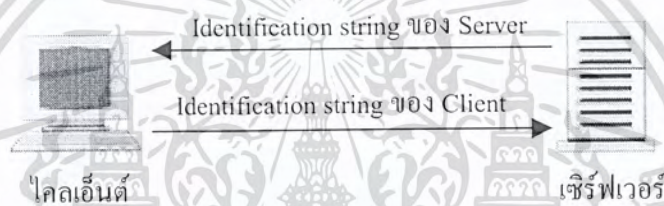
ตัวผู้ให้บริการจะรอดิคต่อนบน โพรโตคอลที่ซีพี/ไอพีที่พอร์ต 22 ตัวผู้ให้บริการจะใช้พอร์ตใดของตนก็ได้ในการติดต่อเข้ากับเครื่องผู้ให้บริการ แต่ถ้าตัวเครื่องให้บริการต้องการจะใช้การติดต่อแบบ .rhosts หรือ /etc/hosts.equiv จะต้องทำการติดต่อจากพอร์ตที่ถูกสงวนไว้ (หมายเลขพอร์ตที่ต่ำกว่าเบอร์ 1024)

5.3 รายละเอียดขั้นตอนการติดต่อ

5.3.1 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 แต่ถ้ารองรับเครื่องผู้ให้บริการก็จะส่งข้อความที่เป็นข้อมูลของตัวเองลักษณะเดียวกับที่เครื่องผู้ให้บริการส่งมากลับไป เมื่อเครื่องผู้ให้บริการได้รับก็จะตีความหมายว่าเป็นเวอร์ชันของโพรโตคอลที่รองรับหรือไม่ ถ้าใช่ก็จะส่งข้อความแรกกลับไป ซึ่งข้อความจะอยู่ในรูปของ binary packet protocol



รูปที่ 5-5 แสดงการตรวจสอบเวอร์ชัน Identification

5.3.2 ช่วยการแลกเปลี่ยนคีย์ (Key Exchange)

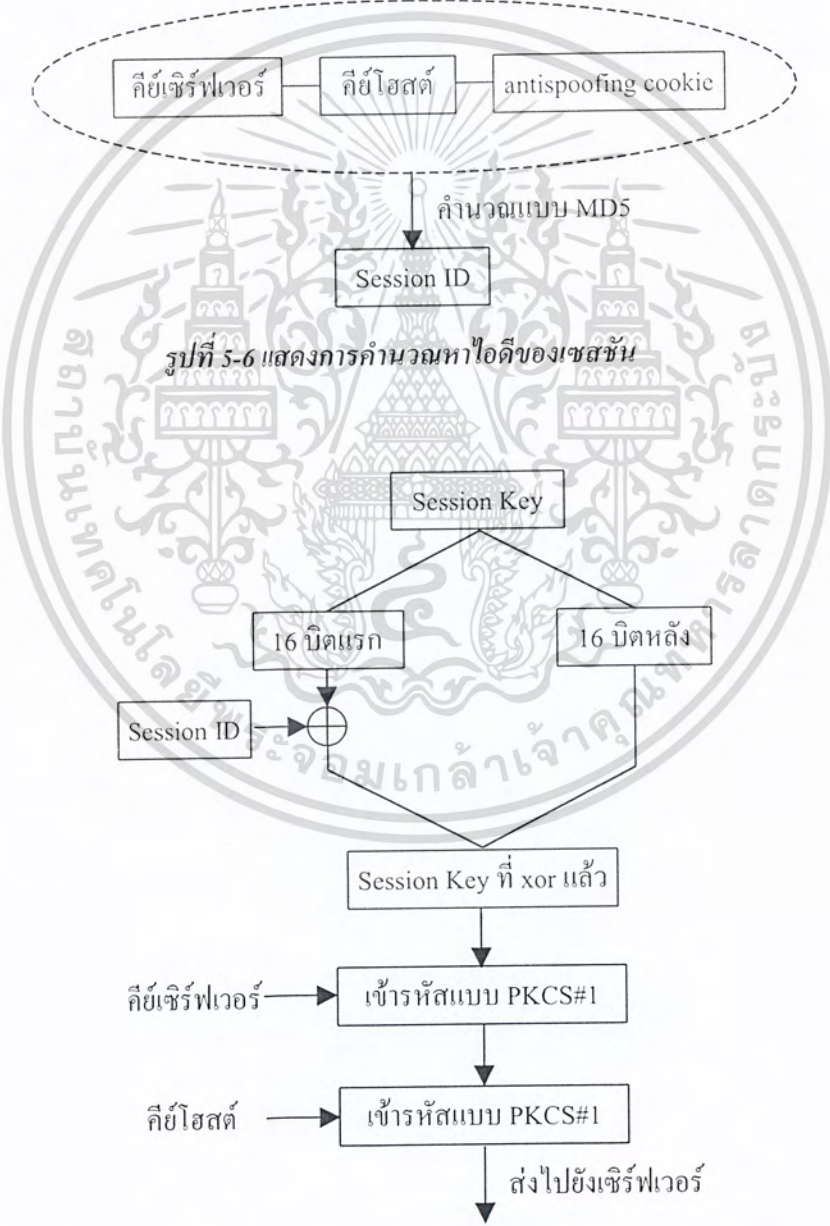
หลังจากที่ทำการตรวจสอบเวอร์ชันว่ารองรับกันได้แล้ว ก็จะเข้าสู่การแลกเปลี่ยนคีย์ เนื่องจากการเข้ารหัสของข้อมูลต่างๆ ที่ใช้ติดต่อกันทั้งสองฝ่ายจะต้องมีคีย์เดียวกันในการเข้ารหัสและถอดรหัสของข้อมูลนั้นๆ ซึ่งจะต้องมีวิธีการทำให้ทั้งคู่มีคีย์เดียวกัน โดยที่ผู้อื่นไม่ทราบวาคีย์นี้ คืออะไร ซึ่งมีขั้นตอน คือ

หลังจากที่เซิร์ฟเวอร์ได้รับ Identification String จากไคลเอ็นต์และตรวจสอบว่ารองรับแล้ว เซิร์ฟเวอร์จะส่งข้อความในรูปแบบ Binary Packet Protocol ที่เป็นชนิดของ SSH_MSG_PUBLIC_KEY ซึ่งจะประกอบด้วย

- ค่าที่สุ่มขึ้นมาขนาด 64 บิต (คูกัก) เพื่อให้ไคลเอ็นต์ส่งข้อมูลนี้กลับมา ทำให้ยากต่อการทำการปลอมแปลง IP
- โฮสต์คีย์ (host key) เป็นคีย์สาธารณะของเซิร์ฟเวอร์ มีขนาด 1024 บิต
- คีย์เซิร์ฟเวอร์ (server key) เป็นคีย์สาธารณะของเซิร์ฟเวอร์ที่มีการสร้างใหม่ทุกๆ ชั่วโมง มีขนาด 768 บิต
- การเข้ารหัสที่รองรับ (supported ciphers) เป็นชนิดของการเข้ารหัสที่เซิร์ฟเวอร์รองรับ
- การพิสูจน์สิทธิ์ที่รองรับ (supported authentication method) เป็นวิธีการพิสูจน์ตนที่เซิร์ฟเวอร์รองรับ

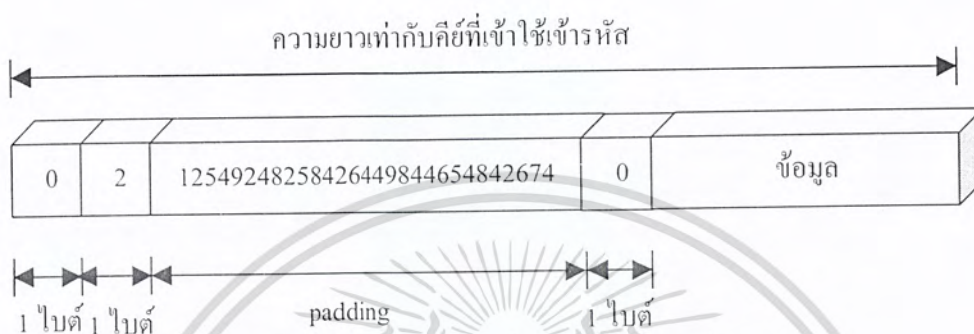
เมื่อไคลเอ็นต์ได้รับข้อความ SSH_MSG_PUBLIC_KEY ไคลเอ็นต์จะสร้างคีย์ (เรียกว่า คีย์เซสชัน) ที่ใช้สำหรับเข้ารหัสแพ็กเก็ตต่าง ๆ ขึ้นมาโดยการสุ่มค่าขึ้น (คีย์มีขนาด 32 ไบต์) เพื่อจะทำการส่งไปให้เซิร์ฟเวอร์ โดยก่อนที่จะทำการส่งจะนำเอาคีย์เซสชัน 16 บิตแรก มาเอ็กซ์คลูซีฟออริกกับไอดีของเซสชัน (มีขนาด 16 ไบต์) แล้วนำมาต่อรวมกับ 16 บิตหลังของคีย์เซสชัน จากนั้นจึงทำการเข้ารหัสแบบ PKCS#1 (ซึ่งเป็นการเข้ารหัสแบบ RSA) 2 ครั้ง ด้วยคีย์เซิร์ฟเวอร์และคีย์โฮสต์ โดยใช้กุญแจที่สั้นกว่าก่อน (นั่นคือคีย์เซิร์ฟเวอร์)

สำหรับไอดีของเซสชันเป็นค่าที่ทั้งสองฝั่งคำนวณหาได้ โดยการนำเอาคีย์โฮสต์ต่อด้วยคีย์เซิร์ฟเวอร์และต่อด้วยคูกี้ แล้วนำมาคำนวณด้วยอัลกอริทึมแบบ MD5 ซึ่งทั้งสองฝั่งจะได้ค่านี้เช่นเดียวกัน



รูปที่ 5-7 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 เอ็กซ์โปเนนทของคีย์สาธารณะ และ n คือ โมดูลัสของคีย์สาธารณะ)

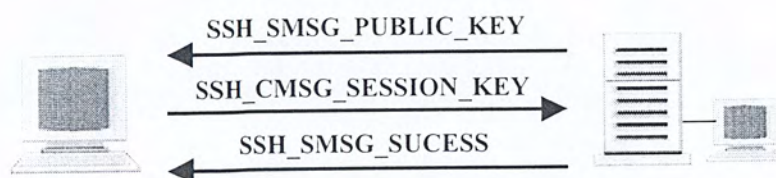


รูปที่ 5-8 แสดงตัวอย่างของข้อมูลที่จะทำการเข้ารหัส PKCS#1

หลังจากที่ได้ข้อมูลที่เป็นคีย์ (ที่ทำการเข้ารหัสข้างต้น) ก็จะส่งแพ็กเก็ตชนิด SSH_MSG_SESSION_KEY โดยมีข้อมูลต่างๆดังต่อไปนี้ ตามลำดับ

- ชนิดของการเข้ารหัส
- คู่คีย์ที่ได้จากเซิร์ฟเวอร์
- เซสชันคีย์ที่ถูกเข้ารหัสข้างต้น
- โพรโตคอลเวอร์ชันที่ใช้

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเก็ต SSH_MSG_SESSION_KEY ก็จะถอดรหัสและคำนวณหาเซสชันคีย์ออกมา หากลักษณะของข้อมูลถูกต้องก็จะส่งแพ็กเก็ตกลับไปด้วยเรียบร้อยแล้ว โดยมีชนิดว่า SSH_MSG_SUCCESS ซึ่งข้อมูลนี้จะถูกทำการเข้ารหัสด้วยวิธีการตามที่ไคลเอ็นต์เลือกมา



รูปที่ 5-9 ขั้นตอนการส่งของแพ็กเก็ตต่างๆ ในช่วงการแลกเปลี่ยนคีย์

5.4 ช่วงตรวจสอบชื่อและพิสูจน์ตน (Declare User Name and Authentication Phase)

ไคลเอ็นต์จะส่งแพ็กเก็ตชนิด SSH_CMSG_USER ไปยังเซิร์ฟเวอร์ ซึ่งจะมีชื่อของผู้ใช้ในแพ็กเก็ตนี้ เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบว่ามีผู้ใช้นั้นในรายการหรือไม่ และตรวจสอบการตรวจสอบสิทธิ์ที่ต้องใช้ แล้วจะตอบกลับด้วยแพ็กเก็ตชนิด SSH_SMSG_SUCCESS ถ้าผู้ใช้นี้ถูกกำหนดไว้ว่าไม่ต้องการให้มีการพิสูจน์ตน (ไม่ต้องใช้รหัสผ่าน) หรือ SSH_SMSG_FAILURE ถ้าผู้ใช้งานจำเป็นต้องมีการพิสูจน์ตน

สำหรับในกรณีที่ไม่มีชื่อของผู้ใช้นี้ เซิร์ฟเวอร์ก็จะส่ง SSH_SMSG_FAILURE มาเช่นกัน ทำให้ผู้ใช้ไม่ทราบว่าผู้ใช้นี้มีอยู่หรือเปล่า

เมื่อไคลเอ็นต์ได้รับแพ็กเก็ต SSH_SMSG_FAILURE ไคลเอ็นต์จะทำการส่งวิธีการในการพิสูจน์ตนให้กับเซิร์ฟเวอร์ ซึ่งวิธีการพิสูจน์ตนที่รองรับ มีวิธีต่าง ๆ ดังนี้

- 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_SUCCESS แต่หากไม่รองรับก็จะส่งแพ็กเก็ตชนิด SSH_SMSG_FAILURE

เวลาสำหรับการพิสูจน์ตนจะถูกกำหนดไว้ไม่เกิน 5 นาที (ถ้าหากเกิน เซิร์ฟเวอร์ก็จะทำการยกเลิกการติดต่อ) สำหรับวิธีการตรวจสอบสิทธิ์วิธีต่าง ๆ มีรายละเอียดดังนี้

5.4.1 ตรวจสอบสิทธิ์จากไฟล์ `.rhosts` หรือ `/etc/hosts.equiv`

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

การวิธีการพิสูจน์ตนด้วยระบบที่เชื่อถือฝั่งรีโมต (trusts the remote host) สามารถที่จะถูกอ่านของข้อมูลได้โดยวิธีการปลอมแปลงเลขไอพี แต่จะถูกป้องกันไว้จากโปรโตคอล (เพราะทุกๆ แพ็กเก็ตจะถูกเข้ารหัสอยู่)

5.4.2 ตรวจสอบสิทธิ์ด้วยวิธี RSA

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

ถ้าเซิร์ฟเวอร์ไม่พบคีย์การพิสูจน์ตนของไคลเอ็นต์ในตัวมัน (ค่าคีย์สาธารณะทั้งหมดของไคลเอ็นต์) เซิร์ฟเวอร์จะตอบกลับทันทีด้วย SSH_MSG_FAILURE ในทางตรงกันข้าม ถ้าเซิร์ฟเวอร์พบ มันจะสร้าง challenge ซึ่งทำการเข้ารหัสด้วยคีย์สาธารณะของไคลเอ็นต์

เมื่อไคลเอ็นต์ได้รับ จะทำการถอดรหัสโดยคีย์ส่วนตัว แล้วนำมาต่อกับไอดีของเซสชันที่มีอยู่ ทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมาขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบเทียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge + เซสชันไอดี => ทำ MD5) ถ้าถูกต้องก็จะส่ง SSH_MSG_SUCCESS แต่ในทางตรงกันข้ามก็จะส่ง SSH_MSG_FAILURE และปฏิเสธการพิสูจน์ตนนี้

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

เป็นวิธีที่เพิ่มจากวิธีการตรวจสอบสิทธิ์จากไฟล์ .rhosts หรือ /etc/hosts.equiv โดยมีการพิสูจน์ตนแบบ RSA ด้วย โดยไคลเอ็นต์จะส่ง SSH_MSG_AUTH_RHOSTS_RSA ตามด้วยชื่อของผู้ใช้และกุญแจสาธารณะของไคลเอ็นต์

โดยเริ่มแรก เซิร์ฟเวอร์จะทำการตรวจสอบตามปกติเช่นเดียวกับตรวจสอบจากไฟล์ .rhosts และ /etc/hosts.equiv ถ้าไม่ถูกต้องก็จะส่ง SSH_MSG_FAILURE กลับมา ตรงกันข้ามมันจะตรวจสอบกุญแจสาธารณะในตัว ถ้าหากไม่พบก็จะส่ง SSH_MSG_FAILURE เพื่อยกเลิกการติดต่อ

ถ้าเซิร์ฟเวอร์รู้โฮสต์คีย์ของเครื่องไคลเอ็นต์ มันจะทำการตรวจสอบจากโฮสต์คีย์ที่ได้มาว่าตรงกันหรือไม่ ถ้าไม่ ก็จะส่ง SSH_MSG_FAILURE เพื่อยกเลิกการติดต่อเช่นกัน

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

ไคลเอ็นต์เมื่อได้รับจะทำการถอดรหัสโดยกุญแจส่วนตัว แล้วนำมาต่อกับเซสชันไอดีที่มีอยู่ มาทำการคำนวณแบบ MD5 ได้ผลลัพธ์ออกมาขนาด 16 ไบต์ แล้วส่งกลับไปในแพ็กเก็ตของ SSH_AUTH_RSA_RESPONSE

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบโดยเปรียบเทียบค่าที่ได้รับมา และค่าที่ตัวมี (challenge + เซสชันไอดี => ทำ MD5) ถ้าถูกต้องก็จะส่ง SSH_MSG_SUCCESS ตรงกันข้ามก็จะส่ง SSH_MSG_FAILURE และปฏิเสธการพิสูจน์ตนนี้

5.4.4 ตรวจสอบสิทธิ์จากรหัสผ่าน

โดยไคลเอ็นต์จะส่ง SSH_MSG_AUTH_PASSWORD ตามด้วยข้อความที่เป็นรหัสผ่าน (ข้อความนี้จะยังไม่มีการทำอะไร แต่จะถูกทำการเข้ารหัสด้วยกลไกของแพ็กเก็ตโดยอัตโนมัติ)

เมื่อเซิร์ฟเวอร์ได้รับจะทำการตรวจสอบรหัสผ่าน ถ้าการพิสูจน์ตนถูกต้อง เซิร์ฟเวอร์จะส่ง SSH_MSG_SUCCESS แต่ถ้าผิดพลาดเซิร์ฟเวอร์จะส่ง SSH_MSG_FAILURE

5.5 ช่วงการเตรียมการ (Preparatory Operation)

หลังจากที่ทำการพิสูจน์ตนแล้ว เซิร์ฟเวอร์จะรอคำร้องขอจากไคลเอ็นต์และทำการจัดการกับคำสั่งที่เข้ามา เซิร์ฟเวอร์จะตอบกลับด้วย SSH_MSG_SUCCESS เมื่อคำร้องขอที่เข้าถูกจัดการเสร็จสิ้น แต่ในกรณีตรงกันข้าม เมื่อคำร้องขอที่เข้ามาไม่สามารถจัดการได้หรือจัดการไม่สำเร็จก็จะส่ง SSH_MSG_FAILURE ซึ่งข้อความช่วงนี้เป็นการเตรียมการสำหรับในช่วงต่อไป

5.6 เซสชันแบบอินเตอร์แอ็กทีฟและการแลกเปลี่ยนข้อมูล (Interactive Session and Exchange of Data)

ในเซสชันแบบอินเตอร์แอ็กทีฟ ข้อมูลทุกตัวจะถูกเขียนโดยเซสล์หรือคำสั่งที่ทำงานอยู่บนเครื่องเซิร์ฟเวอร์เพื่อส่งออกไปยังส่วนแสดงผลหรือส่วนแสดงข้อผิดพลาดบนเครื่องไคลเอ็นต์ ส่วนอินพุตจะมาจากส่วนรับข้อมูลบนเครื่องไคลเอ็นต์ ซึ่งจะถูกส่งไปยังโปรแกรมบนเครื่องเซิร์ฟเวอร์

การแลกเปลี่ยนข้อมูลทั้งหมดจะเป็นอะซิงโครนัส (asynchronous) โดยการส่งสามารถเกิดขึ้นได้ตลอดเวลาและไม่ต้องมีการตอบสนอง (ปกติแล้ว ที่ซีพี/ไอพีจัดการสร้างความน่าเชื่อถือได้อยู่แล้ว และโปรโตคอลแพ็กเก็ตก็จะทำการป้องกันการเข้ามายุ่งหรือทำการปลอมแปลงไอพี)

เมื่อไคลเอ็นต์ได้รับ EOF จากส่วนรับข้อมูล มันจะส่ง SSH_MSG_EOF การแลกเปลี่ยนข้อมูลและโหมคอินเตอร์แอ็กทีฟจะสิ้นสุดลงเมื่อเซิร์ฟเวอร์ส่ง SSH_MSG_EXITSTATUS เพื่อที่จะแสดงว่าการติดต่อกับตัวไคลเอ็นต์ได้สิ้นสุดลง ส่วนไคลเอ็นต์หรือเซิร์ฟเวอร์จะสามารถยกเลิกการติดต่อเมื่อใดก็ได้ โดยการส่งข้อความ SSH_MSG_DISCONNECT หรือทำการปิดการเชื่อมต่อ (close the connection)

5.7 การยกเลิกการติดต่อ (Termination of the Connection)

โดยปกติการยกเลิกการติดต่อจะเริ่มโดยเซิร์ฟเวอร์ซึ่งจะทำการส่ง SSH_MSG_EXITSTATUS หลังจากโปรแกรมจบลง ตัวไคลเอ็นต์จะทำการตอบสนองกับข้อความดังกล่าว โดยการส่ง SSH_MSG_EXIT_CONFIRMATION และทำการปิดช่องเกิดของตัวเองลง ส่วนเซิร์ฟเวอร์เมื่อได้รับข้อความจึงค่อยทำการปิดช่องเกิดลง เป้าหมายสำคัญสำหรับการยืนยันในการยกเลิกคือ ในบางระบบอาจจะสูญเสียข้อมูลที่ส่งไปก่อนหน้านี้เมื่อช่องเกิดถูกปิด และการยกเลิกในฝั่งไคลเอ็นต์ก่อนทำให้เกิด TCP/IP TIME_WAIT [RFC 0793] ซึ่งทำให้เซิร์ฟเวอร์รอรับคำตอบจากฝั่งไคลเอ็นต์ที่ไม่ได้ใช้ ทำให้เซิร์ฟเวอร์ต้องสูญเสียรีซอร์ส

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

5.8 การใช้งานการฟอร์เวิร์ดพอร์ตของโปรโตคอลเอสเอสเอสเอช 1

การฟอร์เวิร์ดพอร์ตความสามารถที่เป็นประโยชน์มากอีกอย่างของโปรโตคอลเอสเอสเอสเอช 1 ทำให้เราสามารถใช้งานโปรแกรมประยุกต์อื่นๆ ที่ไม่มีความปลอดภัย คือไม่มีการเข้ารหัสข้อมูล เป็นการสื่อสารที่มีความปลอดภัยด้วยการเข้ารหัสของโปรโตคอลเอสเอสเอสเอช 1 แบ่งออกเป็น 2 ชนิดคือ

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

5.8.1 การฟอร์เวิร์ดเอ็กซ์ (X Forwarding)

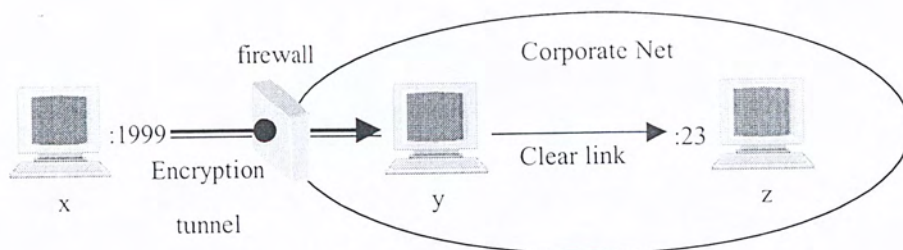
สำหรับในระบบยูนิกซ์นั้น ผู้ใช้สามารถใช้งานการติดต่อแบบกราฟิกได้โดยการเรียกใช้เอ็กซ์วินโดว์ (X window) ซึ่งการเรียกใช้เอ็กซ์วินโดว์จะใช้โปรโตคอลเอ็กซ์ 11 โปรโตคอลเอสเอสเอสเอช 1 นั้นสามารถช่วยเข้ารหัสข้อมูลที่ถูกรับส่งโดยโปรโตคอลเอ็กซ์ 11 นี้ได้โดยใช้หลักการเดียวกับการฟอร์เวิร์ดพอร์ตทีซีพี/ไอพีซึ่งจะอธิบายต่อไป

5.8.2 การฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี (TCP/IP Port Forwarding)

ปกติแล้วของโปรโตคอลที่ทำงานอยู่บนโปรโตคอลทีซีพี/ไอพี เช่น เอชทีทีพี, เอฟทีพี, เทลเน็ต, เอสเอ็มทีพี นั้นจะถูกแบ่งแยกโดยหมายเลขพอร์ต โปรโตคอลเอสเอสเอสเอช 1 ก็เช่นกัน โดยโปรโตคอลเอสเอสเอสเอช 1 ทำงานที่พอร์ตหมายเลข 22 เมื่อเราเรียกใช้งานโปรโตคอลเอสเอสเอสเอช 1 แล้ว การส่งข้อมูลระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะถูกเข้ารหัสเฉพาะข้อมูลของโปรโตคอลเอสเอสเอสเอช 1 เท่านั้น โดยข้อมูลของโปรโตคอลอื่นๆ นั้นไม่มีการเข้ารหัสเลย โปรโตคอลเอสเอสเอสเอช 1 สามารถช่วยให้โปรโตคอลต่างๆ ที่ทำงานอยู่บนโปรโตคอลทีซีพี/ไอพีมีการเข้ารหัสข้อมูลได้โดยการเรียกใช้บริการฟอร์เวิร์ดพอร์ตทีซีพี/ไอพี ซึ่งแบ่งออกได้เป็น 2 ลักษณะ คือ การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ หรือ มีการสร้างพอร์ตที่ฝั่งของเซิร์ฟเวอร์โปรโตคอลเอสเอสเอสเอช 1

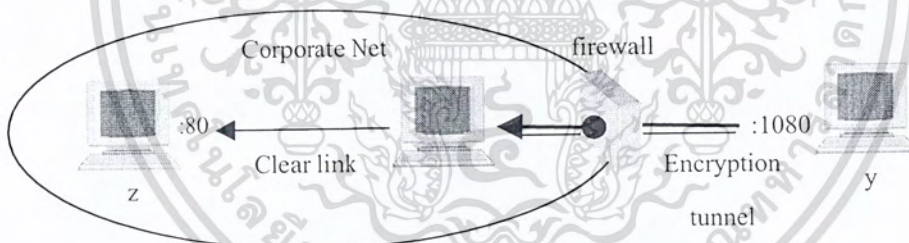
- การสร้างพอร์ตที่ฝั่งของไคลเอ็นต์ (Local) สามารถอธิบายง่ายๆ ได้ด้วยตัวอย่างดังรูปที่ 5-10 เป็นการฟอร์เวิร์ดพอร์ตจากโฮสต์ฝั่งไคลเอ็นต์ไปยังเครื่องรีโมต คือ พิจารณาโฮสต์ 3 แห่ง คือ x y และ z โฮสต์ x นั้นเป็นเครื่องไคลเอ็นต์และสร้างการเชื่อมต่อกับโฮสต์ y ซึ่งเป็นเครื่องเซิร์ฟเวอร์ผ่านโปรโตคอลเอสเอสเอสเอช 1 พอร์ตที่ 1999 บนเครื่อง x ถูกสร้างส่งต่อไปยังเครื่องอีกเครื่องนั่นคือ โฮสต์ z ที่พอร์ต 23 พอร์ต 1999 นี้จะส่งผ่านช่องทางที่มีการเข้ารหัสไปยังเครื่อง y ซึ่งจะส่งต่อไปที่เครื่อง z ที่พอร์ต 23 โดยที่ไม่มีการเข้ารหัสในการเชื่อมต่อส่วนนี้

ด้วยเหตุนี้เอง ถ้าใช้คำสั่ง telnet localhost 1999 จะมีผลเหมือนกับการเชื่อมต่อเพื่อสร้างการติดต่อกับ z โดยผ่านช่องทางที่มีความปลอดภัยของเครื่อง y



รูปที่ 5-10 แสดงการทำงานของพอร์ตฟอร์เวิร์ดแบบโลคอล

- การสร้างพอร์ตที่ฝั่งเซิร์ฟเวอร์โปรโตคอลเอสเอสเอช 1 (remote) เป็นการพอร์ตฟอร์เวิร์ดของโฮสต์รีโมตไปยังเครื่องโลคอล คือ จะพอร์ตฟอร์เวิร์ดของเครื่องรีโมต ซึ่งคือ y พอร์ตที่ 1080 ไปยังโฮสต์ z พอร์ต 80 บนช่องทางที่มีการเข้ารหัสของ x ดังรูปที่ 5-11 ซึ่งสามารถถูกมองเป็น pseudo proxy capability หรือ tunneling IP และมีผลกระทบในแง่ของจุดบกพร่องทางด้านความปลอดภัยในไฟล์วอลล์ ซึ่งจะอนุญาตให้โปรโตคอลเอสเอสเอช 1 ผ่านไปได้ ดังนั้นไฟล์วอลล์หรืออุปกรณ์ที่ทำหน้าที่ใกล้เคียงกัน จะคิดว่ามันกำลังอนุญาตให้การติดต่อของโปรโตคอลเอสเอสเอช 1 ผ่านเข้าไป แต่จะมีผลที่จริงแล้วเหมือนกับการอนุญาตให้มีการติดต่อกันอย่างอิสระระหว่างระบบภายในกับระบบภายนอก



รูปที่ 5-11 แสดงการทำงานของพอร์ตฟอร์เวิร์ดแบบรีโมต

บทที่ 6

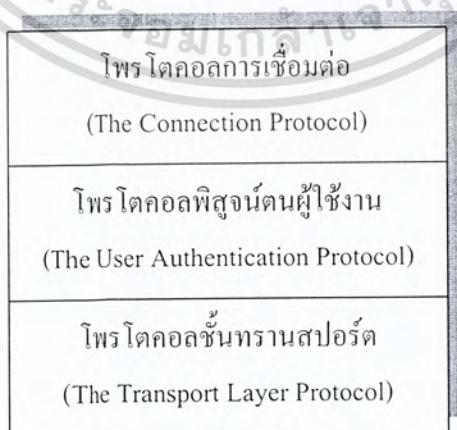
โพรโทคอลเอสเอสเอช 2

6.1 แนะนำ

โพรโทคอลเอสเอสเอช 2 เป็นโพรโทคอลสำหรับการล็อกอินระยะไกลแบบปลอดภัย และการใช้บริการอื่นๆผ่านเครือข่ายแบบปลอดภัย บนเครือข่ายที่ไม่ปลอดภัย ประกอบด้วยสามคอมโพเนนต์หลักคือ

- โพรโทคอลชั้นทรานสปอร์ต (The Transport Layer Protocol) ทำให้มีการพิสูจน์เซิร์ฟเวอร์ (server authentication), การสร้างความมั่นใจในความปลอดภัย (confidentiality) และความสมบูรณ์ (integrity) และสามารถเลือกให้มีการบีบอัด (compression) ได้ด้วย โดยปกติชั้น ทรานสปอร์ตจะทำงานบนการเชื่อมต่อแบบทีซีพี/ไอพี (TCP/IP Connection) แต่ก็อาจถูกนำไปใช้งานอยู่บนสตรีมข้อมูลที่ไว้วางใจได้ (reliable data stream) อื่นๆอีกด้วย
- โพรโทคอลพิสูจน์ตนผู้ใช้ (The User Authentication Protocol) ทำหน้าที่พิสูจน์ตนของผู้ใช้งานฝั่งไคลเอนต์ (client-side) ให้แก่เซิร์ฟเวอร์ โพรโทคอลนี้ทำงานบนโพรโทคอลชั้นทรานสปอร์ต
- โพรโทคอลการเชื่อมต่อ (The Connection Protocol) ทำหน้าที่มัลติเพล็กซ์ (multiplex) ทันเนลที่เข้ารหัส (encrypted tunnel) เป็นหลายๆแชนเนล (logical channel) โพรโทคอลนี้จะทำงานอยู่บนโพรโทคอลพิสูจน์ตนผู้ใช้งาน

โดยการจัดวางตัวของชั้นโพรโทคอลหลักของโพรโทคอลเอสเอสเอช 2 เป็นดังรูปที่ 6-1



รูปที่ 6-1 การจัดวางตัวของโพรโทคอลหลักภายในโพรโทคอลเอสเอสเอช 2

ไคลเอ็นต์จะส่งคำร้องขอบริการ (service request) เมื่อการเชื่อมต่อชั้นทรานสปอร์ตที่ปลอดภัยได้ถูกก่อตั้งขึ้นแล้ว คำร้องขอบริการครั้งที่สองจะถูกส่งหลังจากการพิสูจน์ตนของผู้ใช้ได้ทำสำเร็จลง

โพรโตคอลการเชื่อมต่อให้แชนเนลที่สามารถนำไปใช้ได้หลากหลายจุดประสงค์ วิธีการมาตรฐานถูกใช้ในการตั้งค่าเซสชันของเซลล์แบบอินเทอร์แอคทีฟที่ปลอดภัย และสำหรับการฟอร์เวิร์ด (forwarding) หรือการทันเนล พอร์ตทีซีพี/ไอพีและการเชื่อมต่อ X11

6.2 สถาปัตยกรรม

6.2.1 คีย์โฮสต์ (Host Keys)

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

คีย์โฮสต์ของเซิร์ฟเวอร์ถูกใช้ในช่วงการแลกเปลี่ยนคีย์ เพื่อใช้ในการตรวจสอบว่าไคลเอ็นต์ได้ติดต่อกับเซิร์ฟเวอร์ที่ถูกต้องจริง เพื่อให้ใช้วิธีการนี้ได้ ไคลเอ็นต์ต้องรู้คีย์โฮสต์สาธารณะของเซิร์ฟเวอร์ก่อน

มีโมเดลการไว้วางใจ (trust modes) 2 โมเดลที่แตกต่างกันซึ่งสามารถนำมาใช้

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

ทางเลือกที่สองช่วยบรรเทาปัญหาการบำรุงรักษา เนื่องจากตามอุดมคติ คีย์ของ CA คีย์เพียงเดียวที่จำเป็นต้องถูกเก็บอย่างปลอดภัยลงในไคลเอ็นต์ แต่คีย์โฮสต์แต่ละคีย์ต้องถูกรับรองโดย CA ก่อนที่จะสามารถทำการพิสูจน์ตนได้

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

การอิมพลีเมนต์อาจให้วิธีการเพิ่มเติมสำหรับตรวจสอบความถูกต้องของคีย์โฮสต์ เช่น ฟังก์ชันแฮชแบบเลขฐานสิบหก ที่ได้อัลกอริทึมแฮช SHA-1 ของคีย์สาธารณะ ฟังก์ชันแฮชที่ถูกรวบรวมได้ง่ายโดยใช้โทรศัพท์หรือเซนเซอร์สื่อสารภายนอก

ทุกการอิมพลีเมนต์ควรให้ทางเลือกที่ไม่ทำการยอมรับคีย์โฮสต์ที่ไม่สามารถตรวจสอบได้

6.2.2 ความสามารถในการขยาย (Extensibility)

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

เราเลือกที่จะแบ่งชื่ออัลกอริทึม, วิธีการ, รูปแบบ และโพรโตคอลขยายเพิ่ม ด้วยชื่อในรูปแบบข้อความซึ่งอยู่ในรูปแบบที่กำหนดไว้ ชื่อ DNS ถูกใช้ในการตั้งโลคอลเนมสเปซ (local namespace) ทำให้การทดลองหรือการขยายที่เป็นหมวดหมู่สามารถกำหนดได้โดยไม่ต้องกลัวว่าจะเกิดความขัดแย้งกับการอิมพลีเมนต์อื่น

เป้าหมายหนึ่งของการออกแบบคือพยายามให้โพรโตคอลพื้นฐานง่ายที่สุดเท่าที่จะเป็นไปได้ และต้องการอัลกอริทึมน้อยที่สุดเท่าที่จะเป็นไปได้ อย่างไรก็ตาม ทุกการอิมพลีเมนต์ต้องรองรับกลุ่มอัลกอริทึมเล็กที่สุดที่ทำให้มีความสามารถทำงานร่วมกันระหว่างระบบได้อย่างมั่นใจ (ไม่ได้บอกว่ายโยบาย (policy) โคลดบนทุกโฮสต์จำเป็นต้องใช้อัลกอริทึมเหล่านี้) อัลกอริทึมขาดไม่ได้ (mandatory) ถูกกำหนดในเอกสารของโพรโตคอล

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

6.2.3 การกำหนดนโยบาย (Policy Issues)

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

การกำหนดนโยบายควรกำหนดลงในกลไกการปรับแต่งค่าของแต่ละการอิมพลีเมนต์ดังต่อไปนี้

- อัลกอริทึมการเข้ารหัส, อัลกอริทึมการตรวจสอบความสมบูรณ์ และอัลกอริทึมการบีบอัด จะแยกกันสำหรับแต่ละทิศทาง นโยบายต้องกำหนดอัลกอริทึมเสนอ (preferred algorithm) เช่น ลิสต์อัลกอริทึมแรกในแต่ละหมวด
- อัลกอริทึมคีย์สาธารณะ และวิธีการแลกเปลี่ยนคีย์ ถูกใช้ในการพิสูจน์โฮสต์ การมีอยู่ของคีย์โฮสต์ที่ไว้วางใจได้สำหรับอัลกอริทึมคีย์สาธารณะที่แตกต่างกัน จะมีผลต่อตัวเลือกนี้

- วิธีการพิสูจน์ตนที่เซิร์ฟเวอร์ต้องการสำหรับแต่ละผู้ใช้ นโยบายของเซิร์ฟเวอร์อาจต้องการพิสูจน์ตนหลายครั้ง สำหรับบางผู้ใช้หรือกับทุกผู้ใช้ อัลกอริทึมที่ต้องการขึ้นอยู่กับที่ ๆ ผู้ใช้กำลังพยายาม ล็อกอินมาจากที่นั้น
- การดำเนินการที่ผู้ใช้ได้รับอนุญาตให้กระทำโดยใช้โพรโตคอลการเชื่อมต่อ บางประเด็นมีความเกี่ยวข้องกับความปลอดภัย ตัวอย่างเช่น นโยบายไม่ควรอนุญาตให้เซิร์ฟเวอร์สแตตัสเซชันหรือรันคอมมานด์บนเครื่องไคลเอนต์ และต้องไม่อนุญาตให้มีการเชื่อมต่อไปยังเอเจนต์พิสูจน์ตน (authentication agent) เว้นแต่การฟอร์เวิร์ดที่การเชื่อมต่อถูกร้องขอมา ประเด็นอื่น ๆ เช่น พอร์ตที่ซีพี/ไอพีใด ๆ ที่สามารถถูกฟอร์เวิร์ด และโดยใครได้บ้าง เป็นประเด็นของนโยบายบนโพลอด หลายประเด็นนี้เกี่ยวข้องกับการทราเวอร์ส (traversing) และการบายพาส (bypassing) ผ่านไฟร์วอลล์ (firewalls) และมีความเกี่ยวข้องกันกับนโยบายความปลอดภัยบนโพลอด

6.2.4 คุณสมบัติความปลอดภัย (Security Properties)

เป้าหมายพื้นฐานของโพรโตคอลเอสเอสเอสเอช 2 คือความปลอดภัยบนอินเทอร์เน็ตที่ได้รับการปรับปรุงต่อไปนี้เป็นแนวทางเพื่อให้ง่ายต่อการใช้งาน

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

การยินยอมอ่อนข้อถูกทำเพื่อสร้างความง่ายเพื่อให้การใช้งานแพร่หลาย กรณีพิเศษที่เกิดขึ้นคือการตรวจสอบว่าคีย์โฮสต์ของเซิร์ฟเวอร์ เป็นของโฮสต์ที่ต้องการจริง โพรโตคอลอนุญาตให้การตรวจสอบถูกละเว้นได้ (แต่ไม่แนะนำ) เชื่อว่าจะปรับปรุงให้การใช้งานในระยะสั้นๆ สะดวกขึ้นอย่างเห็นได้ชัดเจน จนกว่าจะมีโครงสร้างคีย์สาธารณะสำหรับอินเทอร์เน็ตที่กว้างขวางเกิดขึ้น

6.2.5 ขนาดแพ็กเก็ตและโอเวอร์เฮด (Packet Size and Overhead)

ผู้อ่านอาจกังวลเกี่ยวกับการเพิ่มขึ้นของขนาดแพ็กเก็ตเพื่อเฮดเดอร์ใหม่ แพดดิ้ง (padding) ใหม่ และ MAC ใหม่ ขนาดแพ็กเก็ตที่เล็กที่สุดอยู่ในลำดับของ 28 ไบต์ (ขึ้นอยู่กับอัลกอริทึมที่เจรจา) การเพิ่มขึ้นเป็นเรื่องเล็กน้อยสำหรับแพ็กเก็ตขนาดใหญ่ แต่จะเป็นเรื่องสำคัญสำหรับแพ็กเก็ตขนาดเล็ก (เช่น 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 สำหรับเซชันอินเตอร์แอคทีฟแล้ว เราไม่ต้องการแพ็กเก็ตที่มีขนาดใหญ่เกินความจำเป็น เพื่อลดความเสี่ยงในการอัปเดตหน้าจอให้น้อยที่สุด ขนาดแพ็กเก็ตใหญ่ที่สุดจะถูกเจรจาแยกกันสำหรับแต่ละเซชัน

6.3 การแทนชนิดข้อมูล (Data Type Representation) ที่ใช้ในโปรโตคอลเอสเอสเอช 2

- **byte** *byte* ใช้แทนค่า 8 บิตใด ๆ (octet) [RFC-1700] บางครั้งข้อมูลที่มีความยาวคงที่ถูกแทนด้วยอาร์เรย์ของ *byte* เขียนด้วย *byte[n]* โดยที่ *n* เป็นจำนวนของ *byte* ในอาร์เรย์
- **boolean** ค่า *boolean* ถูกเก็บด้วยไบนารีเดียว 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 (เรียงแบบไบต์เครื่องช่วย) ตัวอย่างเช่น ค่า 699921578 (0x29b7f4aa) ถูกเก็บเรียง 29 b7 f4 aa
- **uint64** ใช้แทนค่าจำนวนเต็มที่ไม่เป็นลบขนาด 64 บิต เก็บลงแบบข้อมูลชนิด *byte* 8 ไบต์ โดยมีลำดับนัยสำคัญเรียงจากมากไปน้อยเช่นเดียวกับ *uint32*
- **string** ใช้แทนสตริงไบนารีความยาวใดๆ สตริงสามารถบรรจุข้อมูลไบนารีใด ๆ ก็ได้ รวมถึงตัวอักษร null และตัวอักษร 8 บิต มันถูกเก็บแบบ *uint32* ที่บรรจุความยาวของมันเอง (จำนวนไบต์ที่ตามมา) และ 0 (เป็นสตริงว่าง) หรือหลายไบต์ที่เก็บค่าของสตริง และไม่ใช่ตัวอักษร 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) ในรูปแบบทวูคอมพลิเมนต์ (two’s complement) ถูกจัดเก็บแบบสตริง เริ่มจากไบต์ที่มีนัยสำคัญมากก่อน เลขที่ติดลบจะมีค่าที่บิตนัยสำคัญสูงสุดของไบต์แรกของส่วนข้อมูลเป็น 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 6.1 แสดงตัวอย่างการแทนค่าด้วย mpint

- **name-list** เป็นสตริงที่บรรจุลิสต์ของชื่อที่ใช้อักขระจุลภาค (,) แบ่งคั่นภายใน ลิสต์ชื่อจะถูกแทนด้วย uint32 ที่บรรจุความยาวของมัน (จำนวนไบต์ที่ตามมา) ตามด้วยลิสต์ของชื่อศูนย์ชื่อหรือมากกว่าที่ใช้อักขระจุลภาคแบ่งคั่นภายใน ชื่อจะต้องมีความยาวไม่เป็นศูนย์และต้องไม่ประกอบด้วยอักขระจุลภาค และอาจมีข้อกำหนดเพิ่มเติมกับชื่อ เช่น จะต้องเป็นไอเดนติไฟเออร์อัลกอริทึมที่ถูกต้อง (ดูในหัวข้อการตั้งชื่ออัลกอริทึม) หรือเป็นแท็กภาษา [RFC-1766] ลำดับของชื่อในลิสต์อาจเป็นหรือไม่เป็นนัยสำคัญก็ได้ ขึ้นอยู่กับคอนเท็กซ์ที่ลิสต์นั้นใช้ 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 6.2 แสดงตัวอย่างการแทนค่าด้วย name-list

6.4 การตั้งชื่ออัลกอริทึม (Algorithm Naming)

โพรโทคอลเอสเอสเอชอ้างอิงอัลกอริทึมแฮช อัลกอริทึมการเข้ารหัส อัลกอริทึมตรวจสอบความสมบูรณ์ อัลกอริทึมการบีบอัด อัลกอริทึมการแลกเปลี่ยนคีย์ หรือโพรโทคอลด้วยชื่อ มีบางอัลกอริทึมมาตรฐานที่ทุกการอิมพลีเมนต์ต้องรองรับ มีบางอัลกอริทึมที่ถูกนิยามไว้ในการกำหนดลักษณะเฉพาะของโพรโทคอลแต่เป็นทางเลือก (optional) ยิ่งไปกว่านี้ บางองค์กรต้องการที่จะใช้อัลกอริทึมของตน

ในโพรโทคอลนี้ ทุกไอดีเนติฟายเออร์สำหรับอัลกอริทึมต้องเป็นสตริง US-ASCII ที่สามารถพิมพ์เห็นได้โดยไม่เป็นสตริงว่าง และไม่ยาวเกิน 64 อักขระ ชื่อต้องเป็นแบบมีความแตกต่างในอักขรเล็ก-ใหญ่

ชื่ออัลกอริทึมมี 2 รูปแบบ

- ชื่อที่ไม่ประกอบด้วยอักขระ @ ถูกสงวนไว้เพื่อกำหนดโดยมติของ 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] และถูกควบคุมโดยบุคคลหรือองค์กรที่นิยามชื่อแล้วแต่ละโดเมนว่าจะจัดการโดเมนเนมสเปซของตนอย่างไร

6.5 หมายเลขเมสเสจ (Message Number)

แพ็กเก็ตเอสเอสเอชมีหมายเลขเมสเสจอยู่ในช่วง 1 ถึง 255 หมายเลขถูกกำหนดไว้ดังนี้

โพรโทคอลชั้นทรานสปอร์ต :

- 1 ถึง 19 ส่วนหลัก (generic) ของโพรโทคอลชั้นทรานสปอร์ต (เช่น disconnect, ignore, debug เป็นต้น)
- 20 ถึง 29 การเจรจาอัลกอริทึม
- 30 ถึง 49 ระเบียบวิธีการแลกเปลี่ยนคีย์ (หมายเลขสามารถถูกนำมาใช้ใหม่สำหรับวิธีการพิสูจน์ตนที่ต่างกัน)

โพรโทคอลการพิสูจน์ตนผู้ใช้ :

- 50 ถึง 59 ส่วนหลัก (generic) ของโพรโทคอลพิสูจน์ตนผู้ใช้
- 60 ถึง 79 ระเบียบวิธีการพิสูจน์ตนผู้ใช้ (หมายเลขสามารถถูกนำมาใช้ใหม่สำหรับวิธีการพิสูจน์ตนที่ต่างกัน)

โพรโตคอลการเชื่อมต่อ :

80 ถึง 89 ส่วนหลัก (generic) ของโพรโตคอลการเชื่อมต่อ

90 ถึง 127 เมสเสจที่เกี่ยวข้องกับแชนเนล (Channel related messages)

สงวนไว้สำหรับโพรโตคอลบนไคลเอ็นต์ :

128 ถึง 191 สงวนไว้

ส่วนขยายบนโพล :

192 ถึง 255 ส่วนขยายบนโพล (Local extensions)

6.6 การพิจารณาของ IANA

การจัดสรรชื่อของชนิดต่อไปนี้ในโพรโตคอลเอสเอสเอช ถูกกำหนดโดยมติของ IETF

- ชื่ออัลกอริทึมการเข้ารหัส
- ชื่ออัลกอริทึม MAC
- ชื่ออัลกอริทึมสาธารณะ (อัลกอริทึมคีย์สาธารณะซึ่งขึ้นอยู่กับความสามารถของการเข้ารูปแบบและความสามารถในการลงลายเซ็น/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

หมายเลขเมสเสจในช่วง 0 ถึง 191 ควรถูกจัดสรรผ่านมติของ IETF หมายเลขเมสเสจในช่วง 192 ถึง 255 ชุดส่วนขยายบนโพลถูกสงวนไว้ใช้ส่วนตัว

6.7 พิจารณาความปลอดภัย

ควรใส่ใจเป็นพิเศษ เพื่อให้มั่นใจว่าหมายเลขรุ่นทั้งหมดมีคุณภาพดี หมายเลขรุ่นควรถูกสร้างด้วยกลไกที่ปลอดภัย อธิบายไว้ใน [RFC-1750]

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

ควรหลีกเลี่ยงการไม่ใช้ MAC หรือการเข้ารหัส ถ้าการเข้ารหัสถูกระงับการใช้งาน โพรโทคอลพิสูจน์ตนผู้ใช้จะเป็นเป้าหมายของการโจมตีแบบแมนอินเดอะมิดเดิล โพรโทคอลเอสเอสเอสจะไม่สามารถป้องกันการเปลี่ยนแปลงข้อความ หาก MAC ไม่ถูกนำไปใช้

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 7

โพรโทคอลชั้นทรานสปอร์ตเอสเอสเอช 2

7.1 แนะนำ

ชั้นทรานสปอร์ตเอสเอสเอช 2 เป็นโพรโทคอลนำส่งในระดับล่างที่ปลอดภัย ทำให้เกิดการเข้ารหัสที่แข็งแกร่ง การพิสูจน์โฮสต์แบบเข้ารหัส และการปกป้องความสมบูรณ์ของข้อมูล

การพิสูจน์ตนในระดับโพรโทคอลนี้ตั้งอยู่บนพื้นฐานของโฮสต์ (host-based) โพรโทคอลนี้ไม่ได้ดำเนินการพิสูจน์ตนผู้ใช้ โดยการพิสูจน์ตนผู้ใช้งานจะกระทำในโพรโทคอลระดับที่สูงขึ้นไป

โพรโทคอลนี้ถูกออกแบบมาให้เรียบง่าย 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

7.2 การก่อตั้งการเชื่อมต่อ (Connection Setup)

โพรโทคอลเอสเอสเอช 2 ทำงานบน 8 บิตอย่างชัดเจน ลำเลียงข้อมูลแบบไบนารี-ทรานสพาเรนต์ (binary-transparent transport) การลำเลียงพื้นฐานควรป้องกันการเกิดความผิดพลาดเนื่องจากการส่งผ่าน เช่นความผิดพลาดอันเป็นสาเหตุให้การเชื่อมต่อเอสเอสเอช 2 ต้องยุติลง โคลเอ็นต์เริ่มทำการเชื่อมต่อ

7.2.1 ใช้งานบนทีซีพี/ไอพี (Use over TCP/IP)

เมื่อใช้งานบนทีซีพี/ไอพี ปกติเซิร์ฟเวอร์จะรอฟังการเชื่อมต่อที่พอร์ต 22 ซึ่งหมายเลขพอร์ตนี้ได้ลงทะเบียนกับ IANA ที่ได้กำหนดให้แก่โพรโทคอลเอสเอสเอช 2 อย่างเป็นทางการแล้ว

7.2.2 การแลกเปลี่ยนเวอร์ชันของโพรโทคอล (Protocol Version Exchange)

เมื่อการเชื่อมต่อถูกก่อตั้งขึ้น ทั้ง 2 ฝ่ายต้องทำการส่งสตริงไเดนติฟิเคชัน (identification string) ซึ่งมันอยู่ในรูปแบบ "SSH-protoversion-softwareversioncomments" ตามด้วยอักขระ carriage return และ newline (ASCII 13 และ 11 ตามลำดับ) ทั้งสองฝั่งต้องสามารถดำเนินการกับสตริงไเดนติฟิเคชันที่ไม่มีอักขระ carriage return ด้วยได้ ไม่มีการส่งอักขระ null สตริงมีความยาวสูงสุดได้ 255 ตัวอักขระ ซึ่งนับรวมอักขระ carriage return และ newline แล้ว

ในส่วนของสตริงไอน์เดนติเฟอเคชันที่นำหน้าอักขระ carriage return และ newline นั้น จะถูกนำไปใช้ในการแลกเปลี่ยนคีย์แบบดิฟฟี-เฮลแมน (Diffie-Hellman key exchange)

เซิร์ฟเวอร์อาจจะส่งบรรทัดของข้อมูลอื่นก่อนส่งสตริงเวอร์ชัน แต่ละบรรทัดควรจบท้ายด้วยอักขระ carriage return และ newline แต่ละบรรทัดต้องไม่ขึ้นต้นด้วย “SSH-” และควรถูกเข้ารูปแบบแบบ ISO-10646 UTF-8 [RFC-2279] ไคลเอนต์ต้องสามารถดำเนินการกับแต่ละบรรทัดได้ โดยอาจจะเพิกเฉยหรือแสดงให้แก่ผู้ใช้งานบนฝั่งไคลเอนต์ ถ้าแสดง ควรใช้การฟิลเตอร์อักขระควบคุม (อธิบายในบทที่ 6) การใช้งานหลักของคุณสมบัตินี้คืออนุญาตให้ทีซีพี-แรพเพอร์ (TCP-wrappers) แสดงข้อความความผิดพลาดก่อนยุติการเชื่อมต่อ

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

การแลกเปลี่ยนคีย์จะเริ่มทันทีหลังจากการส่งไอน์เดนติไฟเออร์นี้ ทุกแพ็กเก็ตที่ตามหลังสตริงไอน์เดนติไฟเออร์จะต้องใช้โพรโตคอลแพ็กเก็ตไบนารี (binary packet protocol) ซึ่งจะได้อธิบายต่อไป

7.2.3 ความเข้ากันได้กับเวอร์ชันเอสเอสเอช 2 ที่เก่ากว่า (Compatibility With Old SSH Versions)

ในช่วงแห่งการเปลี่ยนแปลง จำเป็นที่จะต้องทำงานเข้ากันได้กับไคลเอนต์และเซิร์ฟเวอร์ที่ได้เคยติดตั้งไปแล้วซึ่งใช้โพรโตคอลที่เป็นเวอร์ชันที่เก่ากว่า ในหัวข้อนี้เกี่ยวข้องกับการอิมพลีเมนต์ที่รองรับความเข้ากันได้กับเวอร์ชันเอสเอสเอช 2.1.x เท่านั้น

7.2.3.1 ไคลเอนต์เก่าเซิร์ฟเวอร์ใหม่

การอิมพลีเมนต์เซิร์ฟเวอร์ อาจรองรับแฟล็ก “compatibility” ที่สามารถกำหนดค่าได้ เมื่อแฟล็กถูกเซต จะสามารถเข้ากันได้กับเวอร์ชันเก่า เซิร์ฟเวอร์ควรบ่งบอกเวอร์ชันของโพรโตคอลเป็น “1.99” ไคลเอนต์ที่ใช้เวอร์ชัน 2.0 ต้องสามารถบ่งชี้ได้ว่าเหมือนเวอร์ชัน “2.0” ในโหมดนี้เซิร์ฟเวอร์ไม่ควรส่งอักขระ carriage return (ASCII 13) หลังสตริงไอน์เดนติเฟอเคชันของเวอร์ชัน

ในโหมด compatibility เซิร์ฟเวอร์ไม่ควรส่งข้อมูลอื่นอีกหลังจากส่งสตริงของตน จนกว่าจะได้รับการสตริงไอน์เดนติเฟอเคชันจากไคลเอนต์ จากนั้นเซิร์ฟเวอร์จะสามารถพิจารณาได้ว่าเซิร์ฟเวอร์ใช้โพรโตคอลเก่าหรือไม่ และสามารถเปลี่ยนไปใช้โพรโตคอลเก่าได้ถ้าจำเป็น ในโหมด compatibility เซิร์ฟเวอร์ต้องไม่ส่งข้อมูลเพิ่มก่อนสตริงเวอร์ชัน

เมื่อความเข้ากันได้กับเวอร์ชันเก่าไม่มีความจำเป็นต่อไป เซิร์ฟเวอร์อาจส่งข้อมูลแลกเปลี่ยนคีย์ส่วนแรกทันทีหลังจากสตริงไอน์เดนติเฟอเคชัน

7.2.3.2 โคลเอนต์ใหม่ เซิร์ฟเวอร์เก่า

เนื่องจากโคลเอนต์ใหม่อาจส่งข้อมูลเพิ่มทันทีหลังจากสตรีงไอดีเนติฟิเคชันของมัน (ก่อนการรับไอดีเนติฟิเคชันของเซิร์ฟเวอร์) ทำให้เมื่อโคลเอนต์รู้ว่าเซิร์ฟเวอร์ใช้โปรโตคอลเก่า โปรโตคอลของเซิร์ฟเวอร์อาจล้าสมัยไปแล้ว กรณีนี้ โคลเอนต์ควรทำการปิดการเชื่อมต่อกับเซิร์ฟเวอร์ และทำการเชื่อมต่อใหม่ด้วยโปรโตคอลเก่า

7.3 โปรโตคอลแพ็กเก็ตแบบไบนารี (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 ถ้าการบีบอัดได้ถูกเจรจาแล้ว ฟิลด์นี้จะถูกบีบอัด โดยในตอนเริ่มต้นการบีบอัดต้องเป็น "none"
padding	เป็น padding ความยาวไม่เจาะจง ซึ่งทำให้ความยาวรวมของ (packet_length padding_length payload padding) เป็นจำนวนเท่าของขนาดบล็อกของไซเฟอร์หรือ 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 ต้องเป็นจำนวนเท่าของขนาดบล็อกของไซเฟอร์หรือ 8 ข้อบังคับนี้ต้องถูกบังคับใช้เมื่อใช้ไซเฟอร์แบบสตรีม สังเกตว่าฟิลด์ packet_length จะถูกเข้ารหัสด้วย และการดำเนินการกับมันต้องการความเอาใจใส่เป็นพิเศษในขณะที่ทำการส่งหรือรับแพ็กเก็ต

7.3.1 ความยาวแพ็กเก็ตสูงสุด (Maximum Packet Length)

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

7.3.2 การบีบอัด (Compression)

ถ้าการบีบอัดถูกเจรจา ฟیلด์ *payload* (และเพียงฟیلด์นี้เท่านั้น) จะถูกบีบอัดด้วยอัลกอริทึมที่เจรจา ฟیلด์ *length* และ *MAC* จะถูกคำนวณจาก *payload* ที่บีบอัดแล้ว การเข้ารหัสจะถูกทำหลังจากการบีบอัด การบีบอัดอาจเป็นแบบสเตทฟูล (stateful) ขึ้นอยู่กับวิธีการ การบีบอัดของแต่ละทิศทางต้องเป็นอิสระต่อกัน และการอิมพลีเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมสำหรับแต่ละทิศทางเป็นอิสระจากกันด้วย

วิธีการบีบอัดถูกนิยามไว้ในขณะนี้

<i>none</i>	REQUIRED	<i>no compression</i>
<i>zlib</i>	OPTIONAL	GNU ZLIB (LZ77) compression

การบีบอัด *zlib* ถูกอธิบายไว้ใน [RFC-1950] และใน [RFC-1951] เนื้อหาการบีบอัดถูกเตรียมหลังจากแต่ละการแลกเปลี่ยนคีย์ และถูกส่งจากแพ็กเก็ตหนึ่งไปยังแพ็กเก็ตถัดไปด้วยการพาเซ็ลฟลัช (partial flush) ครั้งหนึ่ง โดยทำที่ส่วนท้ายของแต่ละแพ็กเก็ต การพาเซ็ลฟลัชหมายถึงข้อมูลทั้งหมดจะถูกเป็นเอาท์พุท แต่แพ็กเก็ตถัดไปจะใช้ตารางการบีบอัด (compression tables) ต่อจากท้ายแพ็กเก็ตที่แล้ว

วิธีการเพิ่มเติมอาจถูกนิยามได้ตามข้อกำหนดในบทที่ 6

7.3.3 การเข้ารหัส (Encryption)

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

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

ไซเฟอร์ในแต่ละทิศทางต้องทำงานเป็นอิสระจากทิศทางข้าม และการอิมพลิเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมสำหรับแต่ละทิศทางเป็นอิสระกัน (ถ้าอัลกอริทึมที่อนุญาตให้ใช้ตามนโยบายของแต่ละฝั่ง)

ไซเฟอร์ถูกนิยามไว้ในขณะนี้ดังนี้

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

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

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

ไซเฟอร์ *twofish-cbc*
หรือ *twofish256-cbc* เป็น Twofish ในโหมด CBC ด้วยคีย์ขนาด 128 บิต ซึ่งอธิบายไว้ใน [TWOFISH] เป็นไซเฟอร์แบบบล็อกด้วยบล็อกขนาด 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 และการถูกใช้ด้วยความระมัดระวัง
- 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

7.3.4 ความสมบูรณ์ของข้อมูล (Data Integrity)

ความสมบูรณ์ของข้อมูลถูกปกป้องด้วยการเพิ่มและเพิกเกิดด้วย MAC ซึ่งถูกคำนวณมาจากความลับร่วม หมายเลขลำดับเพิกเกิด และข้อมูลภายในเพิกเกิด

อัลกอริทึมพิสูจน์ข้อความและคีย์ ถูกเจรจาระหว่างการแลกเปลี่ยนคีย์ ในตอนเริ่มต้น MAC จะไม่มีผล และความยาวของมันต้องเป็นศูนย์ หลังจากการแลกเปลี่ยนคีย์ MAC ที่เลือกไว้จะถูกคำนวณก่อนการเข้ารหัสจากการต่อข้อมูลในเพิกเกิดคือ

$$mac = MAC(key, sequence_number || unencrypted_packet)$$

โดย

unencrypted_packet คือทั้งเพิกเกิด 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 และการอิมพลิเมนต์ต้องอนุญาตให้เลือกอัลกอริทึมเป็นอิสระต่อกันระหว่างทั้งสองทิศทาง

ไบนารีของ MAC ที่เป็นผลลัพธ์จากอัลกอริทึม MAC ต้องถูกส่งลำเลียงโดยไม่ถูกเข้ารหัส โดยมันจะเป็นส่วนท้ายของแพ็กเก็ต จำนวนไบนารีของ MAC ขึ้นกับอัลกอริทึมที่เลือก

อัลกอริทึม MAC ที่นิยามไว้ในขณะนี้มีดังนี้

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

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

อัลกอริทึมแฮชถูกอธิบายไว้ใน [SCHNEIER]

วิธีการ “none” ไม่แนะนำให้อำนาจ ถ้าใช้วิธีการนี้ ผู้โจมตีแบบแอกทีฟอาจสามารถเปลี่ยนแปลงข้อมูลที่ส่งผ่าน

วิธีการเพิ่มเติมอาจถูกนิยามตามข้อกำหนดในบทที่ 6

7.3.5 วิธีการแลกเปลี่ยนคีย์ (Key Exchange Methods)

วิธีการแลกเปลี่ยนคีย์ระบุถึงการสร้างคีย์สำหรับเซสชันแบบใช้ครั้งเดียว (one-time session keys) ที่จะนำไปใช้ในการเข้ารหัสและการพิสูจน์ตน นั้นทำอย่างไร และระบุถึงการพิสูจน์เซิร์ฟเวอร์นั้นทำอย่างไร

มีเพียงวิธีการแลกเปลี่ยนคีย์เดียวที่จำเป็น นิยามไว้ดังนี้

<i>diffie-hellman-group1-sha1</i>	REQUIRED
-----------------------------------	----------

7.3.6 อัลกอริทึมคีย์สาธารณะ (Public Key Algorithms)

โพรโตคอลนี้ถูกออกแบบมาให้สามารถใช้งานร่วมกับแทบทุก รูปแบบคีย์สาธารณะ การเข้ารูปแบบ และอัลกอริทึม (การลงลายเซ็น และหรือ การเข้ารหัส)

มีหลายแง่มุมที่ใช้ในการนิยามชนิดของคีย์สาธารณะ

- รูปแบบคีย์ (key format) คีย์ถูกเข้ารูปแบบอย่างไร และใบรับรองอยู่ในรูปแบบใด บล็อกคีย์ในโพรโตคอลนี้อาจบรรจุใบรับรอง
- อัลกอริทึมการลงลายเซ็น และหรือการเข้ารหัส คีย์บางชนิดไม่รองรับทั้งการลงลายเซ็นและการเข้ารหัส การใช้คีย์อาจถูกบังคับตามข้อกำหนดของนโยบายเช่นในใบรับรอง ในกรณีนี้ นโยบายที่ต่างกันควรกำหนดคีย์ที่ต่างชนิดกัน
- การเข้ารูปแบบของ ลายเซ็น และหรือ ข้อมูลที่เข้ารหัสแล้ว อาจรวมถึง การแพคคิง, ลำดับไบนารี 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) ปกติแล้วมันจะไม่อยู่ในบล็อกคีย์

ใบรับรองและคีย์สาธารณะถูกเข้ารูปแบบดังนี้

string certificate or public key format identifier
byte[n] key/certificate data

ส่วนของใบรับรองอาจเป็นสตริงว่าง (ความยาวศูนย์) แต่คีย์สาธารณะจำเป็นต้องมี เป็นคีย์สาธารณะที่จะถูกใช้ในการพิสูจน์ตน กลุ่มใบรับรองที่บรรจุในบล็อกใบรับรองสามารถนำไปใช้ในการกำหนดสิทธิ์

รูปแบบคีย์ "ssh-dss" กำหนดไว้ดังนี้

```
string  "ssh-dss"
mpint   p
mpint   q
mpint   g
mpint   y
```

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

การลงลายเซ็นและการตรวจสอบที่ใช้รูปแบบคีย์นี้ โดยเป็นไปตามมาตรฐานลายเซ็นดิจิทัล (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) ไม่มีส่วนความยาวหรือแพคคิง ไม่ลงลายเซ็น และเรียงลำดับไบนารีแบบเน็ตเวิร์ก

รูปแบบคีย์ "ssh-rsa" ถูกกำหนดการเข้ารูปแบบดังนี้

```
string  "ssh-rsa"
mpint   e
mpint   n
```

โดยพารามิเตอร์ e และ n ก่อรูปแบบเป็นบล็อบลายเซ็น

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

ลายเซ็นผลลัพธ์มีรูปแบบดังนี้

```
string  "ssh-rsa"
string  rsa_signature_blob
```


rsa_signature_blob ถูกเข้ารูปแบบเป็นสตริงที่บรรจุ s (เป็น integer ไม่มีส่วนความยาวหรือแพคคิง ไม่ลงลายเซ็น และเรียงลำดับไบนารีแบบเน็ตเวิร์ก)

- 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” บ่งชี้ถึงบล็อกใบรับรองบรรจุกลุ่มใบรับรองแบบ SPKI รูปแบบใบรับรอง SPKI ถูกอธิบายไว้ใน [RFC-2693] วิธีการนี้บ่งชี้ว่าคีย์ (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] วิธีการนี้บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ RSA
- วิธี “pgp-sign-dss” คล้ายกับวิธี “pgp-sign-rsa” แต่บ่งชี้ว่าคีย์ (หรือหนึ่งในกลุ่มคีย์ในใบรับรอง) เป็นคีย์ DSS

7.4 การแลกเปลี่ยนคีย์ (Key Exchange)

การแลกเปลี่ยนคีย์เริ่มโดยแต่ละฝั่งส่งสัดของอัลกอริทึมที่รองรับ แต่ละฝั่งจะมีอัลกอริทึมเสนอในแต่ละหมวด และถือว่าการอิมพลีเมนต์ส่วนใหญ่ใช้อัลกอริทึมเสนอเหมือนกัน แต่ละฝั่งอาจเจออัลกอริทึมที่อีกฝั่งจะใช้ และอาจส่งแพ็กเก็ตแลกเปลี่ยนคีย์เริ่มต้นที่เป็นไปตามอัลกอริทึมถ้าตรงกับวิธีการเสนอ

การเจอถูกพิจารณาว่าผิด ถ้า

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

หากไม่ตรงกับที่กล่าวมา การเจอจะถูกพิจารณาว่าถูกต้อง และแพ็กเก็ตที่ถูกส่งจะถูกจัดการเป็นแพ็กเก็ตแลกเปลี่ยนคีย์เริ่มต้น

อย่างไรก็ตาม ถ้าการเจอผิดพลาด และแพ็กเก็ตได้ถูกส่งจากฝั่งใดฝั่งหนึ่งหรือจากทั้งสองฝั่งแล้ว แพ็กเก็ตเหล่านั้นจะต้องถูกเพิกเฉย (แม้ว่าการเจอผิดพลาดไม่ได้มีผลต่อข้อมูลภายในแพ็กเก็ต) และฝั่งที่เหมาะสมต้องส่งแพ็กเก็ตเริ่มต้นที่ถูกต้อง

การพิสูจน์เซิร์ฟเวอร์ในการแลกเปลี่ยนคีย์อาจทำโดยอ้อม หลังจากการแลกเปลี่ยนคีย์ด้วยการพิสูจน์เซิร์ฟเวอร์โดยอ้อมแล้ว โคลเ็นต์ต้องรอการตอบสนองของแมสเซอร์ซึ่งขอบริการที่ส่งไป ก่อนที่จะทำการส่งข้อมูลอื่นต่อไป

7.4.1 การเจรจาอัลกอริทึม (Algorithm Negotiation)

การแลกเปลี่ยนคีย์เริ่ม โดยแต่ละฝั่งส่งแพ็คเกจต่อไปนี้

```
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)
```

แต่ละสตริงอัลกอริทึมต้องเป็นลิสต์ของชื่ออัลกอริทึมที่สนับสนุนด้วยอักขระ (ดูการตั้งชื่ออัลกอริทึมในบทที่ 6) แต่ละอัลกอริทึมที่รองรับ (อนุญาต) ต้องถูกลิสต์ตามลำดับการเสนอ

อัลกอริทึมแรกในแต่ละลิสต์ต้องเป็นอัลกอริทึมเสนอ (เดา) แต่ละสตริงต้องบรรจุชื่ออัลกอริทึมอย่างน้อยที่สุด 1 ชื่อ

cookie

คุกกี้ต้องเป็นค่าสุ่มที่สร้างขึ้นโดยผู้ส่ง จุดประสงค์คือเพื่อทำให้การที่ฝั่งใดฝั่งหนึ่งรู้จัก และไอดีเนติฟายเออร์ของเซสชัน (session identifier) นั้นเป็นไปได้

kex_algorithm

อัลกอริทึมการแลกเปลี่ยนคีย์ได้นิยามดังที่กล่าวมา อัลกอริทึมแรกต้องเป็นอัลกอริทึมเสนอ (และเดา) ถ้าทั้งสองฝั่งเดาตรงกัน อัลกอริทึมนั้นจะต้องถูกใช้ ในทางตรงกันข้าม อัลกอริทึมต่อไปจะต้องถูก

เลือกใช้ในการแลกเปลี่ยนคีย์ 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

เป็นลิสต์ของอัลกอริทึมบีบอัดที่รองรับ ตามลำดับการเสนอ อัลกอริทึมที่เลือกต้องเป็นอัลกอริทึมแรกในลิสต์ของไคลเอนต์ซึ่งอยู่ในลิสต์ของเซิร์ฟเวอร์ด้วย ถ้าไม่มีอัลกอริทึมใดเลย ทั้งสองฝั่งต้องยุติการเชื่อมต่อ

สังเกตว่า “none” ต้องอยู่ในลิสต์ด้วย ถ้ายอมรับ ชื่ออัลกอริทึมบีบอัดแสดงไว้ในหัวข้อการบีบอัด

languages

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

first_kex_packet_follows

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

หลังจากได้รับแพ็กเก็ต SSH_MSG_KEXINIT จากฝั่งตรงข้ามแล้ว แต่ละฝั่งจะรู้ว่าการเดาของตนถูกต้องหรือไม่ ถ้าการเดาของฝั่งตรงข้ามผิด ฟิลด์นี้จะเป็น TRUE แพ็กเก็ตต่อไปจะถูกเพิกเฉย และทั้งสองฝั่งต้องปฏิบัติตามที่กำหนดโดยวิธีการแลกเปลี่ยนคีย์ที่เจรจาไว้ ถ้าการเดาถูกต้อง การแลกเปลี่ยนคีย์ต้องดำเนินการต่อโดยใช้แพ็กเก็ตเดา

หลังจากการแลกเปลี่ยนแพ็กเก็ต KEXINIT แล้ว อัลกอริทึมการแลกเปลี่ยนคีย์จะทำงาน มันอาจเกี่ยวข้องกับการแลกเปลี่ยนหลายแพ็กเก็ต ตามที่ระบุโดยวิธีการแลกเปลี่ยนคีย์

7.4.2 เอาท์พุทที่ได้จากการแลกเปลี่ยนคีย์ (Output from Key Exchange)

การแลกเปลี่ยนคีย์สร้าง 2 ค่าคือ ค่าความลับร่วม (shared secret) K และแฮชแลกเปลี่ยน (exchange hash) H คีย์การเข้ารหัสและคีย์การพิสูจน์ตนได้มาจาก 2 ค่านี้ แฮชแลกเปลี่ยน H จากการแลกเปลี่ยนคีย์ครั้งแรก ถูกใช้เป็นไคเดนติไฟเออร์ของเซสชันด้วย ซึ่งเป็นไคเดนติไฟเออร์ที่เป็นเอกลักษณ์สำหรับการเชื่อมต่อ ถูกใช้ในการพิสูจน์ตนโดยเป็นส่วนหนึ่งของข้อมูลที่ถูกลงลายเซ็น เพื่อใช้พิสูจน์ความเป็นเจ้าของของคีย์ส่วนตัว เมื่อคำนวณแล้ว ไคเดนติไฟเออร์ของเซสชันจะไม่ถูกเปลี่ยนแปลง แม้ว่าคีย์จะถูกเปลี่ยนแปลงใหม่ในภายหลัง

แต่ละวิธีการแลกเปลี่ยนคีย์ระบุฟังก์ชันแฮชที่ใช้ในการแลกเปลี่ยนคีย์ ในการได้มาซึ่งคีย์ต้องใช้อัลกอริทึมแฮชเดียวกัน จากนั้นจะเรียกว่า *HASH*

คีย์เข้ารหัสต้องถูกคำนวณจากค่าแฮชของค่าที่รู้และ K ดังนี้

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

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

$$K1 = HASH(K \parallel H \parallel X \parallel session_id) \quad (X \text{ is e.g. } "A")$$

$$K2 = HASH(K \parallel H \parallel K1)$$

$$K3 = HASH(K \parallel H \parallel K1 \parallel K2)$$

$$\dots$$

$$key = K1 \parallel K2 \parallel K3 \parallel \dots$$

กระบวนการนี้จะสูญเสียเอนโทรปี (entropy) ถ้าผลรวมของเอนโทรปีใน K มากกว่าขนาดสเตทภายในของ $HASH$

7.4.3 การนำคีย์ไปใช้งาน (Taking Keys Into Use)

การแลกเปลี่ยนคีย์จบลงโดยแต่ละฝั่งทำการส่งแพสเซจ `SSH_MSG_NEWKEYS` แพสเซจนี้ถูกส่งพร้อมกับคีย์เดิมและอัลกอริทึมเดิม ทุกแพสเซจที่ส่งหลังจากนี้ต้องใช้คีย์ใหม่และอัลกอริทึมใหม่

เมื่อได้รับแพสเซจนี้ คีย์ใหม่และอัลกอริทึมใหม่ต้องถูกนำมาใช้ในการรับ

แพสเซจนี้เป็นแพสเซจที่ถูกต้องหลังจากการแลกเปลี่ยนคีย์เท่านั้น รวมถึงแพสเซจ `SSH_MSG_DEBUG`, `SSH_MSG_DISCONNECT` และ `SSH_MSG_IGNORE` ด้วย จุดประสงค์ของแพสเซจนี้เพื่อให้แน่ใจว่า แต่ละฝั่งสามารถที่จะตอบสนองด้วยแพสเซจยุติการเชื่อมต่อที่อีกฝั่งสามารถเข้าใจเมื่อการแลกเปลี่ยนคีย์มีบางสิ่งผิดพลาดไป หลังจากการแลกเปลี่ยนคีย์ การอิมพลิเมนต์ต้องไม่ยอมรับแพสเซจอื่นๆ ก่อนที่จะได้รับ `SSH_MSG_NEWKEYS`

byte `SSH_MSG_NEWKEYS`

7.5 การแลกเปลี่ยนคีย์ใหม่ (Key Re-Exchange)

การแลกเปลี่ยนคีย์ใหม่จะถูกเริ่มโดยการส่งแพ็กเก็ต `SSH_MSG_KEXINIT` เมื่อไม่สามารถทำการแลกเปลี่ยนคีย์ได้ (อธิบายไว้ในหัวข้อการเจรจาอัลกอริทึม) เมื่อได้รับแพ็กเก็ตนี้ จะต้องตอบสนองด้วยแพ็กเก็ต `SSH_MSG_KEXINIT` ของตนเอง ยกเว้นเมื่อ `SSH_MSG_KEXINIT` ที่ได้รับได้ถูกตอบกลับไปแล้ว ฝ่ายใดฝ่ายหนึ่งอาจทำการเริ่มแลกเปลี่ยนคีย์ใหม่ แต่บทบาทต้องไม่ถูกเปลี่ยนไป (เช่น เซิร์ฟเวอร์ยังคงเป็นเซิร์ฟเวอร์ และไคลเอนต์ก็ยังคงเป็นไคลเอนต์)

การแลกเปลี่ยนคีย์ใหม่ถูกดำเนินการโดยใช้การเข้ารหัสอะไรก็ได้ซึ่งจะมีผลเมื่อการแลกเปลี่ยนถูกเริ่มต้นขึ้น การเข้ารหัส, การบีบอัด และวิธีการ MAC จะไม่เปลี่ยนก่อนที่ `SSH_MSG_NEWKEYS` ใหม่จะถูกส่งหลังจากการแลกเปลี่ยนคีย์ (เหมือนกับการแลกเปลี่ยนคีย์เริ่มต้น) การแลกเปลี่ยนใหม่ถูกทำกระบวนการเหมือนกับการแลกเปลี่ยนคีย์เริ่มต้นเว้นแต่ไคเดนติไฟเออร์ของเซสชันจะยังคงไม่เปลี่ยนแปลง ทั้งนี้ยอมให้มีการเปลี่ยนอัลกอริทึมบางส่วนหรือทั้งหมดระหว่างการแลกเปลี่ยนคีย์ใหม่ และคีย์โฮสต์สามารถเปลี่ยนใหม่ได้ด้วย คีย์และอินซีเคิลไคเซชันเวกเตอร์ทั้งหมดถูกคำนวณใหม่หลังจากการแลกเปลี่ยน การบีบอัดและการเข้ารหัสจะถูกรีเซ็ต

แนะนำให้เปลี่ยนคีย์หลังจากทำการส่งผ่านข้อมูลครบทุกกิกะไบต์ หรือทุก ๆ ชั่วโมงของเวลาการเชื่อมต่อ โดยอะไรครบก็ทำวิธีนั้นก่อน อย่างไรก็ตามเนื่องจากการแลกเปลี่ยนคีย์ใหม่เป็นการทำงานคีย์ของสาธารณะ จำเป็นที่ใช้การประมวลผลที่คุ้มค่าและไม่ควรทำบ่อยเกินไป

ข้อมูลโปรแกรมประยุกต์จำนวนมากอาจถูกส่งหลังจากแพ็กเก็ต `SSH_MSG_NEWKEYS` ได้ถูกส่งไปแล้ว การแลกเปลี่ยนคีย์ไม่มีผลกระทบต่อโปรโตคอลที่วางตัวอยู่เหนือชั้นทรานสปอร์ตเอสเอสเอช 2

7.6 การร้องขอบริการ (Service Request)

หลังจากการแลกเปลี่ยนคีย์ ไคลเอนต์ร้องขอบริการ บริการถูกบ่งชี้โดยชื่อ รูปแบบของชื่อและกระบวนการนิยามชื่อใหม่ได้นิยามไว้ในบทที่ 6

ปัจจุบันชื่อต่อไปนี้ถูกสงวนไว้

`ssh-userauth`

`ssh-connection`

นโยบายการตั้งชื่อบน โคลออลถูกประยุกต์เข้ากับชื่อบริการ เหมือนกับที่ประยุกต์เข้ากับชื่ออัลกอริทึม บริการบนโคลออลควรใช้ไวยากรณ์ “`servicename@domain`”

`byte` `SSH_MSG_SERVICE_REQUEST`

`string` `service name`

ถ้าเซิร์ฟเวอร์ปฏิเสธการร้องขอบริการนี้ มันควรส่งแพ็กเก็ต `SSH_MSG_DISCONNECT` ที่เหมาะสม และต้องยุติการเชื่อมต่อ

เมื่อการบริการเริ่มขึ้น มันสามารถเข้าใช้ไอน์เดนติไฟเออร์ของเซสชันที่สร้างขึ้นระหว่างการแลกเปลี่ยนคีย์

ถ้าเซิร์ฟเวอร์รองรับการบริการ (และอนุญาตให้ไคลเอ็นต์ใช้ได้) เซิร์ฟเวอร์ต้องตอบสนองดังนี้

```
byte    SSH_MSG_SERVICE_ACCEPT
string  service name
```

หมายเลขแมสเซจที่ใช้โดยบริการควรอยู่ในขอบเขตสงวนไว้สำหรับบริการนั้น ๆ (ดูในหัวข้อสรุปหมายเลขแมสเซจ) ขั้นตอนต่อไปจะดำเนินการกระบวนการด้วยแมสเซจของมันต่อไป

สังเกตว่าหลังจากการแลกเปลี่ยนด้วยการพิสูจน์เซิร์ฟเวอร์แล้ว ไคลเอ็นต์ต้องรอการตอบสนองต่อแมสเซจการร้องขอบริการของมันก่อนที่จะส่งข้อมูลใด ๆ ต่อไป

7.7 แมสเซจเพิ่มเติม (Additional Messages)

แต่ละฝั่งอาจส่งแมสเซจต่อไปนี้ ณ เวลาใดก็ได้

7.7.1 แมสเซจยุติการเชื่อมต่อ (Disconnection Message)

```
byte    SSH_MSG_DISCONNECT
uint32  reason code
string  description [RFC-2279]
string  language tag [RFC-1766]
```

แมสเซจนี้เป็นเหตุให้การเชื่อมต่อยุติลงทันที ทุกการอิมพลิเมนต์ต้องสามารถจัดการกับแมสเซจนี้ได้ 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 การฟิเตอร์อักขระควบคุมดังที่อธิบายในบทที่ 6 ควรถูกนำมาใช้ เพื่อหลีกเลี่ยงการโจมตีโดยการส่งอักขระควบคุมเทอร์มินอล

7.7.2 เมสเซจข้อมูลที่ถูกเพิกเฉย (Ignored Data Message)

byte SSH_MSG_IGNORE
string data

ทุกการอิมพลีเมนต์ต้องเข้าใจ (และเพิกเฉย) เมสเซจนี้ (หลังจากได้รับเวอร์ชันโปรโตคอล) ไม่มีการอิมพลีเมนต์ใดจำเป็นต้องส่งมัน

7.7.3 เมสเซจดีบั๊ก (Debug Message)

byte SSH_MSG_DEBUG
boolean always_display
string message [RFC-2279]
string language tag [RFC-1766]

ทุกการอิมพลีเมนต์ต้องเข้าใจเมสเซจนี้ แต่อนุญาตให้เพิกเฉยต่อเมสเซจนี้ เมสเซจนี้ถูกใช้ในการส่งผ่านข้อมูลที่ช่วยในการดีบั๊กแกกั๊กฝั่ง ถ้า ‘always_display’ เป็น TRUE ‘message’ ควรถูกแสดงในทางกลับกัน มันไม่ควรถูกแสดงออกมา เว้นแต่ผู้ใช้ได้ร้องขอข้อมูลดีบั๊ก

‘message’ ไม่จำเป็นต้องมีอักขระ newline แต่อย่างไรก็ตาม อนุญาตให้ประกอบขึ้นจากหลายบรรทัดได้ ซึ่งแยกแต่ละบรรทัดด้วยคู่อักขระ CRLF (Carriage Return-Line Feed)

ถ้าสตริง ‘message’ ถูกแสดงผล ควรใช้การฟีดเตอร์อักขระควบคุมเทอร์มินอลดังที่อธิบายไว้ใน บทที่ 6 เพื่อหลีกเลี่ยงการ โจมตีโดยการส่งอักขระควบคุมเทอร์มินอล

7.7.4 **แอสเซจสงวน (Reserved Message)**

การอิมพลีเมนต์ต้องตอบสนองต่อทุกแอสเซจที่ไม่สามารถแยกแยะเข้าใจได้ด้วยแอสเซจ SSH_MSG_UNIMPLEMENTED ตามลำดับแอสเซจที่ได้รับซึ่งแอสเซจเหล่านี้ต้องถูกเพิกเฉย เวอร์ชัน โพรโตคอลใหม่อาจนิยามความหมายอื่นแก่ชนิดของแอสเซจเหล่านี้

```
byte      SSH_MSG_UNIMPLEMENTED
uint32    packet sequence number of rejected message
```

7.8 **สรุปหมายเลขแอสเซจ**

หมายเลขแอสเซจต่อไปนี้ถูกนิยามไว้ใน โพรโตคอลนี้

```
#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 ใช้สำหรับเพิกเกิด kex วิธีการ kex ที่ต่างกันอาจใช้หมายเลขในช่วงนี้ใหม่ได้*/

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

7.9 **พิจารณาความปลอดภัย**

โพรโตคอลนี้ทำให้เกิดแซนแนลเข้ารหัสที่ปลอดภัย มันทำให้มีการพิสูจน์โฮสต์เซิร์ฟเวอร์ แลกเปลี่ยนคีย์ เข้ารหัส และปกป้องความสมบูรณ์ของข้อมูล มันยังให้อิเอนดิฟายเออร์ของเซสชันที่เป็นเอกลักษณ์ซึ่งอาจถูกนำไปใช้โดยโพรโตคอลชั้นที่สูงกว่า

บางครั้งโพรโตคอลนี้อาจจะถูกใช้โดยปราศจากการร้องขอยืนยันใช้ความสัมพันธ์ที่ไว้วางใจได้ของระหว่างคีย์โฮสต์เซิร์ฟเวอร์และชื่อโฮสต์เซิร์ฟเวอร์ การใช้เช่นนี้ทำให้ไม่แยกออกจากความไม่ปลอดภัยได้ แต่อาจจะจำเป็นในสภาพแวดล้อมที่ไม่ปลอดภัย และยังคงป้องกันการโจมตีแบบพาสซีฟ อย่างไรก็ตาม ผู้implement โพรโตคอลที่ทำงานบนโพรโตคอลนี้ควรคำนึงถึงความเป็นไปได้ด้วย

โพรโตคอลนี้ถูกออกแบบมาให้ใช้บนการลำเลียงที่น่าเชื่อถือ ถ้าการส่งมีความผิดพลาด การเชื่อมต่อต้องถูกปิดลง ถ้ากรณีนี้เกิดขึ้นการเชื่อมต่อควรถูกก่อตั้งขึ้นใหม่ การโจมตีเพื่อให้หยุดบริการ (denial of service) ในรูปแบบนี้จึงแทบจะเป็นไปไม่ได้ที่จะหลีกเลี่ยง

โพรโตคอลไม่ได้ถูกออกแบบมาให้ทำลายแซนแนลที่ลอบมอง ยกตัวอย่างเช่น แพคคิง, แมสเซจ `SSH_MSG_IGNORE` และจุดอื่นอีกหลายจุด ในโพรโตคอลสามารถถูกใช้ในการส่งข้อมูลที่แอบแฝง และผู้รับไม่มีทางไว้วางใจในการตรวจสอบว่าข้อมูลข่าวสารจำพวกนี้กำลังถูกส่งหรือไม่

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.
- [TWOFISH] Schneir, B., et al: "The Twofish Encryption Algorithm: A 128-Bit Block Cipher", 1st edition, John Wiley & Sons, 22 March 1999.

บทที่ 8

โพรโทคอลการพิสูจน์ตนเอสเอสเอช 2

8.1 แนะนำ

โพรโทคอลการพิสูจน์ตนเอสเอสเอช 2 เป็นโพรโทคอลการพิสูจน์ตนผู้ใช้ที่ทำงานแบบจุดประสงค์ทั่วไป (general purpose) มุ่งหมายให้ทำงานบนโพรโทคอลชั้นทรานสปอร์ตเอสเอสเอช 2 โพรโทคอลนี้เป็นโพรโทคอลพื้นฐานที่ให้การป้องกันด้านความสมบูรณ์ (integrity) และด้านความเชื่อมั่น (confidentiality)

ชื่อบริการสำหรับโพรโทคอลนี้คือ “ssh-userauth”

เมื่อโพรโทคอลนี้เริ่มทำงาน มันจะรับไอดีentifายเออร์จากโพรโทคอลที่ต่ำกว่า (เป็นแฮชแลกเปลี่ยน H จากการแลกเปลี่ยนคีย์ครั้งแรก) เซสชันไอดีentifายเออร์บ่งชี้เซสชันอย่างมีเอกลักษณ์และเหมาะต่อการลงลายเซ็น (signing) เพื่อใช้ในการพิสูจน์ความเป็นเจ้าของคีย์ส่วนตัว (private key) โพรโทคอลจึงจำเป็นต้องรู้ว่าโพรโทคอลที่ต่ำกว่าได้ให้การป้องกันด้านความมั่นใจหรือไม่

8.2 เฟรมเวิร์กโพรโทคอลการพิสูจน์ตน (The Authentication Protocol Framework)

เซิร์ฟเวอร์จะกระตุ้นให้เกิดการพิสูจน์ตน โดยบอกไคลเอ็นต์ว่ามีวิธีการพิสูจน์ตนวิธีใดบ้างที่สามารถใช้ในการแลกเปลี่ยน ไคลเอ็นต์มีอิสระในการเลือกวิธีในรายการที่เซิร์ฟเวอร์ให้มา ทำให้เซิร์ฟเวอร์ควบคุมกระบวนการพิสูจน์ตนได้ตามต้องการ แต่ก็ให้ความยืดหยุ่นแก่ไคลเอ็นต์ในการเลือกใช้วิธีที่ไคลเอ็นต์รองรับหรือสะดวกใช้ที่สุดแก่ผู้ใช้ เมื่อเซิร์ฟเวอร์เสนอให้หลายวิธีการ

วิธีการพิสูจน์ตนถูกบ่งชี้ด้วยชื่อวิธี ตามที่ระบุในสเปกตรัมโพรโทคอลเอสเอสเอช 2 วิธี “none” ถูกสงวนไว้ และต้องไม่อยู่ในลิสต์ที่รองรับ อย่างไรก็ตามอาจจะถูกส่งโดยไคลเอ็นต์ก็ได้ เซิร์ฟเวอร์ต้องปฏิเสธการร้องขอนี้เสมอ เว้นแต่ไคลเอ็นต์ได้รับอนุญาตให้ไม่ต้องการพิสูจน์ตนใดๆ กรณีนี้เซิร์ฟเวอร์ต้องยอมรับการร้องขอ จุดประสงค์หลักของการส่งการร้องขอนี้คือเพื่อร้องขอลิสต์วิธีที่รองรับจากเซิร์ฟเวอร์

เซิร์ฟเวอร์ควรมีระยะเวลาในการพิสูจน์ตน และหยุดการเชื่อมต่อ ถ้าการพิสูจน์ตนไม่ได้รับการยอมรับภายในระยะเวลาหมด แนะนำให้ระยะเวลาหมดเป็น 10 นาที ยิ่งไปกว่านั้นการอิมพลิเมนต์ควรจำกัดจำนวนครั้งของความพยายามในการพิสูจน์ตนที่ล้มเหลวของไคลเอ็นต์ในเซสชันหนึ่งๆ (แนะนำว่าจำกัดไว้ที่ 20 ครั้ง) ถ้าความพยายามเกินที่จำกัดไว้ เซิร์ฟเวอร์ควรหยุดการเชื่อมต่อลง

8.2.1 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.
```

มีเพียงฟิลด์แรกๆ ที่ถูกนิยามไว้ ส่วนฟิลด์ที่เหลือขึ้นกับวิธีการพิสูจน์ตน

ชื่อผู้ใช้ *user name* และชื่อบริการ *service name* จะเหมือนเดิมทุกครั้งในความพยายามพิสูจน์ตนใหม่ และอาจเปลี่ยนได้ การอิมพลีเมนต์เซิร์ฟเวอร์ต้องตรวจสอบอย่างระมัดระวังเกี่ยวกับมันในทุกๆ แอสเซจ และถ้ามันมีการเปลี่ยนแปลง เซิร์ฟเวอร์ต้องละทิ้งสแตทของการพิสูจน์ตนที่เก็บไว้ มันต้องหยุดการเชื่อมต่อถ้าชื่อผู้ใช้หรือบริการเปลี่ยนแปลง

ชื่อบริการระบุบริการที่จะสแตตหลังจากการพิสูจน์ตน มีหลายบริการที่แตกต่างกันให้ใช้ ถ้าบริการที่ร้องขอไม่สามารถใช้ได้ เซิร์ฟเวอร์อาจหยุดการเชื่อมต่อทันทีหรือในภายหลังเวลาใดก็ได้ แนะนำให้ส่งแอสเซจหยุดการเชื่อมต่อในกรณีที่บริการที่ร้องขอไม่มี การพิสูจน์ตนต้องไม่ถูกยอมรับ

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

โกลเอ็นด์สามารถส่งคำร้องขอที่เซิร์ฟเวอร์ไม่ได้ลงในลิสต์ยอมรับได้ การส่งคำร้องขอเช่นนี้ไม่ใช่ความผิดพลาด และเซิร์ฟเวอร์ควรปฏิเสธการร้องขอที่มันไม่รู้จัก

การร้องขอพิสูจน์ตนอาจให้ผลลัพธ์ในการแลกเปลี่ยนแอสเซจครั้งต่อไป แอสเซจทั้งหมดนี้จะขึ้นกับวิธีการพิสูจน์ตนที่ใช้ และโกลเอ็นด์อาจจะส่งแอสเซจ *SSH_MSG_USERAUTH_REQUEST* ใหม่ ณ เวลาใดๆ ซึ่งในกรณีเช่นนี้ เซิร์ฟเวอร์ต้องยกเลิกความพยายามพิสูจน์ตนก่อนหน้านี้ และทำการพิสูจน์ตนใหม่

8.2.2 การตอบสนองต่อการร้องขอพิสูจน์ตน (Responses to Authentication Requests)

ถ้าเซิร์ฟเวอร์ปฏิเสธการร้องขอพิสูจน์ตน มันต้องตอบสนองด้วยแอสเซจที่มีรูปแบบดังนี้

```
byte    SSH_MSG_USERAUTH_FAILURE
string  authentications that can continue
boolean partial success
```


โดย “*authentications that can continue*” เป็นลิสต์ชื่อวิธีการพิสูจน์ตนซึ่งแยกกันด้วยเครื่องหมายคอมม่า แนะนำว่าเซิร์ฟเวอร์ควรรวมเฉพาะวิธีที่มิใช่ประโยชน์ได้จริงลงในลิสต์ อย่างไรก็ตาม ไม่ผิดกฎในการรวมวิธีการที่ไม่สามารถใช้ในการพิสูจน์ตนผู้ใช้

การพิสูจน์ตนที่จบลงด้วยความสำเร็จแล้ว ไม่ควรถูกรวมในลิสต์ เว้นแต่มันควรจะถูกปฏิบัติอีกครั้งด้วยเหตุผลบางข้อ

“*partial success*” ต้องเป็น TRUE ถ้าการร้องขอพิสูจน์ตนที่ได้รับการตอบสนองนี้ประสบความสำเร็จ และต้องเป็น FALSE ถ้าการร้องขอทำงานแล้วไม่ประสบความสำเร็จ

เมื่อเซิร์ฟเวอร์ยอมรับการพิสูจน์ตน มันต้องตอบสนองด้วยเมสเสจนี้

byte SSH_MSG_USERAUTH_SUCCESS

สังเกตว่าเมสเสจนี้จะไม่ถูกส่งหลังแต่ละขั้นตอนในลำดับการพิสูจน์ตนหลายวิธี แต่มันจะถูกส่งเพียงครั้งเดียวเมื่อการพิสูจน์ตนทั้งหมดสำเร็จ

ไคลเอ็นต์อาจส่งหลายการร้องขอพิสูจน์ตนโดยไม่ต้องรอการตอบสนองจากการร้องขอก่อนหน้านี้ เซิร์ฟเวอร์ต้องแจ้งทุกการร้องขอที่ล้มเหลวด้วยเมสเสจ SSH_MSG_USERAUTH_FAILURE อย่างไรก็ตาม SSH_MSG_USERAUTH_SUCCESS ต้องถูกส่งอย่างน้อย 1 ครั้ง และเมื่อ SSH_MSG_USERAUTH_SUCCESS ถูกส่ง การร้องขอพิสูจน์ตนที่ได้รับภายหลังการถูกเพิกเฉย

เมสเสจที่ไม่เกี่ยวกับการพิสูจน์ตนซึ่งส่งโดยไคลเอ็นต์ที่ได้รับเมสเสจ SSH_MSG_USERAUTH_SUCCESS แล้ว ต้องถูกส่งผ่านไปยังบริการที่ทำงานอยู่บนโพรโตคอลนี้ เมสเสจสามารถแยกแยะได้โดยหมายเลขเมสเสจที่จะอธิบายต่อไป

8.2.3 การร้องขอพิสูจน์ตนแบบ “none” (The “none” Authentication Request)

ไคลเอ็นต์อาจจะร้องขอลิสต์ของวิธีการพิสูจน์ตน และอาจจะใช้วิธีการพิสูจน์ตนแบบ “none”

ถ้าไม่มีความจำเป็นในการพิสูจน์ตนผู้ใช้ เซิร์ฟเวอร์ต้องส่ง SSH_MSG_USERAUTH_SUCCESS กลับ ในทางตรงกันข้าม ถ้าจำเป็นต้องมีการพิสูจน์ตนผู้ใช้ เซิร์ฟเวอร์ต้องส่ง SSH_MSG_USERAUTH_FAILURE กลับและอาจส่งลิสต์ของวิธีการพิสูจน์ตนกลับด้วย

8.2.4 ความสำเร็จของการพิสูจน์ตนผู้ใช้ (Completion of User Authentication)

การพิสูจน์ตนจะจบลงเมื่อเซิร์ฟเวอร์ตอบสนองด้วย SSH_MSG_USERAUTH_SUCCESS ทุกเมสเสจที่เกี่ยวข้องกับการพิสูจน์ตนที่ได้รับหลังจากการส่งเมสเสจ SSH_MSG_USERAUTH_SUCCESS แล้ว ควรถูกเพิกเฉย

หลังจากเซิร์ฟเวอร์ส่ง SSH_MSG_USERAUTH_SUCCESS แล้ว เซิร์ฟเวอร์จะสแตนด์บายบริการที่ร้องขอไว้

8.2.5 ข้อความแบนเนอร์ (Banner Message)

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

เซิร์ฟเวอร์เอสเอสเอช 2 อาจส่งเมสเสจ SSH_MSG_USERAUTH_BANNER ในเวลาใดก็ได้ก่อนการพิสูจน์ตนจะประสบความสำเร็จ เมสเสจนี้ประกอบด้วยข้อความที่ถูกแสดงแก่ผู้ใช้ไคลเอนต์ก่อนจะมีความพยายามในการทำการพิสูจน์ตน รูปแบบของเมสเสจเป็นดังนี้

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

ไคลเอนต์ควรแสดงข้อความบนหน้าจอโดยดีฟอลต์ อย่างไรก็ตาม เนื่องจากเมสเสจนี้มักจะส่งให้ทุกๆครั้งที่มีความพยายามล็อกอิน และเนื่องจากบางซอร์ฟแวร์ไคลเอนต์จำเป็นต้องเปิดวินโดว์แยกตัวออกมาใหม่เพื่อแสดงคำเตือนนี้ ซอร์ฟแวร์ไคลเอนต์จึงอาจจะอนุญาตให้ผู้ใช้ดิสเอเบิล (disable) การแสดงแบนเนอร์ที่ได้จากเซิร์ฟเวอร์ได้ ฟิลด์ข้อความอาจประกอบขึ้นจากหลายบรรทัด

ถ้าสตรีงข้อความถูกแสดง ควรใช้ฟิเตอร์ตัวอักษรควบคุม (อธิบายในสถาปัตยกรรมโพรโตคอลเอสเอสเอช 2) เพื่อหลีกเลี่ยงการ โจมตีโดยส่งตัวอักษรควบคุมเทอร์มินอล

8.3 หมายเลขเมสเสจของโพรโตคอลพิสูจน์ตน (Authentication Protocol Message Numbers)

หมายเลขเมสเสจทั้งหมดที่ใช้โดยโพรโตคอลพิสูจน์ตนอยู่ในช่วง 50 ถึง 79 ซึ่งเป็นส่วนหนึ่งในช่วงที่สงวนไว้ สำหรับโพรโตคอลที่ทำงานอยู่เหนือโพรโตคอลชั้นทรานสปอร์ตเอสเอสเอช

หมายเลขเมสเสจตั้งแต่ 80 ขึ้นไป ถูกสงวนไว้สำหรับโพรโตคอลที่ทำงานหลังจากโพรโตคอลพิสูจน์ตนนี้ ดังนั้นการได้รับเมสเสจหมายเลขเหล่านี้ ก่อนการพิสูจน์ตนจะสำเร็จ จะเป็นความผิดพลาด ซึ่งเซิร์ฟเวอร์ต้องตอบสนองด้วยการหยุดการเชื่อมต่อ (ควรส่งเมสเสจหยุดการเชื่อมต่อเป็นเมสเสจแรก เพื่อให้ง่ายต่อการแก้ไขปัญหา)

หลังจากการพิสูจน์ตนสำเร็จลง เมสเสจจะถูกส่งผ่านไปยังบริการชั้นที่สูงกว่า

#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 (ไคลเอนต์สามารถส่งได้เพียงแมสเซจ `SSH_MSG_USERAUTH_REQUEST` เท่านั้น) วิธีการพิสูจน์ตนที่แตกต่างกันสามารถนำหมายเลขแมสเซจที่เหมือนกันมาใช้ใหม่ได้

8.4 วิธีการพิสูจน์ตนด้วยคีย์สาธารณะ : `publickey` (Public Key Authentication Method : `publickey`)

วิธีการพิสูจน์ตนที่จำเป็นต้องมีคือการพิสูจน์ตนแบบคีย์สาธารณะเท่านั้น ทุกการอิมพลิเมนต์ต้องรองรับวิธีนี้ อย่างไรก็ตาม ผู้ใช้ไม่ทุกคนที่ต้องการใช้คีย์สาธารณะ และนโยบายภายในมักไม่ต้องการใช้การพิสูจน์ตนแบบคีย์สาธารณะ

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

คีย์ส่วนตัวมักถูกเก็บไว้ในรูปแบบที่เข้ารหัสที่ไอสต์ไคลเอนต์ และผู้ใช้ต้องกรอกพาสเฟส (passphrase) ก่อน เพื่อให้ลายเซ็นสามารถถูกสร้างขึ้นได้ แม้ว่ามันจะไม่ได้ถูกเข้ารหัสไว้ กระบวนการลงลายเซ็นก็ใช้การคำนวณมาก เพื่อหลีกเลี่ยงกระบวนการและการโต้ตอบกับผู้ที่ไม่จำเป็น 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 โคลเอ็นต์อาจส่งลายเซ็นที่สร้างโดยใช้คีย์ส่วนตัว โคลเอ็นต์อาจส่งลายเซ็นโดยตรงก่อนโดยไม่มีการตรวจสอบก่อนว่าคีย์นั้นยอมรับได้หรือไม่ 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*

เมื่อเซิร์ฟเวอร์ได้รับแพสเซนจ์นี้ มันต้องตรวจสอบว่าคีย์ที่ให้นานี้ยอมรับได้สำหรับการพิสูจน์ตนหรือไม่ และถ้าได้เซิร์ฟเวอร์ต้องตรวจสอบว่าลายเซ็นถูกต้องหรือไม่

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

หมายเลขเมสเสจกำหนดคุณลักษณะเฉพาะวิธีการ (method-specific message) ถูกใช้โดยวิธีการ
การพิสูจน์ตนแบบคีย์สาธารณะดังนี้

```
/* Key-based */
```

```
#define SSH_MSG_USERAUTH_PK_OK    60
```

8.5 วิธีการพิสูจน์ตนแบบรหัสผ่าน : password (Password Authentication Method : password)

การพิสูจน์ตนแบบรหัสผ่านใช้แพ็กเก็ตข้างล่างนี้ สังเกตว่าเซิร์ฟเวอร์อาจจะร้องขอผู้ใช้ให้เปลี่ยน
รหัสผ่าน ทุกการมีปฏิสัมพันธ์ควรรองรับการพิสูจน์ตนแบบรหัสผ่าน

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

สังเกตว่ารหัสผ่านถูกเข้ารหัสแบบ ISO-10646 UTF-8 แล้วแต่เซิร์ฟเวอร์จะตีความหมายและ
ตรวจสอบกับฐานข้อมูลรหัสผ่านอย่างไร อย่างไรก็ตาม ถ้าไคลเอนต์อ่านรหัสผ่านในรูปแบบอื่น (เช่น
ISO 8859-1 (ISO Latin1)) มันต้องแปลงรหัสผ่านไปเป็น ISO-10646 UTF-8 ก่อนทำการส่ง และ
เซิร์ฟเวอร์ต้องแปลงรหัสผ่านไปเป็นรูปแบบที่ใช้สำหรับรหัสผ่านในระบบนั้น

สังเกตว่า แม้ว่ารหัสผ่านแบบข้อความที่อ่านได้ถูกส่งไปในแพ็กเก็ต แต่ทั้งแพ็กเก็ตก็ถูกเข้ารหัส
โดยชั้นทรานสปอร์ต ทั้งเซิร์ฟเวอร์และไคลเอนต์ควรตรวจสอบว่าชั้นทรานสปอร์ตนั้นให้ความไว้วางใจ
(confidentiality) ได้หรือไม่ (เช่น ใช้การเข้ารหัส) ถ้าไม่ให้ความไว้วางใจ (ไม่มีการเข้ารหัส), การพิสูจน์
ตนแบบรหัสผ่านควรถูกยกเลิกใช้ และหากไม่ให้ความไว้วางใจหรือไม่มี MAC การเปลี่ยนรหัสผ่านควร
ถูกยกเลิกใช้

โดยปกติเซิร์ฟเวอร์ตอบสนองแก่เมสเสจนี้ด้วยความสำเร็จหรือล้มเหลว อย่างไรก็ตาม
เซิร์ฟเวอร์อาจตอบสนองด้วย SSH_MSG_USERAUTH_PASSWD_CHANGEREQ

```
byte    SSH_MSG_USERAUTH_PASSWD_CHANGEREQ
string  prompt (ISO-10646 UTF-8)
string  language tag (as defined in [RFC1766])
```

ในกรณีนี้ โคลเอ็นด์ซอร์ฟแวร์ร้องขอรหัสผ่านใหม่จากผู้ใช้ และการส่งการร้องขอใหม่โดย
ใช้แมสเชงข้างล่างนี้ โคลเอ็นด์อาจส่งแมสเชงนี้แทนการร้องขอการพิสูจน์ตนแบบรหัสผ่านปกติ โดย
ปราศจากการถามจากเซิร์ฟเวอร์

```
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` ความหมายของแต่ละแมสเชงเป็น
ดังนี้

- `SSH_MSG_USERAUTH_SUCCESS` รหัสผ่านถูกเปลี่ยนแล้ว และการพิสูจน์ตนได้จบลงด้วย
ความสำเร็จ
- `SSH_MSG_USERAUTH_FAILURE with partial success` รหัสผ่านถูกเปลี่ยนแล้ว แต่การพิสูจน์ตน
เพิ่มยังคงจำเป็น
- `SSH_MSG_USERAUTH_FAILURE without partial success` รหัสผ่านยังไม่ได้ถูกเปลี่ยน เกิดจากไม่
รองรับการเปลี่ยนรหัสผ่าน หรือป้อนรหัสเก่าผิด สังเกตว่าถ้าเซิร์ฟเวอร์ได้ส่ง `SSH_MSG_USERAUTH_PASSWD_CHANGEREQ` แล้ว เราจะรู้ว่ามันรองรับการเปลี่ยนรหัสผ่าน
- `SSH_MSG_USERAUTH_CHANGEREQ` รหัสผ่านไม่ได้ถูกเปลี่ยน เพราะไม่สามารถยอมรับรหัส
ใหม่ได้ (เช่น ง่ายต่อการคาดเดาเกินไป)

หมายเลขแมสเชงระบุลักษณะเฉพาะวิธีการ (method-specific) ข้างล่างนี้ ถูกใช้โดยวิธีการพิสูจน์
ตนแบบรหัสผ่าน

```
#define SSH_MSG_USERAUTH_PASSWD_CHANGEREQ 60
```

8.6 การพิสูจน์ตนบนพื้นฐานของโฮสต์ (Host-Based Authentication : hostbased)

บางทีประสงค์ที่จะอนุญาตให้ทำการพิสูจน์ตนบนพื้นฐานของโฮสต์ที่ผู้ใช้ได้ใช้เข้ามา และชื่อผู้
ใช้บนเครื่องโฮสต์ทางไกล แม้ว่ารูปแบบการพิสูจน์ตนนี้ไม่เหมาะสมกับที่ที่ต้องการความปลอดภัยสูง

แต่มันก็เหมาะสมกับในสถานการณ์การ รูปแบบการพิสูจน์นี้เป็นตัวเลือก (OPTIONAL) เมื่อถูกใช้ควรต้องระมัดระวังเป็นพิเศษในการป้องกันคีย์โฮสต์ส่วนตัว (private host key) จากผู้อื่น

โคลเอนต์ร้องขอการพิสูจน์ตนรูปแบบนี้โดยส่งแพคเกจข้างล่างนี้ มันคล้ายกับรูปแบบการพิสูจน์ตน "rhost" และ "host.quiv" ใน UNIX เว้นแต่ไอดีของโฮสต์โคลเอนต์จะถูกตรวจสอบอย่างเข้มงวดมากกว่า

วิธีการนี้ทำงานโดยโคลเอนต์ส่งลายเซ็นที่สร้างด้วยคีย์ส่วนตัวของโฮสต์โคลเอนต์ ซึ่งเซิร์ฟเวอร์จะสามารถตรวจสอบได้ด้วยคีย์สาธารณะของโฮสต์นั้น เมื่อไอดีของโฮสต์โคลเอนต์ถูกก่อตั้งขึ้นการให้เอกสิทธิ์ (แต่จะไม่มีกรพิสูจน์ตนอีก) ถูกดำเนินการโดยตั้งอยู่บนพื้นฐานของชื่อผู้ใช้นเซิร์ฟเวอร์และโคลเอนต์ และชื่อโฮสต์โคลเอนต์

```
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
```

ชื่ออัลกอริทึมสาธารณะที่ใช้ใน 'public key algorithm for host key' ถูกนิยามในการกำหนดคุณลักษณะเฉพาะชั้นทรานสปอร์ต 'public host key for client host' อาจรวมเซอร์ทิฟิเกต (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)
```

เซิร์ฟเวอร์ต้องตรวจสอบว่าคีย์โฮสต์เป็นของโฮสต์โคลเอนต์ที่มีชื่ออยู่ในแมสเชจ ซึ่งเป็นโฮสต์โคลเอนต์ที่อนุญาตให้ผู้ใช้อีกอิน และลายเซ็นนั้นเป็นลายเซ็นที่ถูกต้องสอดคล้องกับคีย์โฮสต์ที่ให้ เซิร์ฟเวอร์อาจเพิกเฉยชื่อผู้ใช้นโคลเอนต์ถ้าต้องการพิสูจน์ตนเฉพาะโฮสต์โคลเอนต์

แนะนำว่าเมื่อใดก็ตามที่เป็นไปได้ เซิร์ฟเวอร์ดำเนินการตรวจสอบเพิ่มเพื่อตรวจสอบว่าแอดเดรสเครือข่าย (network address) ที่ได้จากเครือข่าย (ไม่น่าไว้วางใจ) ตรงกับชื่อโฮสต์โคลเอนต์ที่ได้ ทำให้การนำคีย์โฮสต์ที่ใช้ไม่ได้แล้วมาใช้ประโยชน์มีความยากมากขึ้น สังเกตว่ามันอาจต้องการการควบคุมเป็นพิเศษสำหรับการเชื่อมต่อที่เข้ามาผ่านไฟร์วอลล์ (firewall)

8.7 พิจารณาด้านความปลอดภัย

จุดประสงค์ของโพรโตคอลนี้คือเพื่อดำเนินการพิสูจน์ตนผู้ใช้นฝั่งโคลเอนต์ มันจะถือว่าทำงานอยู่บนโพรโตคอลชั้นทรานสปอร์ตที่ปลอดภัยซึ่งได้ผ่านการพิสูจน์ตนกับเครื่องเซิร์ฟเวอร์ ก่อตั้งแนวนลสื่อสารที่เข้ารหัส และคำนวณไอเคเอ็มดีฟายเออร์ของเซสชันที่เป็นเอกลักษณ์แก่เซสชันนี้ ชั้นทรานสปอร์ตให้การส่งผ่านที่เป็นความลับสำหรับการพิสูจน์ตนแบบรหัสผ่านและวิธีอื่นๆ ซึ่งเชื่อมั่นข้อมูลที่เป็นความลับ

เซิร์ฟเวอร์อาจเข้าสู่ช่วง "sleep" ได้ หลังจากการพิสูจน์ตนที่ไม่ประสบความสำเร็จติดต่อกันซ้ำๆ เพื่อให้การค้นหายากขึ้น

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

อนุญาตให้มีวิธีการพิสูจน์ตนหลายแบบหลายคุณลักษณะความปลอดภัยที่แตกต่างกันไป แล้วแต่นโยบายฝั่งโกลของแต่ละเซิร์ฟเวอร์ที่จะติดต่อกัน ในการตัดสินใจว่าวิธีการใด (หรือใช้หลายวิธีการประกอบกัน) ที่มันจะยอมรับสำหรับแต่ละผู้ใช้

ควรให้ความใส่ใจเป็นพิเศษเมื่อทำการออกแบบข้อความดีบัก ถ้าการออกแบบไม่เหมาะสม แมสเชจเหล่านี้อาจจะเปิดเผยข้อมูลอย่างน่าตกใจเกี่ยวกับโฮสต์ ข้อความดีบักสามารถถูกยกเลิกห้ามใช้งาน (ระหว่างช่วงการพิสูจน์ตนผู้ใช้) เมื่อต้องการความปลอดภัยสูง

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.

บทที่ 9

โพรโทคอลเชื่อมต่อเอสเอสเอช 2

9.1 แนะนำ

โพรโทคอลเชื่อมต่อถูกออกแบบมาให้ทำงานบนชั้นทรานสปอร์ต และโพรโทคอลการพิสูจน์ตนผู้ใช้ ทำให้เกิดเซสชันล็อกอินแบบอินเทอร์แอคทีฟ (interactive login sessions), การเอ็กซีคิวต์คำสั่งระยะไกล (remote execute of commands), การเชื่อมต่อทีซีพี/ไอพีแบบฟอร์เวิร์ด (forwarded TCP/IP connections) และการเชื่อมต่อ X11 แบบฟอร์เวิร์ด (forwarded X11 connections) ชื่อบริการสำหรับโพรโทคอลนี้ (หลังจากการพิสูจน์ตนผู้ใช้) เรียกว่า “ssh-connection”

9.2 การร้องขอทั่วไป (Global Requests)

การร้องขอหลายชนิดที่มีผลกระทบต่อสแตต (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) จะตอบสนองต่อเมสเสจ (message) นี้ด้วย SSH_MSG_REQUEST_SUCCESS, SSH_MSG_REQUEST_FAILURE, หรือ เมสเสจร้องการระบุต่อเนื่อง

byte SSH_MSG_REQUEST_SUCCESS

ถ้าผู้รับไม่สามารถแยกแยะหรือรองรับการร้องขอ จะตอบสนองด้วย SSH_MSG_REQUEST_FAILURE

byte SSH_MSG_REQUEST_FAILURE

9.3 กลไกของแชนเนล (Channel Mechanism)

ทุกเซสชันแบบเทอร์มินอล, การเชื่อมต่อแบบฟอร์เวิร์ด เป็นต้น เป็นแชนเนล ฟังใดฟังหนึ่งอาจเป็นผู้เปิดการเชื่อมต่อ หลายแชนเนลสามารถถูกมัลติเพลกซ์ (multiplex) เป็นการเชื่อมต่อเดียว (a single connection)

แชนเนลถูกบ่งชี้โดยหมายเลขของแต่ละฝั่ง หมายเลขที่อ้างสำหรับหนึ่งแชนเนลที่แต่ละฝั่งอาจไม่เหมือนกัน การร้องขอเพื่อเปิดแชนเนลหนึ่งนั้นจะบรรจุหมายเลขแชนเนลของผู้ส่ง แมสเซจอื่นๆที่เกี่ยวข้องกับแชนเนลของผู้รับจะบรรจุหมายเลขแชนเนลของผู้รับสำหรับแชนเนลนั้นๆ

แชนเนลเป็นการควบคุมการไหล จะไม่มีข้อมูลถูกส่งไปยังแชนเนลจนกว่าจะได้แมสเซจที่บ่งบอกว่าเนื้อที่วินโดว์ (window space) ว่าง

9.3.1 การเปิดแชนเนล (Opening 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)

ชนิดของแชนเนลได้อธิบายไว้ในสถาปัตยกรรมของเอสเอสเอสเอช 2 ด้วยกลไกขยายเพิ่มเติมที่คล้ายคลึงกัน โดย

- 'sender channel' เป็นตัวบ่งชี้ฝั่งโลคอลสำหรับแชนเนลซึ่งถูกใช้โดยผู้ส่งแมสเซจนี้
- 'initial window size' ระบุจำนวนไบนารีของข้อมูลในแชนเนลที่สามารถส่งให้แก่ผู้ส่งแมสเซจนี้
- '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’ เป็นหมายเลขแชนเนลที่ส่งโดยฝั่งรีโมท

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])

ถ้าผู้รับเมสเสจ SSH_MSG_CHANNEL_OPEN ไม่รองรับชนิดแชนเนลที่ระบุไว้ มันจะตอบ
สนองด้วย SSH_MSG_CHANNEL_OPEN_FAILURE โคลเอ็นต์อาจจะแสดงข่าวสารเพิ่มเติมนี้แก่ผู้ใช้
และถ้าเกิดขึ้น ซอร์ฟแวร์โคลเอ็นต์ควรจะป้องกันตามที่อธิบายไว้ในบทที่ 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
```

9.3.2 การถ่ายโอนข้อมูล (Data Transfer)

ขนาดวินโดว์บ่งบอกถึงจำนวนไบนารีที่อีกฝั่งสามารถส่งเพื่อไม่ให้มันต้องรอให้วินโดว์ถูกปรับ
เปลี่ยน ทั้ง2ฝ่ายใช้เมสเสจต่อไปนี้ในการปรับเปลี่ยนวินโดว์

```

byte    SSH_MSG_CHANNEL_WINDOW_ADJUST
uint32  recipient_channel
uint32  bytes_to_add

```

หลังจากผู้รับได้เมสเสจนี้แล้ว อาจจะส่งจำนวนไบนารีมากกว่าที่เคยได้รับอนุญาต ขนาดวินโดว์จะถูกเพิ่มขึ้น

การถ่ายโอนข้อมูลถูกกระทำด้วยเมสเสจชนิดนี้

```

byte    SSH_MSG_CHANNEL_DATA
uint32  recipient_channel
string  data

```

ขนาดสูงสุดของส่วนข้อมูล 'data' ที่อนุญาตคือขนาดวินโดว์ปัจจุบัน ขนาดวินโดว์ลดลงโดยข้อมูลที่ส่งมา ทั้งสองฝ่ายอาจจะเพิกเฉยข้อมูลพิเศษที่ส่งหลังจากวินโดว์ที่อนุญาตไว้ว่างเปล่า

ยิ่งไปกว่านั้น บางแชนเนลสามารถถ่ายโอนข้อมูลหลายชนิดได้ ตัวอย่างคือ stderr data จากเซชันอินเทอร์เน็ตแอฟ บางข้อมูลสามารถถูกส่งผ่านด้วยเมสเสจ `SSH_MSG_CHANNEL_EXTENDED_DATA` โดยใช้จำนวนเต็มเป็นตัวระบุชนิดข้อมูล ชนิดและความหมายขึ้นกับชนิดของแชนเนล

```

byte    SSH_MSG_CHANNEL_EXTENDED_DATA
uint32  recipient_channel
uint32  data_type_code
string  data

```

ข้อมูลที่ส่งด้วยเมสเสจนี้ จะใช้วินโดว์เดียวกับที่ใช้กับข้อมูลปกติ ในขณะนี้มีเพียงชนิดเดียวที่นิยามไว้

```
#define SSH_EXTENDED_DATA_STDERR 1
```

9.3.3 การปิดแชนเนล (Closing a Channel)

เมื่อฝ่ายใดจะไม่ส่งข้อมูลแก่แชนเนลอีกต่อไป มันควรส่ง `SSH_MSG_CHANNEL_EOF` มีรูปแบบดังนี้

```
byte    SSH_MSG_CHANNEL_EOF
```


uint32 recipient_channel

ไม่มีการตอบสนองโดยตรงแก่เมสเซนจ์นี อย่างไรก็ตาม แอปพลิเคชันอาจส่ง EOF ให้แก่อะไรก็ตามที่อยู่อีกฝั่งของแชนเนล สังเกตว่าแชนเนลยังคงเปิดอยู่หลังส่งเมสเซนจ์นี และข้อมูลยังคงสามารถส่งกลับมาได้ เมสเซนจ์นีไม่ใช่เนื้อที่วินโดว และสามารถส่งแม้ไม่มีเนื้อที่วินโดวเหลือก็ตาม

เมื่อฝ่ายใดฝ่ายหนึ่งประสงค์จะยุติ (terminate) แชนเนล มันจะส่ง `SSH_MSG_CHANNEL_CLOSE` เมื่อได้รับเมสเซนจ์นี ผู้รับต้องส่ง `SSH_MSG_CHANNEL_CLOSE` กลับไป เว้นแต่เป็นผู้ส่งเมสเซนจ์สำหรับแชนเนลนี้ไป

แชนเนลจะถือว่าปิดสำหรับแต่ละฝ่ายเมื่อได้ส่งและได้รับ `SSH_MSG_CHANNEL_CLOSE` แล้ว จากนั้นแล้วสามารถนำหมายเลขแชนเนลกลับมาใช้ใหม่ได้ แต่ละฝ่ายสามารถส่ง `SSH_MSG_CHANNEL_CLOSE` ได้โดยปราศจากการส่ง `SSH_MSG_CHANNEL_EOF`

รูปแบบของเมสเซนจ์เป็นดังนี้

byte SSH_MSG_CHANNEL_CLOSE

uint32 recipient_channel

เมสเซนจ์นีไม่ใช่เนื้อที่วินโดว และสามารถส่งแม้ไม่มีเนื้อที่วินโดวเหลือก็ตาม
แนะนำว่าถ้าเป็นไปได้ ทุกข้อมูลที่ส่งก่อนเมสเซนจ์นี้ถูกนำส่งไปยังจุดหมายได้อย่างแท้จริง

9.3.4 การร้องขอกำหนดลักษณะเฉพาะของแชนเนล (Channel-Specific Requests)

หลายชนิดของแชนเนลมีส่วนขยายซึ่งเป็นลักษณะเฉพาะของชนิดแชนเนลพิเศษ ตัวอย่างเช่น การร้องขอเทอร์มินอลเทียม `pty` (pseudo terminal) สำหรับแชนเนลอินเทอร์แอคทีฟ

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` หรือเมสเซนจ์ร้องขอกำหนดคุณลักษณะเฉพาะต่อเนื่อง ถ้าผู้รับเมสเซนจ์ `SSH_MSG_CHANNEL_OPEN` ไม่รองรับชนิดแชนเนลที่ระบุไว้ มันจะตอบสนองด้วย `SSH_MSG_CHANNEL_OPEN_FAILURE`

เมสเซนจ์นีไม่ใช่เนื้อที่วินโดว และสามารถส่งแม้ไม่มีเนื้อที่วินโดวเหลือก็ตาม 'request type' เป็นไคคอลสำหรับแต่ละชนิดแชนเนล

นั้น

ไคลเอ็นต์จะได้รับอนุญาตให้ส่งแพสเซจต่อไป โดยไม่ต้องรอคอยการตอบสนองของการร้องขอ

byte *SSH_MSG_CHANNEL_SUCCESS*

uint32 *recipient_channel*

byte *SSH_MSG_CHANNEL_FAILURE*

uint32 *recipient_channel*

แพสเซจทั้งสองนี้ไม่ใช่เนื้อที่วินโดว์ และสามารถส่งแม้ไม่มีเนื้อที่วินโดว์เหลือก็ตาม

9.4 เซสชันแบบอินเทอร์แอคทีฟ (Interactive Sessions)

เซสชันคือการเอ็กซีคิว (execution) โปรแกรมระยะไกล โปรแกรมอาจจะเป็นเชลล์ (shell), แอปพลิเคชัน, คำสั่งระบบ หรือระบบย่อยสำเร็จ (built-in subsystem) อาจจะมีหรือไม่มี tty ก็ได้ และอาจรวมถึงหรือไม่รวมถึงการฟอร์เวิร์ด X11 (X11 forwarding) เซสชันทั้งหมดสามารถทำงานด้วยตัวเองพร้อมกัน

9.4.1 การเปิดเซสชัน (Opening a Session)

เซสชันจะถูกเริ่มต้นโดยการส่งแพสเซจดังนี้

byte *SSH_MSG_CHANNEL_OPEN*

string *"session"*

uint32 *sender_channel*

uint32 *initial window size*

uint32 *maximum packet size*

การอิมพลีเมนต์ ไคลเอ็นต์ควรปฏิเสธทุกการร้องขอเปิดเซสชัน (session channel) เพื่อให้ยากขึ้นสำหรับเซิร์ฟเวอร์ที่ประสงค์ไม่ดี จะมาโจมตีไคลเอ็นต์

9.4.2 การร้องขอเทอร์มินอลเทียม (Requesting a Pseudo-Terminal)

เทอร์มินอลเทียมสามารถถูกจัดสรรสำหรับเซสชันได้โดยการส่งแพสเซจนี้

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 เลือกใช้มิติอักษร/แถว ก่อนเลือกใช้พิกเซล (เมื่อไม่เป็น 0) มิติในรูปพิกเซลบอกถึงพื้นที่ของวินโดว์ที่สามารถวาดได้ พารามิเตอร์มิติเป็นเพียงสนทศ

ไคลเอ็นต์ควรจะเพิกเฉยต่อการร้องขอ pty

9.4.3 การฟอร์เวิร์ด X11 (X11 Forwarding)

9.4.3.1 การร้องขอฟอร์เวิร์ด X11 (Requesting X11 Forwarding)

การฟอร์เวิร์ด X11 อาจจะถูกร้องขอแก่เซสชันโดยการส่งเมสเสจดังนี้

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

แนะนำให้ลูกค้าพิสูจน์ตน (authentication cookie) ที่ถูกส่งเป็นของปลอม (fake) ได้จากการสุ่ม และลูกค้านี้จะถูกตรวจสอบและแทนด้วยคีย์ที่จริงเมื่อการร้องขอการเชื่อมต่อได้รับแล้ว

การฟอร์เวิร์ดการเชื่อมต่อ X11 ควรหยุดเมื่อแชนเนลของเซสชันถูกปิดลง อย่างไรก็ตาม เมื่อแชนเนลของเซสชันถูกปิดลง การฟอร์เวิร์ดที่เปิดไว้แล้วควรจะไม่ต้องปิดอัตโนมัติ

ถ้า 'single connection' เป็น TRUE จะมีการเชื่อมต่อเดียวที่ถูกฟอร์เวิร์ด จะไม่มีการเชื่อมต่ออื่นถูกฟอร์เวิร์ดอีกหลังจากการเชื่อมต่อแรกได้ฟอร์เวิร์ด หรือหลังจากแขนเนลของเซสชันได้ปิดลง

'x11 authentication protocol' เป็นชื่อของวิธีการพิสูจน์ตน X11 เช่น "MIT-MAGIC-COOKIE-1" โพรโทคอลของ X อธิบายในเอกสารของ [SCHEIFLER]

9.4.3.2 แขนเนล X11 (X11 Channels)

แขนเนล X11 เปิดได้โดยการร้องขอเปิดแขนเนล แขนเนลที่ได้จะเป็นอิสระจากเซสชัน และการปิดแขนเนลของเซสชันจะไม่ปิดแขนเนล 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

การอิมพลีเมนต์ต้องปฏิเสธทุกการร้องขอเปิดแขนเนล X11 ถ้าไม่ร้องขอทำการฟอร์เวิร์ด X11 ก่อน

9.4.4 การผ่านค่าตัวแปรสภาพแวดล้อม (Environment Variable Passing)

ตัวแปรสภาพแวดล้อมอาจจะถูกส่งผ่านไปยัง เชลล์/คอมมานด์ (shell/command) เพื่อสตาร์ทในภายหลัง โดยปกติแต่ละเครื่องจะมีชุดกำหนดค่าเริ่มต้นของตัวแปร (a preconfigured set of variables) เนื่องจากการตั้งค่าตัวแปรแวดล้อมโดยไม่ควบคุมเป็นอันตรายอย่างยิ่ง แนะนำว่าการอิมพลีเมนต์อนุญาตให้ตั้งค่าเพียงตัวแปรซึ่งมีข้อบ่งบอกชัดเจนว่าอนุญาตให้ปรับแต่งได้

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


9.4.5 การสตาร์ทเชลล์หรือคอมมานด์ (Starting a Shell or a Command)

เมื่อเซสชันได้ถูกคิดตั้งแล้ว โปรแกรมจะถูกสตาร์ทที่ฝั่งรีโมท โปรแกรมสามารถเป็นเชลล์, แอปพลิเคชัน หรือระบบย่อยกับชื่อที่อิสระจากโฮสต์ (a subsystem with a host-independent name) นี่เพียงหนึ่งในการร้องขอเหล่านี้ที่ทำให้สำเร็จต่อเซสชัน

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

แมสเชนนี้จะร้องขอเชลล์ที่เป็นดีฟอลต์ของผู้ใช้ (ในระบบ UNIX โดยทั่วไปจะนิยามไว้ใน /etc/passwd) เพื่อสตาร์ทที่อีกฝั่ง

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

แมสเชนนี้จะร้องขอให้เซิร์ฟเวอร์สตาร์ทการเอ็กซีคิวคอมมานด์ 'command' ที่ให้ สตริงคำสั่ง อาจจะมีพาท (path) ต้องทำการป้องกันการเอ็กซีคิวคำสั่งที่ละเมิดสิทธิ์

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

รูปแบบสุดท้ายนี้ใช้เอ็กซีคิวระบบย่อยที่นิยามไว้ ซึ่งประกอบด้วยกลไกถ่ายโอนเพิ่มข้อมูลและอื่นๆไว้ การอิมพลิเมนต์อาจจะอนุญาตให้ปรับแต่งกลไกต่างๆด้วย แนะนำให้โพรโตคอลระบบย่อยมีเมจิกคุกกี้ (magic cookie) ตั้งแต่เริ่มต้นการทราฟฟิกของโพรโตคอล เพื่อให้สามารถแยกแยะจากเอาท์พุทของสคริปต์เริ่มต้นเชลล์ได้ เอาท์พุทจากเชลล์อาจจะถูกกรองออกที่เซิร์ฟเวอร์หรือไคลเอนต์

เซิร์ฟเวอร์ควรจะหยุด (halt) การเอ็กซีคิวชันของสแตคโปรโตคอลในขณะที่สแตคเซลล์หรือโปรแกรม อินพุตและเอาต์พุตทั้งหมดควรถูกเปลี่ยนทิศทางใหม่ (redirect) ไปยังแชนเนลหรือไปยังทันเนล (tunnel) ที่เข้ารหัส

สำหรับแอสเซมบลีเหล่านี้แนะนำให้ร้องขอและตรวจสอบการตอบกลับ โคลเอ็นต์ควรจะเพิกเฉยต่อแอสเซมบลีเหล่านี้

9.4.6 การถ่ายโอนข้อมูลบนเซสชัน (Session Data Transfer)

การถ่ายโอนข้อมูลสำหรับเซสชันถูกทำโดยใช้แพ็คเกจ *SSH_MSG_CHANNEL_DATA* และแพ็คเกจ *SSH_MSG_CHANNEL_EXTENDED_DATA* และกลไกวินโดว์ ชนิดข้อมูลส่วนขยาย *SSH_EXTENDED_DATA_STDERR* ถูกนิยามไว้สำหรับข้อมูล *stderr*

9.4.7 แอสเซมบลีเปลี่ยนขนาดของวินโดว์ (Window Dimension Change Message)

เมื่อขนาดวินโดว์ (เทอร์มินอล) เปลี่ยนบนฝั่งโคลเอ็นต์ มันอาจจะส่งแอสเซมบลีไปยังอีกฝั่งเพื่อแจ้งขนาดใหม่ รูปแบบแอสเซมบลีเป็นดังนี้

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*

ควรไม่ส่งการตอบสนองต่อแอสเซมบลีนี้

9.4.8 โฟลว์คอนโทรลฝั่งโลคอล (Local Flow Control)

ในหลายๆระบบ เป็นไปได้ที่จะกำหนดว่าเทอร์มินอลใช้โฟลว์คอนโทรล *control-S/control-Q* หากได้รับอนุญาตให้ใช้โฟลว์คอนโทรลได้ บ่อยครั้งที่มีความต้องการใช้โฟลว์คอนโทรลที่ฝั่งโคลเอ็นต์ เพื่อเพิ่มความเร็วของการตอบสนองต่อการร้องขอของผู้ใช้ มันถูกทำให้สะดวกโดยมีขั้นตอนคือ เริ่มต้นด้วยเซิร์ฟเวอร์จะรับผิชอบเรื่องโฟลว์คอนโทรล (ต่อไปให้เข้าใจว่า โคลเอ็นต์หมายถึงฝั่งที่เริ่มก่อนการสร้างเซสชัน และเซิร์ฟเวอร์หมายถึงอีกฝั่ง)

แพคเกจข้างล่างนี้ถูกใช้โดยเซิร์ฟเวอร์ เพื่อแจ้งแก่ไคลเอ็นต์ว่าเมื่อใดไคลเอ็นต์สามารถใช้หรือไม่สามารถใช้โฟลวคอนโทรลได้ ถ้า 'client can do' เป็น TRUE ไคลเอ็นต์จะได้รับอนุญาตให้ทำโฟลวคอนโทรลโดยใช้ control-S และ control-Q ไคลเอ็นต์อาจเพิกเฉยต่อแพคเกจนี้

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

จะไม่มีการส่งแพคเกจตอบสนองต่อแพคเกจนี้

9.4.9 ซิกแนล (Signals)

ซิกแนลสามารถถูกส่งไปยังกระบวนการหรือบริการบนฝั่งรีโมท (remote process/service) โดยใช้แพคเกจดังนี้

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

บางระบบอาจจะไม่อิมพลีเมนต์ซิกแนล ในกรณีเช่นนี้ควรเพิกเฉยต่อแพคเกจนี้ 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' ชื่อซิกแนลที่ไม่เป็นมาตรฐานจะพบว่าจะอยู่ในรูปแบบ "SIG@xyz.config.guess" โดย 'SIG' เป็นชื่อซิกแนลซึ่งไม่มี 'SIG' นำหน้า และ 'xyz' เป็นชนิดโฮสต์ ตามที่กำหนดไว้ใน 'config.guess'

9.4.10 การคืนค่าสถานะจบ (Returning Exit Status)

เมื่อคำสั่งที่ทำงานบนฝั่งตรงข้ามจบลง เมสเสจต่อไปนี้ใช้ส่งเพื่อคืนค่าการจบของคำสั่ง แนะนำว่าการคืนค่าสถานะจะไม่มี การแจ้งการได้รับ (acknowledgement) เมสเสจนี้ แชนเนลจำเป็นต้องปิดด้วย `SSH_MSG_CHANNEL_CLOSE` หลังเมสเสจนี้ และไคลเอ็นต์อาจจะเพิกเฉยเมสเสจนี้

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

คำสั่งที่ทำงานบนรีโมทอาจจบลงทันทีด้วยเมสเสจนี้
ในบางสถานการณ์ อาจใช้เมสเสจข้างล่างนี้ ซึ่งค่าที่คืนเมื่อจบการทำงานด้วยความสำเร็จนั้น โดยปกติจะให้ป็นศูนย์

```
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 ซอร์ฟแวร์ไคลเอ็นต์อาจจะแสดงข้อความนี้แก่ผู้ใช้ หากเคยเกิดขึ้นแล้ว ซอร์ฟแวร์ไคลเอ็นต์ควรมีการป้องกันเช่นอธิบายไว้ในบทที่ 6

9.5 การฟอร์เวิร์ดพอร์ตที่ซีพี/ไอพี (TCP/IP Port Forwarding)

9.5.1 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 โคลเอ็นต์ยังคงสามารถคัดสรรการเชื่อมต่อบนพื้นฐานของข้อมูลที่ส่งผ่านมากับการร้องขอเปิด)

การอิมพลีเมนต์ควรอนุญาตให้ทำการฟอร์เวิร์ดพอร์ตที่มีสิทธิ์ต่อเมื่อผู้ใช้ได้ผ่านการพิสูจน์ตนว่าเป็นผู้ใช้ที่มีสิทธิ์แล้ว

การอิมพลีเมนต์โคลเอ็นต์ควรปฏิเสธเมสเซจเหล่านี้เพราะโดยปกติแล้วมันจะถูกส่งมาจากโคลเอ็นต์เท่านั้น

การฟอร์เวิร์ดพอร์ตสามารถยกเลิกได้ด้วยเมสเซจข้างล่างนี้ สังเกตว่า การร้องขอเปิดแชนเนลอาจได้รับจนกระทั่งการตอบกลับค่อเมสเซจนี้ได้รับแล้ว

```

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

```

การอิมพลีเมนต์โคลเอ็นต์ควรปฏิเสธเมสเซจเหล่านี้เพราะโดยปกติแล้วมันจะถูกส่งมาจากโคลเอ็นต์เท่านั้น

9.5.2 แชนเนลฟอร์เวิร์ดที่ซีพี/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*

การอิมพลีเมนต์การปฏิเสธแมสเซนจ์ เว้นแต่ว่าก่อนหน้านี้ได้ร้องขอทำการฟอร์เวิร์ดพอร์ตที่ซีพี/ไอพีที่พอร์ตที่ต้องการแล้ว

เมื่อการเชื่อมต่อมาถึงที่พอร์ตที่ซีพี/ไอพีที่ฟอร์เวิร์ดบนโลคอล แพ็กเก็ตต่อไปนี้จะถูกส่งไปยังฝั่งตรงข้าม สังเกตว่าแมสเซนจ์เหล่านี้จะถูกส่งแก่พอร์ตซึ่งไม่เคยร้องขอทำการฟอร์เวิร์ด 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

แชนเนลที่ซีพี/ไอพีที่ฟอร์เวิร์ดแล้วจะเป็นอิสระต่อเซสชันใดๆ และการปิดลงของเซสชันแชนเนลไม่ทำให้การเชื่อมต่อที่ฟอร์เวิร์ดไว้ถูกปิดลง

การอิมพลีเมนต์ส่วนไคลเอ็นต์ควรปฏิเสธการร้องขอการเปิดที่ซีพี/ไอพีโดยตรงด้วยเหตุผลด้านความปลอดภัย

9.6 การเข้ารูปแบบโหมดของเทอร์มินอล (Encoding of Terminal Modes)

โหมดของเทอร์มินอล (ที่มากับการร้องขอเทอร์มินอลเทียม pty) ถูกเข้ารูปแบบเป็นสตรีมของไบต์ (byte stream) โดยมุ่งหมายให้ส่งผ่านข้ามเครือข่ายที่ต่างกันได้

คำอธิบายโหมคของ tty เป็นสตริงของไบต์ สตริงประกอบขึ้นด้วยกลุ่มของ ออปโค้ด-อักขระเมนต์ ซึ่งจะปิดท้ายด้วยออปโค้ด TTY_OP_END ออปโค้ดที่ 1 ถึง 159 มีอักขระเมนต์ uint32 หนึ่งค่า ส่วนออปโค้ด 160 ถึง 255 ยังไม่ได้นิยามไว้และทำให้การพาร์ (parsing) หยุคลง

ไคลเอ็นต์ควรจะได้ทุกโหมคที่รู้จักลงในสตริง และเซิร์ฟเวอร์เองอาจเพิกเฉยโหมคที่ไม่รู้จัก ทำให้เกิดความไม่ขึ้นกับเครื่อง โดยอย่างน้อยที่สุดทั้งสองระบบใช้อินเทอร์เฟซของ tty แบบคล้าย POSIX (POSIX-like)

ออปโค้ดข้างล่างนี้ได้ถูกนิยามไว้ ชื่อของออปโค้ดเป็นไปตามแฟลกโหมคของเทอร์มินอลแบบ 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 'X'.
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.

9.7 สรุปหมายเลขหมายเลขเมสเสจ

#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

9.8 พิจารณาด้านความปลอดภัย

โพรโตคอลนี้จะถือว่าได้ทำงานอยู่บนการลำเลียงที่ผ่านการพิสูจน์ตนที่มีความปลอดภัย โพรโตคอลชั้นล่างทำให้เกิดการพิสูจน์ตนผู้ใช้และการป้องกันต่อการโจมตีระดับเน็ตเวิร์ก (network level attack)

อย่างไรก็ตาม โพรโตคอลนี้สามารถถูกใช้ในการเอ็กซ์คิวกอมมานด์บนเครื่องรีโมท โพรโตคอลยังอนุญาตให้เซิร์ฟเวอร์รันคอมมานด์บนไคลเอ็นต์อีกด้วย การอิมพลีเมนต์อาจประสงค์ให้ไม่อนุญาตให้ทำเช่นนี้ เพื่อป้องกันผู้โจมตีที่มากับเซิร์ฟเวอร์เข้าโจมตีไคลเอ็นต์

การฟอร์เวิร์ด X11 ทำให้เกิดการปรับปรุงด้านความปลอดภัยหลักจากการฟอร์เวิร์ด X11 ที่อยู่บนพื้นฐานของคูกี้ปกติ คูกี้ไม่เคยจำเป็นต้องส่งโดยไม่เข้ารหัส การสื่อสารถูกเข้ารหัสและปกป้องความสมบูรณ์ของข้อมูล จะไม่มีข้อมูลพิสูจน์ตนที่มีประโยชน์เหลืออยู่บนเครื่องเซิร์ฟเวอร์หลังจากการเชื่อมต่อถูกปิดลง ในทางตรงกันข้าม ในบางสถานการณ์การเชื่อมต่อ X11 ที่ทำการฟอร์เวิร์ดอาจถูกใช้เพื่อเข้าถึงเซิร์ฟเวอร์ X ที่ฝั่งไคลเอนต์ผ่านขอบเขตความปลอดภัยได้

การฟอร์เวิร์ดพอร์ตสามารถเป็นการอนุญาตให้ผู้ประสงค์ร้ายข้ามผ่านขอบเขตความปลอดภัย อย่างเช่นไฟร์วอลล์เข้ามาได้ มันไม่ควรยอมให้ผู้ใช้ทำในสิ่งที่ไม่ควรทำ อย่างไรก็ตามมันทำให้การเปิดทันเนลง่ายมาก การอิมพลีเมนต์ควรอนุญาตให้ควบคุมนโยบายบนบริการที่สามารถถูกฟอร์เวิร์ดได้ ผู้ดูแลระบบควรสามารถที่จะปฏิเสธการฟอร์เวิร์ดตามความสมควร

เนื่องจากโพรโตคอลนี้ทำงานอยู่ภายในทันเนลที่เข้ารหัส ไฟร์วอลล์จะไม่สามารถตรวจสอบการส่งผ่านนี้ได้

ถ้าคีย์โฮสต์ถูกเปลี่ยนแปลง แนะนำว่าการอิมพลีเมนต์ควรยกเลิกการใช้งานที่อาจนำซึ่งอันตราย เช่น การฟอร์เวิร์ดอเจนต์, 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/IEEE 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, February 1992.



บทที่ 10

ภาษาจาวา

10.1 ประวัติของภาษาจาวา

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

ภาษาจาวาเริ่มเป็นที่สนใจ เมื่อบริษัท ซัน ไมโครซิสเต็มส์ (Sun Microsystems) ประกาศให้เป็นภาษาสำหรับสร้างโปรแกรมเพื่อใช้งานอินเทอร์เน็ต (Internet) โปรแกรมภาษาจาวาอาจถูกสร้างขึ้นบนคอมพิวเตอร์เครื่องหนึ่ง แล้วนำไปทำงานบนเครื่องคอมพิวเตอร์ต่างระบบได้ โดยไม่ต้องคอมไพล์โปรแกรมใหม่ ความสามารถนี้จะเปลี่ยนโฉมหน้าและบทบาทของอินเทอร์เน็ตอย่างสิ้นเชิง เมื่อเรามีโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ใดๆ ได้ อินเทอร์เน็ตจะกลายเป็นหนึ่งเดียว สิ่งที่ถูกส่งไปมาในอินเทอร์เน็ตไม่จำเป็นต้องแบ่งแยกอีกต่อไป นั่นคือเว็บเพจทั้งหลายจะไม่ใช่ออกสารที่ถูกกระทำ (Passive) แต่จะกลายเป็นเอกสารที่ทำงานได้ (Active) นอกจากนี้ภาษาจาวาจะส่งผลกระทบต่อทั้งผู้ผลิตและผู้ใช้ คือถึงเวลาที่บริษัทผู้ผลิตซอฟต์แวร์ทั้งหลายจะต้องเปลี่ยนแปลง หรือยุบแผนกวินโดวส์, แมคอินทอชและยูนิกซ์ มารวมกันเสียทีเพื่อผลิตโปรแกรมที่ทำงานบนระบบใดๆ ก็ได้เพียงหนึ่งเดียว สำหรับผู้ใช้ลองนึกดูว่า ถ้าคอมพิวเตอร์ของเราเมื่อเปิดเครื่องเข้าสู่บราวเซอร์ จากนั้นเราสามารถเรียกโปรแกรมที่ต้องการจากเซิร์ฟเวอร์ที่ให้บริการในอินเทอร์เน็ตมาใช้งาน รูปแบบการทำงานของเราก็จะไม่เหมือนเดิมอีกต่อไป นั่นคือไม่จำเป็นต้องมีฮาร์ดดิสก์ (Hard disk) หรือแม้แต่ระบบปฏิบัติการอย่างวินโดวส์เสียก็ได้

ผลกระทบเหล่านี้ทำให้ภาษาจาวาเป็นที่น่าสนใจอย่างยิ่ง ในขณะที่บริษัทผู้ผลิตซอฟต์แวร์ทั้งหลายก็ให้การต้อนรับภาษานี้ในทิศทางที่ดีเกินความคาดหมาย ประกอบด้วยภาษาจาวามีประสิทธิภาพในการสร้างโปรแกรมอย่างยิ่ง จึงเป็นเครื่องมือชั้นดีสำหรับเทคโนโลยีการสร้างโปรแกรมด้วยคอมพิวเตอร์ และวิชาวลโปรแกรมมิ่ง เชื่อได้แน่นอนว่าในอนาคตอันใกล้นี้ อุตสาหกรรมซอฟต์แวร์จะเปลี่ยนโฉมหน้าไปจากเดิมและการเรียนรู้ภาษาจาวาจะเป็นเรื่องที่หลีกเลี่ยงไม่ได้

ในช่วงต้น 1990s ตลาดเครื่องใช้ไฟฟ้า เช่นเครื่องซักผ้า, หม้อหุงข้าว, โทรทัศน์, ไมโครเวฟและอื่นๆ มีมูลค่าสูงกว่าตลาดคอมพิวเตอร์หลายเท่าตัว อีกทั้งระบบคอมพิวเตอร์ขนาดเล็กสำหรับควบคุมเครื่องใช้ไฟฟ้าเหล่านี้ก็ถูกพัฒนาดีขึ้นเรื่อยๆ บริษัท ซัน ไมโครซิสเต็มส์ ซึ่งประสบความสำเร็จในตลาด

ระบบเครือข่ายคอมพิวเตอร์อย่างมากในตอนนั้น เห็นว่าควรใช้ความได้เปรียบของบริษัท เจริญพัฒนาเทคโนโลยีเพื่อยึดครองตลาดเครื่องใช้ไฟฟ้าที่มีคอมพิวเตอร์ควบคุมไว้ก่อนคนอื่น จึงจัดทีม Green group ขึ้นในปี 1991 มี James Gosling เป็นหัวหน้า ทำการพัฒนาระบบซอฟต์แวร์สำหรับควบคุมเครื่องใช้ไฟฟ้าขนาดเล็ก พวกเขาสร้างเครื่องต้นแบบที่เรียกว่า Star 7 เป็นระบบรีโมตคอลโทรลขนาดเล็กมีถ่าน สามารถควบคุมการเคลื่อนไหวกว้างวัตถุโดยใช้รีโมตสัมผัสแบบเป็นแสงผล ระบบนี้ถูกทดสอบใช้การแสดงควบคุมตัว The Duke (ซึ่งต่อมาเป็นตัวนำโชคของภาพยนตร์) ใช้เคลื่อนที่ผ่านกลุ่มของวัตถุในมิติสมมติ

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

ปัญหาที่สำคัญกว่าคือ หน่วยประมวลผลที่ใช้ในงานควบคุมมีหลากหลายเบอร์หลายยี่ห้อ และมีชุดคำสั่งไม่เหมือนกัน โปรแกรมที่ทำงานได้บนหน่วยประมวลผลรุ่นหนึ่งจะต้องถูกคอมไพล์ใหม่ จึงจะนำไปใช้งานบนหน่วยประมวลผลอีกรุ่นหนึ่งได้ ด้วยเหตุนี้พวกเขาจึงพัฒนาภาษาใหม่ชื่อโอ๊ก (ได้ชื่อจากต้นไม้โอ๊กที่อยู่ข้างตึกที่ทำงาน) ให้เป็นภาษาที่ง่ายต่อการเรียนรู้และใช้งาน ไม่มีข้อผิดพลาดในตอนทำงาน (bug-free) และเหมาะที่จะทำงานในระบบที่มีหน่วยความจำน้อย พวกเขาแน่ใจว่าระบบควบคุมจะมีขนาดใหญ่และซับซ้อนขึ้นเรื่อยๆ จึงให้โอ๊กเป็นภาษาเชิงวัตถุ เพื่อง่ายในการสร้างและดูแลระบบขนาดใหญ่ ปัญหาใหญ่ของการออกแบบภาษาโอ๊กอยู่ที่ต้องการให้เป็นภาษาที่ทำงานบนหน่วยประมวลผลใดๆ ก็ได้ จึงนำเทคนิคการคอมไพล์โปรแกรมเป็นคำสั่งของหน่วยประมวลผลสมมติตัวหนึ่ง แล้วสร้างอินเทอร์พรีเตอร์ของหน่วยประมวลผลสมมติตัวนั้นให้แก่หน่วยประมวลผลที่จะทำงานโปรแกรมนั้น ด้วยวิธีนี้ โปรแกรมที่สร้างขึ้นจึงสามารถนำไปทำงานบนเครื่องที่มีหน่วยประมวลผลต่างรุ่นได้ เราเรียกคุณสมบัติอย่างนี้ว่า platform independent

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

ในปี 1993 เมื่อ ไสเพอร์เท็กซ์และบราวเซอร์ เปลี่ยนแปลงอินเทอร์เน็ตไปอย่างมากและมีผู้ใช้งานขึ้นอย่างรวดเร็ว บริษัท ซัน ไมโครซิสเต็มส์ มองเห็นความจำเป็นที่ต้องมีภาษาสำหรับสร้างโปรแกรมที่สามารถทำงานบนเครื่องคอมพิวเตอร์ใดๆก็ได้ และเพ็งนึกถึงคุณสมบัติ platform independent ของภาษาโอ๊ก จึงนำมาปรับปรุงใหม่ และทดลองสร้างเว็บเบราว์เซอร์ชื่อเว็บรันเนอร์ที่ทำงานโปรแกรมภาษาโอ๊กได้ เมื่อทดลองจนได้ผล พวกเขาทราบว่าสิ่งสำคัญอย่างยิ่งในมือแล้ว แต่โอ๊กเป็นชื่อทางการค้าที่มีผู้ใช้อยู่ก่อน จึงเปลี่ยนเป็นจาวาในตอนต้นปี 1995 พร้อมกับเปลี่ยนชื่อเว็บรันเนอร์เป็น ฮอตจาวา (HotJava)

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

10.2 คุณสมบัติของภาษาจาวา

ภาษาจาวาถูกออกแบบให้มีลักษณะดังต่อไปนี้

1. เป็นภาษาที่ง่าย (Simple) ในการเรียนรู้และใช้งาน ความหมายของคำว่า “ง่าย” อาจจะถูกตีความหมายได้หลายอย่างในแง่ต่างๆ ต่อไปนี้
 - ภาษาจาวานำไวยากรณ์ภาษาส่วนใหญ่มาจากภาษาซีและซีพลัสพลัส ผู้ที่คุ้นเคยกับภาษาซีหรือซีพลัสพลัสอยู่แล้ว จะเข้าใจไวยากรณ์ภาษาจาวาได้ง่าย หรือใช้เวลาศึกษาไม่นานนัก
 - ภาษาจาวามีกลไกของภาษาจำนวนไม่มากและไม่ซับซ้อนโดยตัดกลไกของภาษาซีและซีพลัสพลัสส่วนที่ทำให้ภาษายุ่งยากออกไปอย่างเช่น pointer arithmetic, default argument, scope resolution, protected and private inheritance และ operator overloading แต่ในขณะเดียวกันก็เพิ่มความสามารถให้คอมไพเลอร์ของภาษาจาวา ทำให้ไม่มี preprocessor commands ดังนั้นจะไม่มี macros definitions, included files, conditional compilation และ header files เมื่อจาวาเป็นภาษาเชิงวัตถุแล้ว กลไกอย่างเช่น structures, unions, bit fields และ enumerated types รวมทั้งการทำ typedef ก็ไม่มีความจำเป็นจึงถูกตัดออกไป ภาษาจาวาถูกออกแบบให้เป็นภาษาเชิงวัตถุอย่างรอบคอบกว่าภาษาซีพลัสพลัส ดังจะเห็นได้ว่ากลไกที่อาจทำให้เกิดความกำกวมอย่างเช่น multiple inheritance และ copy constructor และเป็นกลไกที่อาจจะทำลายแบบแผนการเขียนโปรแกรมเชิงวัตถุที่ดี ไป และ friend methods ก็ถูกตัดออกไปด้วย
 - ภาษาจาวาประสบความสำเร็จอย่างมาก ด้วยการใช้เทคนิคภาษาเชิงวัตถุจึงช่วยสร้างโปรแกรมที่ยุ่งยากให้เป็นไปได้ง่ายขึ้น ดังจะเห็นได้ว่าหากเป็นภาษาอื่น การสร้างโปรแกรมที่เกี่ยวกับ graphic user interfaces, multitasking, network และ distributed objects ผู้ใช้โปรแกรมจะต้องมีความรู้ขั้นสูงจึงจะทำงานได้ หรือไม่ก็ต้องอาศัยโปรแกรมประเภทวิชวลโค้ดอย่างในวิชวลซีพลัสพลัสช่วยสร้างโปรแกรม แต่ภาษาจาวามีกลไกของโปรแกรมเชิงวัตถุที่ส่งเสริมการนำโปรแกรมที่สร้างไว้แล้วนำมาใช้งานใหม่ (reuse) โดยสามารถเปลี่ยนแปลงหรือเพิ่มเติมโปรแกรมบางส่วน ให้เหมาะกับงานที่ต้องการได้โดย

ไม่จำเป็นต้องทราบรายละเอียดของการสร้างโปรแกรมเดิมทั้งหมด 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. โปรแกรมที่สร้างขึ้นด้วยภาษาจาวาจะไม่มีผลผลิตจากข้อบกพร่องของภาษา นั่นคือโปรแกรมจะต้องไม่ล้มเหลวลงด้วยความผิดปกติเพียงเล็กน้อยที่ไม่เกี่ยวกับตรรกะของโปรแกรม คุณสมบัติของภาษาอย่างนี้เรียกว่า ความคงทน (Robust) ภาษาจาวาถูกออกแบบให้มีความคงทนด้วยวิธีการดังนี้
- ภาษาจาวาเน้นการใช้กลไก exception handling เพื่อให้โปรแกรมสามารถจัดการกับความผิดปกติบางอย่างที่เกิดขึ้นในขณะที่โปรแกรมทำงานให้โปรแกรมทำงานต่อไปได้โดยไม่ต้องหยุดลง
 - ภาษาจาวาตัดกลไกบางอย่างในภาษาซีและซีพลัสพลัส ที่อาจจะทำให้เกิดความผิดพลาด หากใช้ไม่ระวัง เช่น global variables, variable length arguments และ goto statement ภาษาจาวาตัดการอ้างถึงแอดเดรสของตัวแปร และการใช้พอยน์เตอร์สำหรับอ่านหรือเขียนข้อมูลในหน่วยความจำโดยตรง
 - ภาษาจาวาไม่มีกลไกสำหรับคืน (de-allocation) หน่วยความจำที่ขอมมาในขณะที่โปรแกรมทำงาน (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) อย่างไรก็ดีตาม ไม่มีภาษาใดที่ปลอดภัยร้อยเปอร์เซ็นต์ ภาษาจาวาถูกจัดว่ามีความปลอดภัย

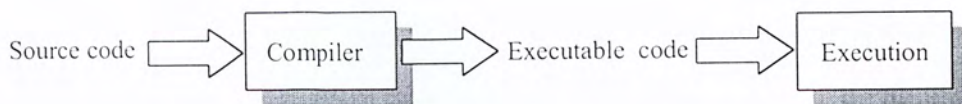
ในระดับสูงเท่านั้น เพราะถูกออกแบบมาเพื่อความปลอดภัยมากกว่าภาษาอื่น โดยแบ่งการป้องกันออกเป็นหลายระดับดังนี้

- ดังที่กล่าวมาแล้วว่า ภาษาจาวาไม่ยอมให้อ้างถึงค่าในหน่วยความจำผ่านทางพอยน์เตอร์และจะทำการตรวจสอบว่าการอ้างถึงสมาชิกในอาร์เรย์อยู่ในขอบเขตหรือไม่ โปรแกรมจึงไม่สามารถเขียนหรืออ่านค่าในหน่วยความจำที่ไม่มีสิทธิ์อ้างถึง การเปลี่ยนแปลงโปรแกรมหรือค่าในหน่วยความจำ เพื่อสร้างโปรแกรมที่จะเป็นอันตรายต่อคนอื่นด้วยวิธีนี้จึงเกิดขึ้นไม่ได้
 - จาวาอินเทอร์พรีเตอร์ มี byte-code verifier ทำหน้าที่ตรวจสอบโปรแกรมที่จะถูกทำงานว่ามีคำสั่งที่ผิดปกติ หรือมีการทำงานที่ไม่สมควรหรือไม่ หากพบก็จะถูกปฏิเสธที่จะทำงานโปรแกรมนั้น
 - ภาษาจาวามีระบบรักษาความปลอดภัยที่เรียกว่า sandbox model นั่นคือโปรแกรมที่ถูกนำมาจากเครื่องอื่นผ่านระบบเครือข่ายจะถือว่าเป็นโปรแกรมที่ไม่น่าไว้วางใจ (un – trusted codes) และถูกเก็บอยู่ในภาชนะที่เรียกว่า sandbox โปรแกรมที่อยู่ใน sandbox จะมีข้อจำกัดในการทำงานหลายอย่าง ซึ่งถูกควบคุมโดย security manager เช่น ไม่สามารถอ่านหรือเขียนไฟล์ได้ เป็นต้น
4. จุดมุ่งหมายสำคัญของการออกแบบภาษาจาวา คือ โปรแกรมต้องสามารถทำงานบนเครื่องต่างระบบกันได้ เรียกคุณสมบัติภาษาอย่างนี้ว่า “ไม่ขึ้นกับระบบ” (architecture neutral หรือ platform independent) เพื่อให้ได้คุณสมบัตินี้ ภาษาจาวาต้องมีการแปลภาษาแบบทั้ง Compilation และ interpretation รวมทั้งการกำหนดเครื่องจักรสมมติของจาวา (Java Virtual Machine) ดังจะกล่าวต่อไปนี้

10.3 คอมไพเลอร์ และอินเทอร์พรีเตอร์ (Compilation and interpretation)

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

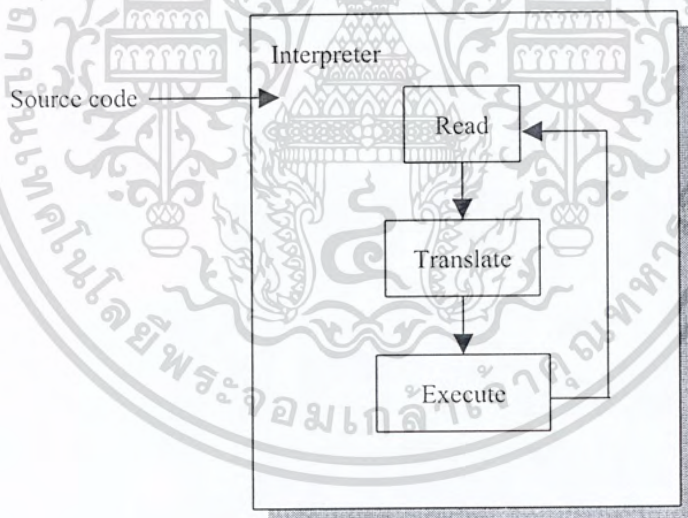
- คอมไพเลอร์ ตัวแปลภาษาในกรณีนี้เรียกว่าคอมไพเลอร์ ทำหน้าที่วิเคราะห์โปรแกรมต้นฉบับแล้วสร้างเอ็กซิกิวต์เอเบิลโค้ดเป็นผลลัพธ์การกระทำดังกล่าวเรียกว่า การคอมไพล์โปรแกรม สรุปได้ดังนี้



รูปที่ 10-1 แสดงการการคอมไพล์โปรแกรม

ข้อดีของวิธีนี้คือ เอ็กซีคิวต์เอเบิลโค้ดทำงานได้เร็วมาก เนื่องจากขั้นตอนในการแปลภาษาถูกแยกออกไปทำก่อนโปรแกรมทำงาน และคอมไพเลอร์ สามารถวิเคราะห์โปรแกรมได้ทั้งโปรแกรม จึงสามารถทำการ optimization ให้โปรแกรมมีประสิทธิภาพ ในการทำงานและมีขนาดเล็ก ตัวอย่างของภาษาที่ใช้การแปลภาษาแบบคอมไพล์เช่นคือ ฟอแทรน, อัลกอล, ปาสคาล, ซี และ ซีพลัสพลัส

- อินเทอร์พรีเทชัน ตัวแปลภาษาในกรณีนี้เรียกว่า อินเทอร์พรีเตอร์ ทำหน้าที่อ่านโปรแกรมต้นฉบับทีละบรรทัด แล้วแปลโปรแกรมบรรทัดนั้นเป็นเอ็กซีคิวต์เอเบิลโค้ดและทำงาน จากนั้นก็อ่านโปรแกรมต้นฉบับบรรทัดต่อไปมาทำเช่นเดิม ไปจนกว่าจะจบโปรแกรม การทำเช่นนี้เรียกว่าอินเทอร์เพต สรุปได้ดังรูปนี้

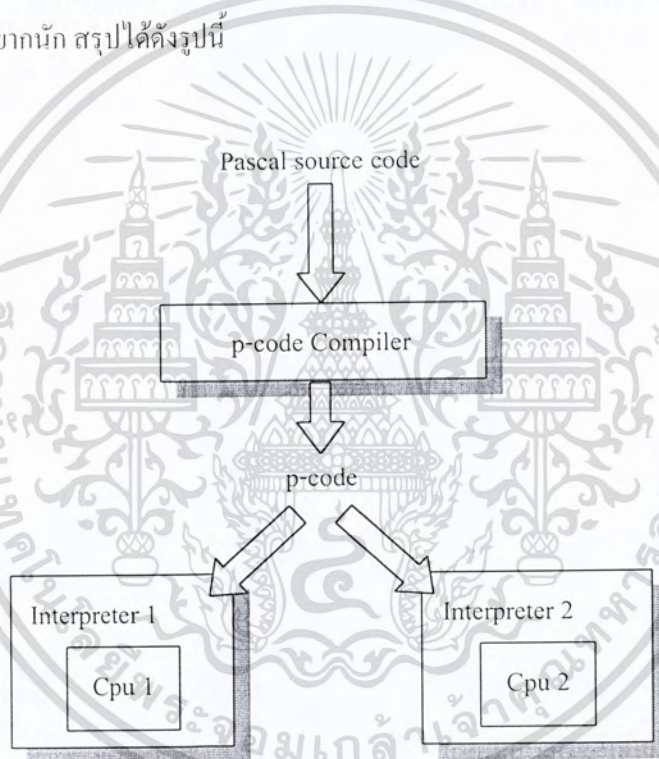


รูปที่ 10-2 แสดงการทำงานของอินเทอร์พรีเตอร์

วิธีนี้จะมีทั้งการแปลภาษาและทำงานโปรแกรมเกิดขึ้นสลับกันไป โปรแกรมจึงทำงานช้ากว่าการคอมไพล์เช่น แต่มีข้อดีคือ อินเทอร์พรีเตอร์ถูกสร้างขึ้นได้ง่ายและมีขนาดเล็กกว่า เพราะการวิเคราะห์และแปลโปรแกรมทีละบรรทัดทำได้ง่ายกว่าทำทั้งโปรแกรม และการแปลภาษาระหว่างโปรแกรมทำงานจะทำให้ได้ภาษาที่มีความยืดหยุ่นในการเขียนโปรแกรมกว่า ตัวอย่างภาษาที่ใช้การแปลภาษาแบบอินเทอร์พรีเทชัน คือ ลิปซ์, เบสิก, โพรล็อก และสมอลล์ทอล์ก

วิธีแปลภาษาทั้งสองแบบมีทั้งข้อดีและข้อเสีย โดยปกติภาษาหนึ่งจะเลือกใช้การแปลภาษาเพียงแบบใดแบบหนึ่ง แต่ก็มีภาษาที่ใช้การแปลภาษาทั้งสองแบบเพื่อประโยชน์บางประการตัวอย่างคือ

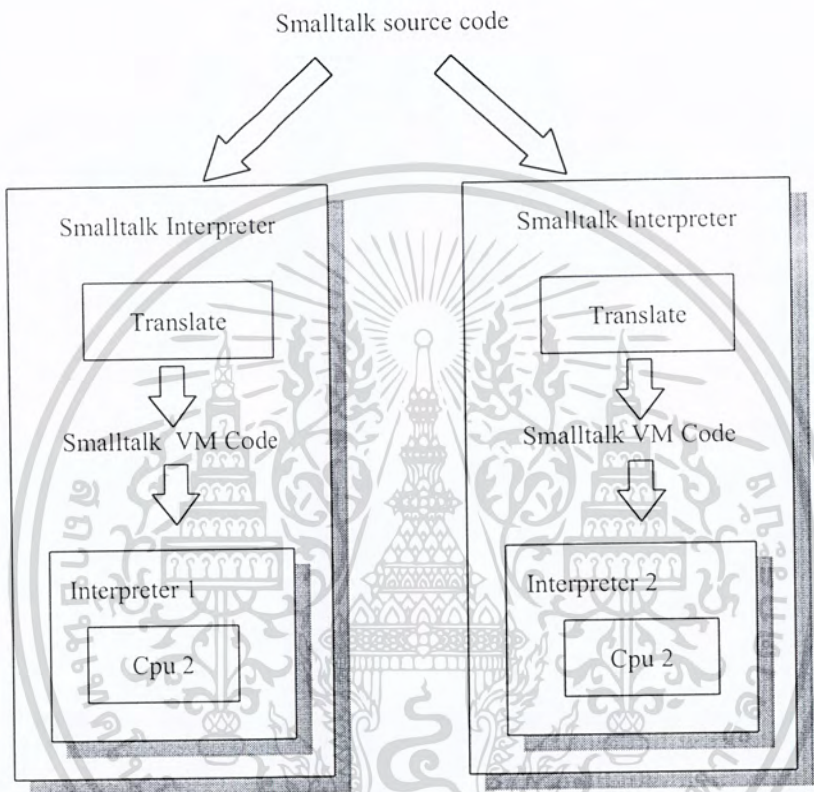
ในตอนต้น 1970s นักวิจัยที่ UC San Diego สร้างคอมไพเลอร์ที่แปลโปรแกรมภาษาปาสคาลเป็นโปรแกรมของภาษาสมมติหนึ่ง เรียกว่า p-code จากนั้นโปรแกรม p-code จะถูกทำงานโดย อินเทอร์พรีเตอร์ สำหรับหน่วยประมวลผลแต่ละรุ่น เหตุที่ต้องทำเช่นนี้ ก็เพราะในสมัยนั้นการสร้างคอมไพเลอร์ที่แปลภาษา ปาสคาลไปเป็นภาษาเครื่อง ถือเป็นเรื่องที่ยากมาก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



รูปที่ 10-3 การทำงานของโปรแกรม p-code

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

ในปลายยุค 1970s ภาษาสมอลล์ทอล์ก พัฒนาความคิดนี้ไปอีกแนวหนึ่งคือ แทนที่จะกำหนดเพียงภาษาสมมติเท่านั้น แต่กำหนดไปถึงโครงสร้างและพฤติกรรมของเครื่องจักรสมมติ (virtual machine) สำหรับทำงานโปรแกรมภาษาสมมตินั้นด้วย ภาษาสมอลล์ทอล์กใช้การแปลภาษาแบบอินเทอร์พรีเทชัน โดยสมอลล์ทอล์กอินเทอร์พรีเตอร์ ทำหน้าที่แปลโปรแกรมทีละบรรทัดไปเป็นโปรแกรมภาษาสมมติ แล้วทำงานโดยใช้เครื่องจักรสมมติที่จำลองขึ้น สรุปได้ดังรูปนี้

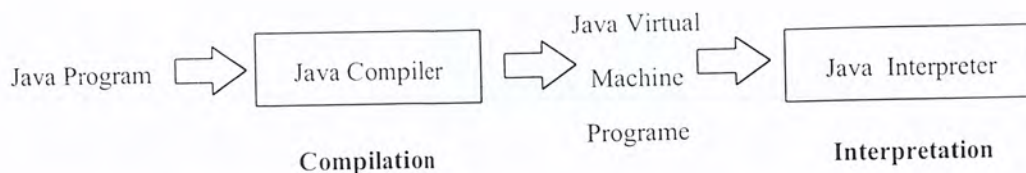


รูปที่ 10-4 แสดงการทำงานของเครื่องจักรสมมติ

วิธีการนี้ทำให้โปรแกรมภาษาสมอลล์ทอล์กทำงานบนเครื่องคอมพิวเตอร์ใดๆที่มีสมอลล์ทอล์กอินเทอร์พรีเตอร์ให้ผลลัพธ์เหมือนกันหมด โดยไม่ต้องเปลี่ยนแปลงโปรแกรมส่วนใดเลย แม้ภาษาสมอลล์ทอล์กจะประสบความสำเร็จในเรื่องนี้ แต่ยังคิดปัญหาที่โปรแกรมทำงานช้ามากจึงไม่ได้รับความนิยมเท่าที่ควร

10.4 เครื่องจักรสมมุติของจาวา (Java Virtual Machine)

ภาษาจาวานำความคิดการสร้างเครื่องจักรสมมติมาใช้ เพื่อให้โปรแกรมทำงานไม่ขึ้นกับระบบ โดยมีคอมไพเลอร์ทำการแปลภาษาให้เป็นโปรแกรมของ JVM แล้วนำโปรแกรมนั้นมาทำงานด้วยเครื่องจักรสมมติที่จำลองขึ้นโดยจาวาอินเทอร์พรีเตอร์ สรุปได้ดังรูปนี้



รูปที่ 10-5 การทำงานด้วยเครื่องจักรสมมติที่จำลองโดยจาวาอินเทอร์พรีเตอร์

ในวิธีนี้โปรแกรมภาษาจาวาจะถูกคอมไพล์โดยจาวาคอมไพล์เลอร์ได้เป็นโปรแกรมของ JVM แล้วสามารถนำไปทำงานบนเครื่องใดๆ ที่มีจาวาอินเทอร์พรีเตอร์ ได้จึงมีคุณสมบัติไม่ขึ้นกับระบบโปรแกรมของJVM จะทำงานได้เร็วกว่าการใช้อินเทอร์พรีเตอร์เพียงอย่างเดียวแบบสมอลล์ทอร์ก เพราะขั้นตอนการคอมไพล์จะถูกแยกออกไปจากการเอ็กซิกิวต์และด้วยการออกแบบคำสั่งของ JVM ให้ใกล้เคียงกับคำสั่งของหน่วยประมวลผลทั่วไปจาวาอินเทอร์พรีเตอร์ จึงเปลี่ยนคำสั่งของ JVM ไปสู่คำสั่งของหน่วยประมวลผลที่ใช้งานได้ง่าย การทำอินเทอร์พรีชันโปรแกรมของ JVM จึงเร็วกว่าการอินเทอร์พรีชันของภาษาระดับสูงอื่นๆ

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

ชุดคำสั่งของ JVM ถูกออกแบบมาเพื่อสนับสนุนการทำงานของโปรแกรมเชิงวัตถุจึงมีคำสั่งเกี่ยวกับการสร้างอินสแตนซ์ (instances) และการอ้างถึงสมาชิกในอินสแตนซ์ ซึ่งไม่มีในหน่วยประมวลผลทั่วไป ภาษาจาวาเป็นภาษาที่เน้นความถูกต้องเกี่ยวกับชนิดของข้อมูล (type) จึงมีคำสั่งสำหรับคำนวณชนิดข้อมูลพื้นฐานแต่ละชนิดเช่นคำสั่ง ladd สำหรับบวกเลขจำนวนเต็มชนิด integer และคำสั่ง dadd สำหรับบวกเลขทศนิยมชนิด double เป็นต้น บางคำสั่งของ JVM จะเหมือนกับคำสั่งที่มีในหน่วยประมวลผลทั่วไป

JVM ถูกออกแบบให้สามารถจำลองได้บนประมวลผลทั่วไป แต่หน่วยประมวลผลทั่วไปมีจำนวน รีจิสเตอร์ไม่เท่ากัน บางรุ่นมีรีจิสเตอร์หน้าที่พิเศษที่รุ่นอื่นไม่มี ผู้ออกแบบจึงตัดปัญหานี้โดยให้ JVM ไม่มีรีจิสเตอร์และทำการคำนวณทั้งหมดบนสแต็ก ชุดคำสั่งของ JVM จึงเป็น stacked operations กล่าวได้ว่า JVM เป็น Stack machine

วิชาวโพรเซสเซอร์ใน JVM จะเปลี่ยนคำสั่งไบต์โค้ดไปเป็นเนทีฟโค้ดที่ทำหน้าที่เดียวกันแล้วทำงาน เนทีฟโค้ดนั้น สังเกตว่าเนทีฟโค้ดนั้นอาจเป็น Application Program Interfaces (API) ของระบบปฏิบัติการที่ใช้หรืออาจจะเป็นไลบรารีมาตรฐานที่สร้างขึ้นสำหรับหน่วยประมวลผลนั้น ทำให้ JVM หนึ่งอาจใช้งานได้ในระบบต่างๆ โดยเปลี่ยนแปลงแค่ไลบรารีมาตรฐานเท่านั้น

โปรแกรมที่ทำงานโดยการอินเทอร์พรีเทนชั่มช้ากว่าโปรแกรมที่ทำงานโดยตรง ปัจจุบันโปรแกรมภาษาจาวาทำงานช้ากว่าภาษาซีโดยเฉลี่ยประมาณ 10 เท่า แต่ก็เร็วกว่าภาษาเบสิก, สมอลล์ทอล์ก หรือจาวาสคริปต์มาก มีการค้นคว้าเพื่อเพิ่มความเร็วของภาษาจาวาเช่นสร้างหน่วยประมวลผลที่สามารถทำงานคำสั่งของ JVM ได้โดยตรง และพัฒนาจาวาอินเทอร์พรีเตอร์ ซึ่งไม่ได้ทำงานคำสั่งของ JVM ทีละคำสั่ง แต่จะเปลี่ยนโปรแกรมของ JVM ทั้งโปรแกรมเป็นเนทีฟโค้ดแล้วทำงานเนทีฟโค้ดนั้น วิธีการนี้เรียกว่า Just in time เพราะคล้ายกับว่าทำการคอมไพล์โปรแกรม JVM ก่อนจะเริ่มทำงาน โปรแกรมจะทำงานเร็วขึ้นมากเพราะคำสั่งที่ถูกทำซ้ำบ่อยๆ เช่นคำสั่งที่อยู่ในประโยคทำซ้ำหรือฟังก์ชันที่ถูกเรียกใช้บ่อยๆ จะถูกแปลเพียงครั้งเดียว ไม่ใช่ทุกครั้งที่จะถูกทำงาน

บทที่ 11

การออกแบบและการสร้าง

การออกแบบโปรแกรมรับส่งเพิ่มข้อมูลแบบปลอดภัยสามารถทำได้โดยการนำเอาโพรโตคอลที่สร้างความปลอดภัยเข้ามาประยุกต์ โดยใช้โพรโตคอลเอสเอสเอชมาช่วยในการเข้ารหัสในส่วน of ข้อมูลที่เกี่ยวข้องกับการติดต่อสื่อสารกับเซิร์ฟเวอร์ซึ่งรวมไปถึงการเข้ารหัสชื่อผู้ใช้, รหัสผ่านและข้อมูลต่างๆ นอกจากนี้ยังช่วยเพิ่มความปลอดภัยในการพิสูจน์ตนเพื่อป้องกันการโจมตีประเภทต่างๆ อีกด้วย

ในการพัฒนานั้น เริ่มด้วยการศึกษาการทำงานของโพรโตคอลเอสเอสเอชที่จะนำมาใช้ในการพัฒนา ซึ่งโพรโตคอลนี้ ประกอบด้วยการทำงานในรูปแบบเวอร์ชัน 1 และ 2 การทำงานของโพรโตคอลเอสเอสเอช 1 และ 2 นั้นทำงานแตกต่างกันโดยสิ้นเชิง อันเป็นผลเนื่องมาจากโพรโตคอลเอสเอสเอช 2 นั้นได้ถูกออกแบบและอิมพลีเมนต์ขึ้นมาใหม่ทั้งหมด โดยไม่ยึดติดหลักการงานและการอิมพลีเมนต์เดิมมาใช้ในการพัฒนา ทำให้ต้องออกแบบโปรแกรมให้มีส่วนที่รองรับการทำงานทั้งสองเวอร์ชันแยกจากกันด้วย โดยทั้งนี้หากส่วนย่อยใดที่ทั้งสองเวอร์ชันทำงานคล้ายหรือเหมือนกัน จึงพัฒนาให้เป็นส่วนย่อยที่ใช้ร่วมกัน

การศึกษาได้แบ่งย่อยออกเป็น 2 ส่วน สอดคล้องกับแต่ละเวอร์ชันของโพรโตคอล เนื่องจากการทำงานของโพรโตคอลเอสเอสเอช 1 นั้นไม่ได้ออกแบบมาเพื่อการถ่ายโอนเพิ่มข้อมูลโดยเฉพาะ หากแต่ออกแบบมาเพื่อทำให้เกิดช่องทางที่ปลอดภัยแบบอนกจุดประสงค์ ทำให้ต้องใช้การประยุกต์ร่วมกันของการทำงานของโพรโตคอลเอฟทีพีและโพรโตคอลเอสเอสเอช 1 โดยอิมพลีเมนต์ตามข้อกำหนดของทั้งสองโพรโตคอล และใช้หลักการการฟอร์เวิร์ดพอร์ต ส่วนการพัฒนาให้รองรับโพรโตคอลเอสเอสเอช 2 นั้น จะพัฒนาตามหลักการงานของระบบย่อยเอสเอฟทีพี (sftp) ที่รองรับโดยโพรโตคอล โดยรายละเอียดจะได้อธิบายต่อไป

จากนั้นจึงเลือกภาษาที่จะนำมาใช้ในการพัฒนา ด้วยความปลอดภัยเป็นเงื่อนไขที่สำคัญในการพัฒนา จึงเลือกใช้ภาษาจาวาที่ให้ความปลอดภัยสูง อีกทั้งเป็นภาษาเชิงวัตถุที่สะดวกต่อการพัฒนาอีกด้วย

11.1 ภาพรวมของการออกแบบ

11.1.1 โพรโตคอลเอฟทีพี

โพรโตคอลเอฟทีพีที่ทำงานอยู่บนโพรโตคอลพื้นฐานคือ ทีซีพี/ไอพี การสื่อสารกันระหว่างเซิร์ฟเวอร์กับไคลเอ็นต์นั้นจะใช้ 2 ช่องทางการติดต่อเสมอ ช่องทางแรกจะใช้สำหรับส่งคำสั่งจากไคลเอ็นต์ไปเซิร์ฟเวอร์และการติดต่อกลับจากเซิร์ฟเวอร์มายังไคลเอ็นต์ (การเชื่อมต่อส่วนควบคุม : Control Connection) ช่องทางที่สองไว้สำหรับถ่ายโอนแฟ้มข้อมูล (การเชื่อมต่อส่วนข้อมูล : Data Connection) ซึ่งช่องทางแรกจะมีความอันตรายจากการถูกดักจับมาก เพราะเป็นช่องทางที่ใช้สำหรับส่งรหัสผู้ใช้งานและรหัสผ่าน

โพรโตคอลเอฟทีพีใช้ช่องทางการติดต่อช่องทางที่สองสำหรับการถ่ายโอนแฟ้มข้อมูล โดยทั่วไปการสร้างช่องทางการติดต่อสำหรับการถ่ายโอนแฟ้มข้อมูล เซิร์ฟเวอร์จะเป็นผู้เริ่มสร้างการติดต่อไปที่ไคลเอ็นต์เอง ลักษณะนี้เรียกว่าเซิร์ฟเวอร์ทำงานแบบแอ็กทีฟ อย่างไรก็ตาม มักจะเป็นปัญหากับเซิร์ฟเวอร์ที่มีการติดตั้งไฟร์วอลล์ ซึ่งมักจะไม่นิยมให้ทำงานแบบแอ็กทีฟ เพราะเหตุผลทางด้านความปลอดภัยบางประการ

เหตุผลที่ไฟร์วอลล์ไม่ยอมให้เซิร์ฟเวอร์ทำงานแบบแอ็กทีฟ

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

1. สแกนพอร์ต (port scanning) ผู้บุกรุกจะใช้เซิร์ฟเวอร์เป็นตัวที่จะสแกนพอร์ตกับเครื่องที่ต้องการได้ โดยสามารถซ่อนที่อยู่จริงของคนที่ทำได้
2. ความสามารถในการทะลุทะลวงผ่านไฟร์วอลล์ได้ เช่น สมมติว่าที่ไซต์แห่งหนึ่งได้เปิดบริการ เอฟทีพีแบบไม่ออกนาม (anonymous FTP) ไว้ ที่อยู่ข้างหลังไฟร์วอลล์ ผู้บุกรุกก็ทำการเริ่มการเชื่อมต่อไปยัง เอฟทีพีเซิร์ฟเวอร์ และส่งคำสั่ง PORT ตามด้วยพารามิเตอร์ที่บอกถึงที่อยู่และพอร์ตที่จะติดต่อ สมมติเป็น 8080 ซึ่งถูกบล็อกจากไฟร์วอลล์ ซึ่งทำให้ผู้บุกรุกสามารถติดต่อกับพอร์ตที่ต้องการได้

การแก้ปัญหาจากการที่ไม่สามารถทำงานใน แอ็กทีฟกับเอฟทีพีเซิร์ฟเวอร์ที่มีไฟร์วอลล์

การแก้ปัญหานี้ทำได้โดยไคลเอ็นต์ เพียงแค่ส่งคำสั่ง PASV เพื่อเปลี่ยนให้เซิร์ฟเวอร์ทำงานแบบพาสซีฟแทน แล้วให้ ไคลเอ็นต์เป็นผู้ติดต่อไปยังเซิร์ฟเวอร์เพื่อสร้างช่องทางสำหรับส่งแฟ้มข้อมูลแทน

ดังนั้นจึงเป็นข้อสรุปว่าการพัฒนาโปรแกรมรับส่งแฟ้มข้อมูลแบบปลอดภัยที่สร้างขึ้นนี้ถูกออกแบบมาให้ใช้กับพาซีฟเป็นหลัก แต่ก็สามารถทำงานแบบแอ็กทีฟได้ในโหมดที่ไม่ต้องการความปลอดภัยได้ด้วย

11.1.2 โพรโทคอลเอสเอสเอช

โพรโทคอลเอสเอสเอชที่นำมาใช้กับเอฟทีพี สามารถใช้การเข้ารหัสได้หลายวิธี และได้ออกแบบส่วนของการเรียกใช้งานให้เป็น โมดูลเพื่อที่สามารถเรียกใช้การเข้ารหัสแบบต่างๆ ได้ง่าย

คุณสมบัติของโพรโทคอลเอสเอสเอช ที่สำคัญมากที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมเอฟทีพีที่พัฒนาขึ้นมา มี 2 ส่วนคือ การใช้โพรโทคอลเอสเอฟทีพีในโพรโทคอลเอสเอสเอช 2 และการฟอร์เวิร์ดพอร์ต (Port Forwarding) ในโพรโทคอลเอสเอสเอช 1

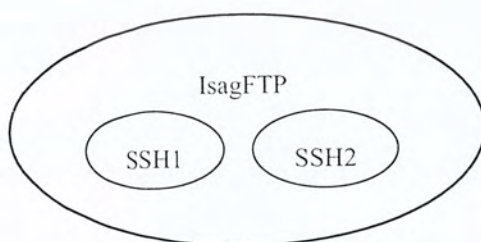
11.2 การออกแบบโปรแกรม

โปรแกรมที่ได้พัฒนามานี้มีชื่อว่า IsagFTP มีลักษณะในมุมมองของการใช้งานของผู้ใช้จะแบ่งออกเป็น 3 ส่วน เกิดจากประยุกต์เอาการใช้งานของโพรโทคอลเอสเอสเอชและ โพรโทคอลเอฟทีพีเข้าด้วยกัน

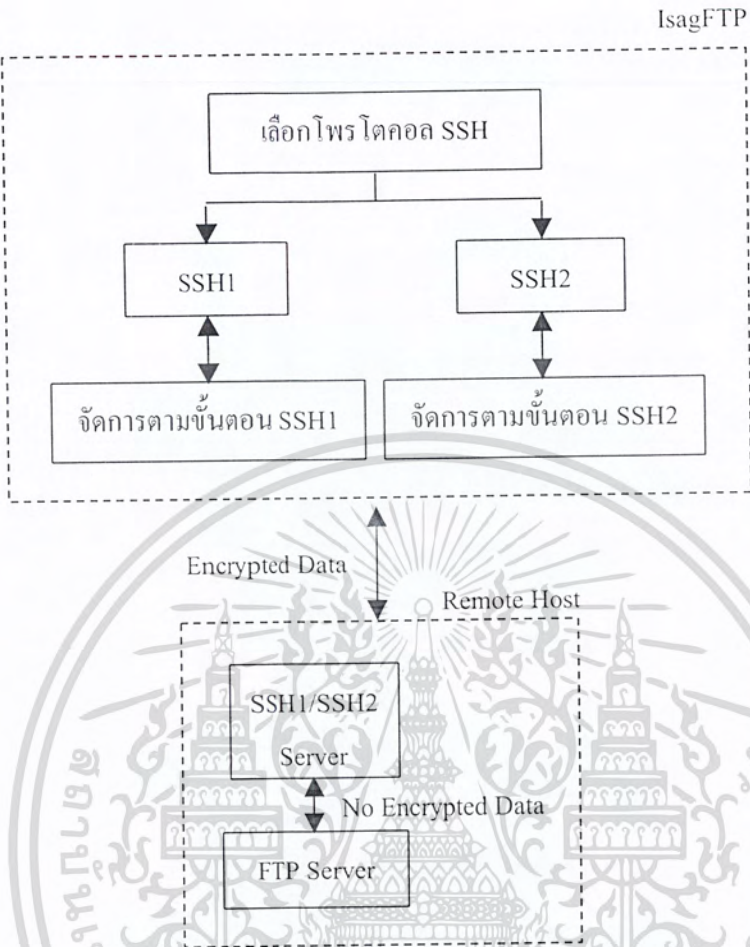
IsagFTP เป็นส่วนที่ทำหน้าที่ในการสื่อสารกับผู้ใช้งานในลักษณะที่เป็นกราฟิก เช่น การเลือกเซิร์ฟเวอร์ที่ต้องการติดต่อ, การเลือกอัลกอริทึมในการเข้ารหัส, การเก็บคุณสมบัติในการติดต่อกับเซิร์ฟเวอร์ เป็นต้น แล้วจะทำการโอนการทำงานให้กับ SSH1 หรือ SSH2 ต่อไป

SSH1 เป็นส่วนที่ทำหน้าที่ในการติดต่อกับโพรโทคอลเอสเอสเอช 1 ของเซิร์ฟเวอร์ ทำหน้าที่ได้เหมือนกับไคลเอ็นต์เอสเอสเอชทั่วไป จะเน้นที่การฟอร์เวิร์ดพอร์ต ซึ่งใช้ในการสร้างช่องทางการติดต่อสื่อสารที่ปลอดภัยให้กับโปรแกรมเอฟทีพี

SSH2 เป็นส่วนที่ทำหน้าที่ในการติดต่อกับโพรโทคอลเอสเอสเอฟ 2 ของเซิร์ฟเวอร์ ซึ่งโพรโทคอลเอสเอสเอชจะใช้ระบบย่อยเอสเอฟทีพีที่ไคลเอ็นต์ต้องการใช้ในการถ่ายโอนแฟ้มข้อมูล



รูปที่ 11-1 ภาพรวมของโปรแกรมมองจากผู้ใช้



รูปที่ 11-2 รูปแบบของโปรแกรมมองจากการใช้งาน

11.3 Socket APIs

หัวใจสำคัญของการติดต่อสื่อสารกับเครื่องผู้ให้บริการในระบบที่ซีพี/ไอที จำเป็นต้องทราบถึงอุปกรณ์ที่ใช้ในการติดต่อ ซึ่งอุปกรณ์นี้เรียกรวม ๆ ว่า Application Program Interface (API) ซึ่งกลุ่มของ API ที่ใช้ในการติดต่อผ่านเครือข่ายที่สำคัญคือ ซ็อกเก็ต (Socket) โดย ซ็อกเก็ตในแอปพลิเคชันนี้แบ่งออกเป็น 2 ลักษณะคือ ซ็อกเก็ตเซิร์ฟเวอร์และซ็อกเก็ตไคลเอ็นต์

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

ขั้นตอนที่ 1 ฝั่งของผู้ใช้บริการทำการเปิดซ็อกเก็ต โดยระบุถึงชื่อของโฮสต์และพอร์ตที่ให้บริการ ในขั้นตอนนี้ขอยกตัวอย่างของโค้ดภาษาจาวา

Socket ClientSocket;

ClientSocket = new Socket(Host,Port);

จากตัวอย่างของโค้ดข้างบนจะทำให้เกิดการทำงานดังนี้

1 : Listen



รูปที่ 11-3 ผู้ให้บริการทำการเปิดซ็อกเก็ต

ผู้ให้บริการจะเปิดซ็อกเก็ตของตัวเองรออยู่แล้ว เมื่อผู้ให้บริการเปิดซ็อกเก็ตจะทำให้เซิร์ฟเวอร์ค้นหาพอร์ตว่าง ๆ สำหรับสร้างช่องทางการติดต่อ

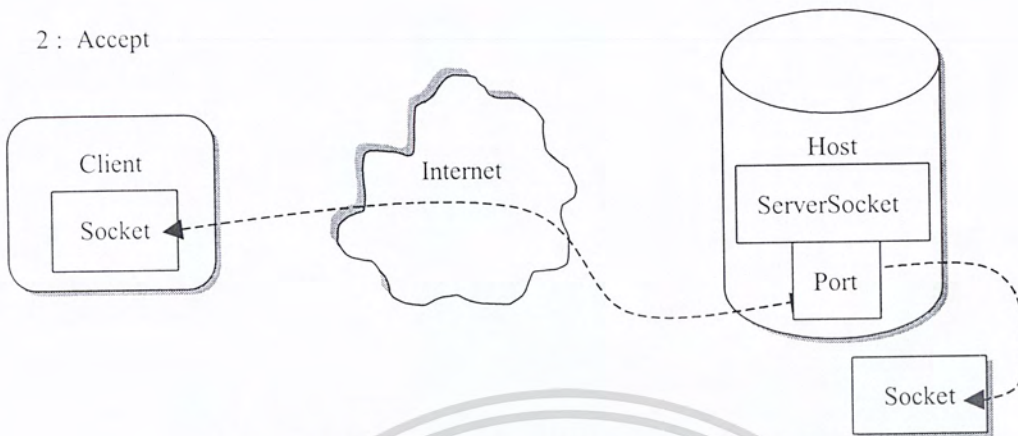
ขั้นตอนที่ 2 ผู้ใช้บริการรอ (Accept) จนกว่าเซิร์ฟเวอร์หาพอร์ตว่างได้ และจัดการกับตัวของเซิร์ฟเวอร์เองให้พร้อมสำหรับให้บริการ โค้ดข้างล่างจะแสดงการรอของฝั่งผู้ใช้บริการ

```

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

จากตัวอย่างของโค้ดในขั้นตอนที่ 2 ทำให้เกิดการทำงานดังนี้

2 : Accept



รูปที่ 11-4 ผู้ให้บริการทำการค้นหาพอร์ตและทำการเปิดพอร์ต

จากโค้ดข้างบนเมื่อมีการผิดพลาดประการใดจากการขอเปิดพอร์ตจะมีการดักจับความผิดพลาดด้วย เพื่อไม่ให้ระบบล้มเหลว เมื่อผู้ให้บริการพร้อมก็ส่งสัญญาณตอบรับกลับไปให้ยังผู้ใช้บริการได้ทราบเพื่อดำเนินขั้นตอนต่อไป

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

BufferedInputStream is;

BufferedOutputStream os;

try {

is = new BufferedInputStream(ClientSocket.getInputStream(),8192);

prepare create pipe and input buffer

}

catch(IOException e) {

Display(" Failed to get stream from socket");

System.exit(5);

}

try {

os = new BufferedOutputStream(ClientSocket.getOutputStream());

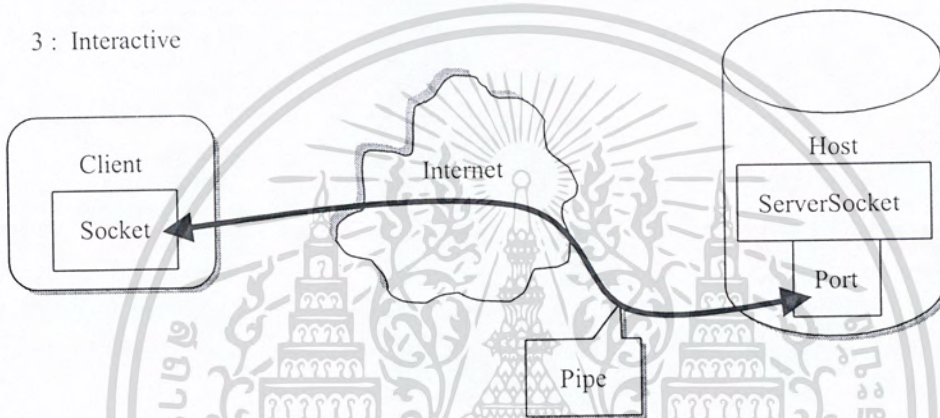

```

prepare create pipe and output buffer
}
catch(IOException e) {
    Display(" Failed to put stream on socket");
    System.exit(5);
}

```

จากตัวอย่างของโค้ดในขั้นตอนที่ 3 ทำให้เกิดการทำงานดังนี้

3 : Interactive



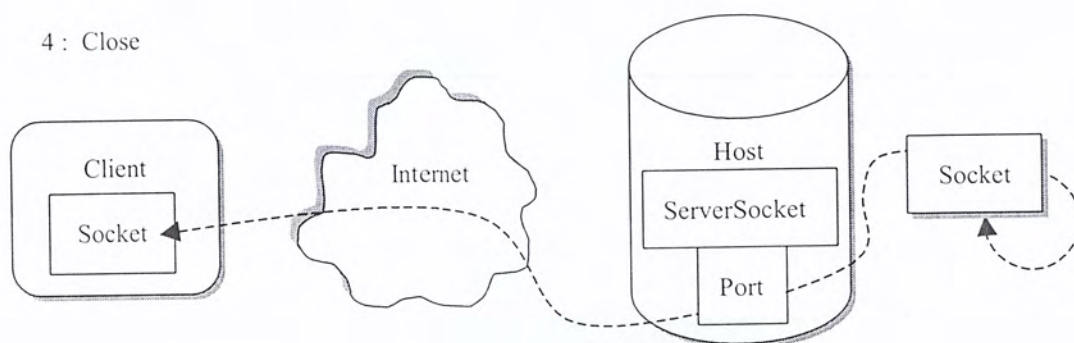
รูปที่ 11-5 แสดงการสร้างเส้นทางติดต่อ (Pipe line) และการสร้างบัฟเฟอร์

ขั้นตอนที่ 4 เป็นขั้นตอนสุดท้ายเมื่อผู้ใช้บริการต้องการปิดการติดต่อ จากโค้ดข้างล่างแสดงโปรแกรมการปิดการติดต่อ

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

จากตัวอย่างของโค้ดข้างบนทำให้เกิดการทำงานดังนี้

4 : Close



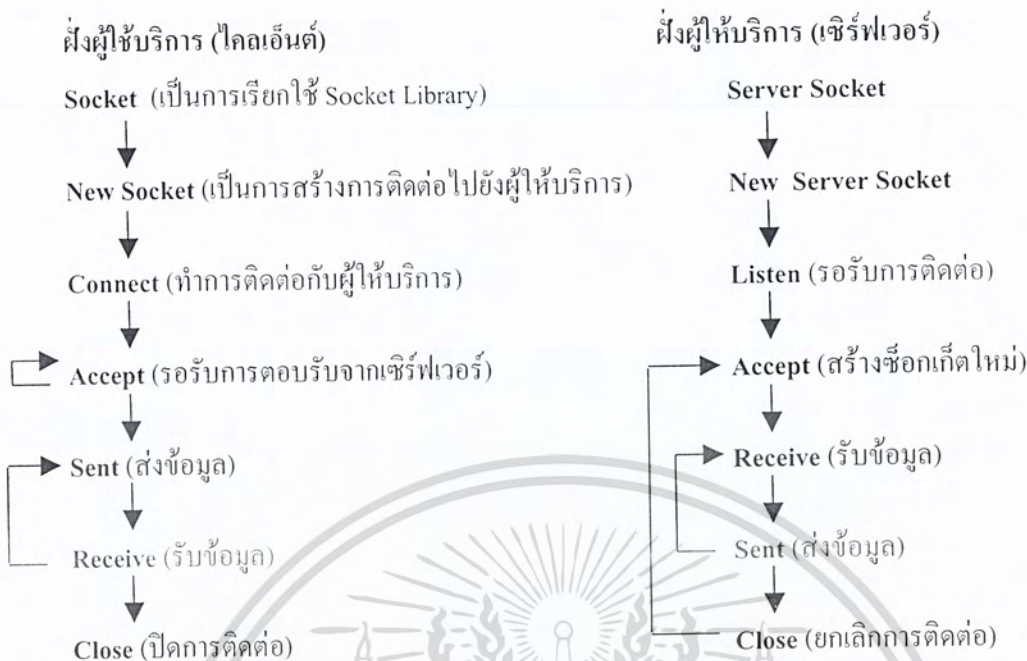
รูปที่ 11-6 แสดงการปิดซ็อกเก็ต

11.3.1 ภาพรวมของฟังก์ชันต่างๆ ที่ใช้สำหรับการติดต่อกับผู้ให้บริการ

เมื่อซ็อกเก็ตถูกสร้างขึ้น สามารถที่จะเป็นได้ทั้งผู้รับการติดต่อหรือผู้สร้างการติดต่อ ซ็อกเก็ตที่ผู้ให้บริการใช้จะคอยรับการติดต่อเข้ามา เรียกว่า **Passive Socket** ขณะที่ซ็อกเก็ตที่ถูกใช้โดยผู้ให้บริการ เรียกว่า **Active Socket** ซึ่งทั้งสองแบบนี้มีการสร้างเหมือนกัน แต่แตกต่างกันที่การใช้งาน

ในการใช้ซ็อกเก็ตในการติดต่อจะต้องมีการเรียกใช้ System Call โดยมีการส่งผ่านค่าพารามิเตอร์ที่จำเป็นลงไป ซึ่งจะใช้ฟังก์ชันของซ็อกเก็ต API ในการเรียก System Call ต่างๆ ดังต่อไปนี้

1. ฟังก์ชัน **Socket** เป็นคลาส Library ที่ใช้สำหรับสร้างการติดต่อกับฝั่งผู้ให้บริการ โดยจะต้องมีการระบุถึงตัวแปรต่างๆ เช่น หมายเลข IP ของเซิร์ฟเวอร์, พอร์ตที่ต้องการติดต่อเป็นต้น
2. ฟังก์ชัน **Connect** หลังจากที่เราสร้าง ซ็อกเก็ต แล้ว ผู้ให้บริการจะเรียกฟังก์ชันนี้ เพื่อทำการติดต่อไปยังเซิร์ฟเวอร์ที่ต้องการ
3. ฟังก์ชัน **Sent** ทั้งผู้ให้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการส่งข้อมูล โดยจะก๊อปปี้ข้อมูลที่จะส่งลงในบัฟเฟอร์เพื่อที่จะส่งออกไป
4. ฟังก์ชัน **Receive** ทั้งผู้ให้บริการและผู้ให้บริการจะใช้ฟังก์ชันนี้ในการรับการข้อมูล
5. ฟังก์ชัน **Close** ฟังก์ชันนี้จะใช้สำหรับจบการใช้ซ็อกเก็ตในแต่ละอัน
6. ฟังก์ชัน **Accept** เป็นฟังก์ชันที่ผู้ให้บริการใช้เปิดซ็อกเก็ตใหม่ เพื่อตอบรับการสร้างคู่ติดต่อของผู้ใช้บริการแต่ละคน
7. ฟังก์ชัน **Listen** เป็นฟังก์ชันเพื่อที่จะรอการติดต่อจากผู้ให้บริการ
8. ฟังก์ชัน **ServerSocket** เป็นคลาส Library ที่ใช้สำหรับรับการร้องขอจากผู้ขอใช้บริการ



รูปที่ 11-7 แสดงลำดับการจัดการกับซ็อกเก็ตบนที่ซีพี/ไอพี

11.4 รายละเอียดของการพัฒนา

คุณสมบัติของโปรโตคอลเอสเอสเอสเอช ที่สำคัญที่สุดที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมเอพีทีพีที่พัฒนาขึ้นมา มี 2 ส่วนคือ การใช้โปรโตคอลเอสเอสเอชทีพีในโปรโตคอลเอสเอสเอสเอช 2 และการฟอร์เวิร์ดพอร์ต (Port Forwarding) ในโปรโตคอลเอสเอสเอสเอช 1

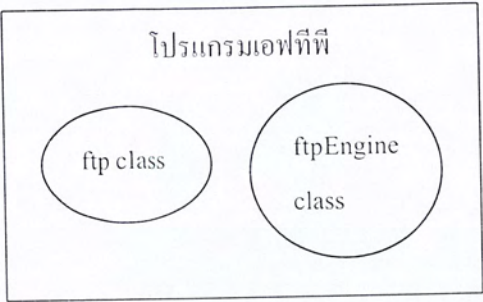
การพัฒนาจะแบ่งออกเป็นขั้นตอนหลักๆ 3 ขั้นตอนคือ

- การพัฒนาส่วนรองรับโปรโตคอลเอสเอสเอสเอช 1
- การพัฒนาส่วนรองรับโปรโตคอลเอสเอสเอสเอช 2
- การพัฒนาส่วนติดต่อกับผู้ใช้งาน เป็นแบบกรรพิก

11.4.1 การพัฒนาส่วนรองรับโปรโตคอลเอสเอสเอสเอช 1

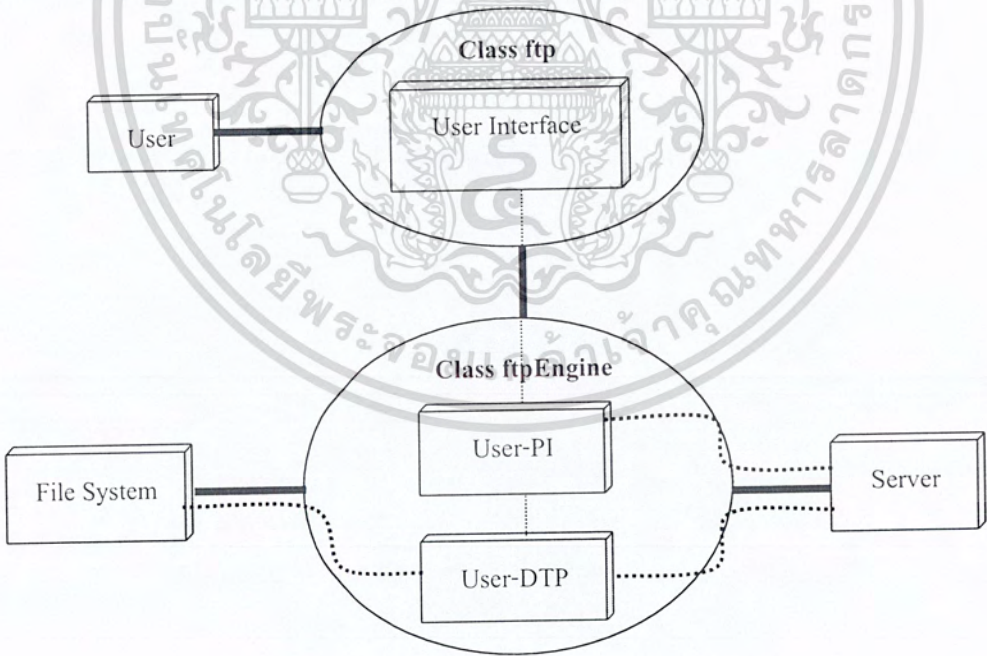
เนื่องจากการพัฒนาในส่วนนี้จะใช้หลักการฟอร์เวิร์ดพอร์ต การพัฒนาจึงประกอบด้วยการสร้างส่วนการถ่ายโอนแฟ้มข้อมูลปกติ (โปรแกรมเอพีทีพี) และการสร้างส่วนที่ทำการฟอร์เวิร์ดพอร์ตด้วยโปรโตคอลเอสเอสเอสเอช 1 มีรายละเอียดดังนี้

11.4.1.1 ขั้นตอนการสร้างส่วนโปรแกรมเอฟทีพี



รูปที่ 11-8 โครงสร้างของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น

โปรแกรม FTP ที่สร้างขึ้นมี 2 คลาสหลัก คือ ftpEngine กับ ftp คลาส ftpEngine จะเป็นส่วนหลักที่จะจัดการทางด้านเน็ตเวิร์ก ทำหน้าที่ในการเปิดการเชื่อมต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์ ส่งคำสั่งเอฟทีพีมาตรฐานและรับการตอบกลับพร้อมทั้งแปลความหมาย (User-PI) และเป็นส่วนดูแลการถ่ายโอนแฟ้มข้อมูล (User-DTP) ซึ่งจะถูกส่งงานผ่านทางคลาส ftp



รูปที่ 11-9 แสดงการทำงานของโปรแกรมเอฟทีพีที่ได้พัฒนาขึ้น

คลาส ftp เป็นคลาสที่ถูกออกแบบมาเพื่อรับคำสั่งจากผู้ใช้งานโดยแสดงผลเป็นกราฟิก (Graphic User Interface) โดยจะทำงานผ่านทางคลาส ftpEngine ซึ่งในจุดนี้ทำให้สามารถพัฒนาส่วนนี้ใหม่เพื่อรูปแบบการใช้งานในรูปแบบอื่นๆ โดยที่ไม่ต้องจัดการกับส่วนการโอนย้ายข้อมูลเลย

การรับข้อมูลจากผู้ใช้งาน

การรับคำสั่งจะคล้ายคลึงกับโปรแกรมเอฟทีพีที่มีลักษณะเป็นกราฟิกทั่วไป โดยจะทำการเชื่อมโยงการทำงานของ menubar, toolbar และ event ต่างๆ ที่รับจากผู้ใช้งาน ให้เข้ากับคำสั่งเอฟทีพีมาตรฐาน และฟังก์ชันที่ใช้ในการทำงานของโปรแกรม

การรับข้อมูลจะทำการรับ action ต่างๆ ที่ผู้ใช้งานกระทำกับโปรแกรม แล้วนำมาพิจารณาว่า action ที่รับมานั้นเชื่อมโยงกับฟังก์ชันใดก็จะเรียกใช้งานฟังก์ชันนั้น

การแสดงผลทางหน้าจอ

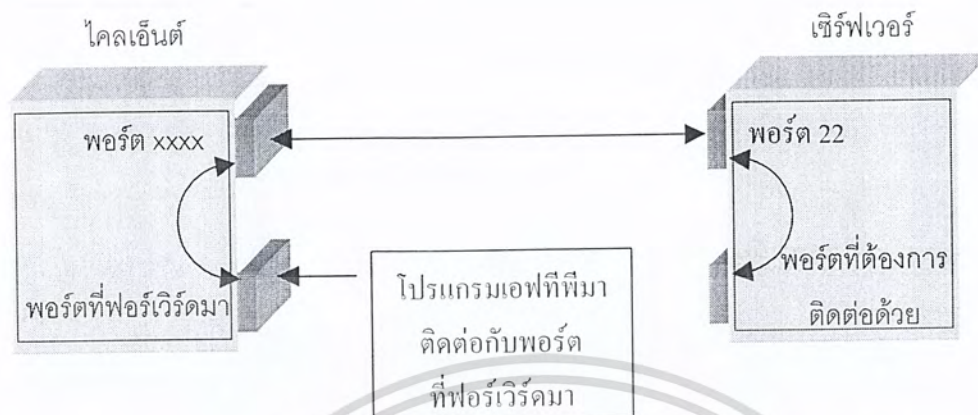
การแสดงผลจะแสดงผลในลักษณะที่เป็น กราฟิกที่เข้าใจได้ง่ายเท่านั้น เนื่องจากโปรแกรมนี้นั้นพัฒนาขึ้นมาเพื่อการใช้งานจริง และต้องการความสะดวกในการใช้งาน

11.4.1.2 ขั้นตอนการสร้างส่วนที่ทำการฟอร์เวิร์ดด้วยโปรโตคอลเอสเอสเอช 1

โปรโตคอลเอสเอสเอช 1 ที่นำมาใช้กับเอฟทีพีนี้จะใช้การเข้ารหัสแบบ 3DES เป็นหลัก และได้ออกแบบส่วนของการเรียกใช้งานให้เป็นโมดูลเพื่อที่สามารถเรียกใช้การเข้ารหัสแบบอื่น ๆ ได้ง่าย

คุณสมบัติของโปรโตคอลเอสเอสเอช 1 ที่สำคัญมากที่ต้องนำมาใช้เสริมสร้างความปลอดภัยให้กับโปรแกรมเอฟทีพี ที่พัฒนาขึ้นมาก็คือ การฟอร์เวิร์ดพอร์ต (Port Forwarding)

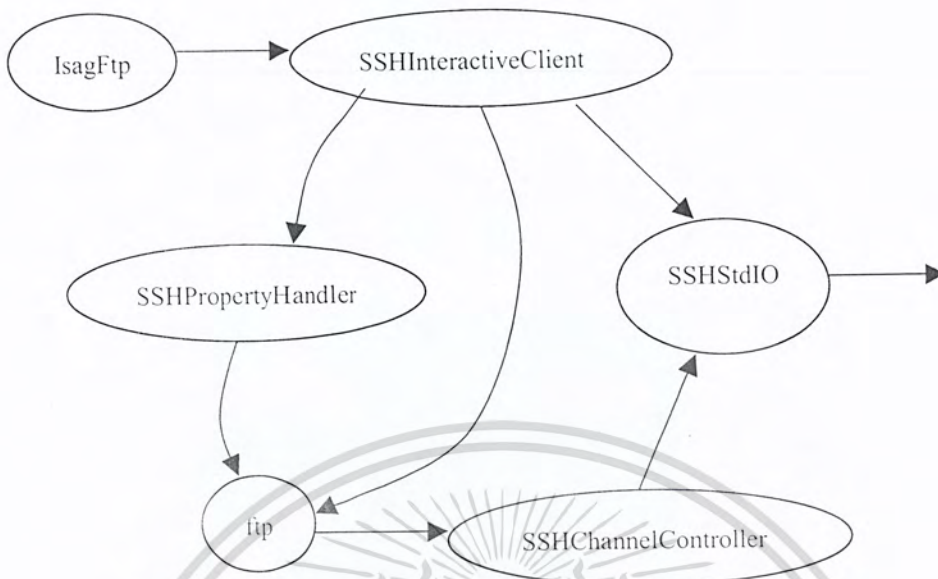
การทำงานของฟอร์เวิร์ดพอร์ตของเซิร์ฟเวอร์นั้น เป็นการส่งการเชื่อมต่อจากเครื่องๆ หนึ่งไปยังพอร์ตของอีกเครื่องหนึ่ง มีหลักการง่ายๆ ก็ระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์ จะมีการสร้างการเชื่อมต่อที่มีความปลอดภัยไว้ การส่งข้อมูลผ่านระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์จะมีการเข้ารหัสตลอด โปรแกรมเอฟทีพี เพียงแค่ติดต่อไปยังพอร์ตที่ทำการฟอร์เวิร์ดมาของไคลเอ็นต์เท่านั้น การส่งข้อมูลก็จะถูกส่งต่อจากพอร์ตดังกล่าวผ่านการช่องทางการติดต่อสื่อสารที่มีการสร้างความปลอดภัยไว้แล้วไปยังเซิร์ฟเวอร์โดยอัตโนมัติ



รูปที่ 11-10 แสดงการใช้งานของพอร์ตฟอร์เวิร์ด

โดยมีคลาสที่จัดการตามโพรโตคอลเอสเอสเอช 1 ดังนี้

- **IsagFtp** คลาสนี้จะทำหน้าที่จัดการเริ่มต้นค่าต่างๆ จากอาร์กิวเมนต์คอนเรียกใช้โปรแกรม หากไม่มีอาร์กิวเมนต์ก็จะนำค่าที่เป็นค่าที่กำหนดไว้ในโปรแกรมก่อนแล้วมาใช้ หลังจากที่มีการเซตค่าเป็นที่เรียบร้อยแล้วก็จะทำการเริ่มการทำงานโดยเรียกใช้คลาส **SSHInteractiveClient** ต่อไป
- **SSHInteractiveClient** คลาสนี้จะเริ่มต้นการทำงานตามโพรโตคอลของเช็กเคียวเซลล์ก่อน หลังจากนั้นจะทำการพิสูจน์ตน หากทางเซิร์ฟเวอร์ไม่มีการรองรับโพรโตคอลเช็กเคียวเซลล์เวอร์ชันหนึ่งหรือโพรโตคอลเช็กเคียวเซลล์เวอร์ชันสอง ก็จะเลิกการทำงานและผ่านการทำงานไปให้โปรแกรมเอฟทีพีธรรมดา ซึ่งจะ ไม่มีการเข้ารหัสแต่อย่างใด แต่ถ้าหากมีการรองรับแล้ว ก็จะเริ่มผ่านการทำงานให้โปรแกรมเอฟทีพีธรรมดาและข้อมูลที่ถูกส่งก็จะมีการเข้ารหัสตลอด
- **SSHStdIO** คลาสนี้จะรับตัวอักษรที่พิมพ์มาจากผู้ใช้ จะใช้ตอนช่วงการพิสูจน์ตนโดยการใส่ชื่อผู้ใช้และรหัสผ่านสำหรับโพรโตคอลของเช็กเคียวเซลล์
- **SSHPropertyHandler** คลาสนี้จะเก็บค่าคุณสมบัติต่างๆ ของโปรแกรมไว้ใช้ เช่น ชื่อโฮสต์ที่ติดต่ออยู่ ชื่อผู้ใช้งานในขณะนั้น ชนิดของการเข้ารหัสที่ใช้อยู่ พอร์ตที่ใช้ในการฟอร์เวิร์ด โดยจะถูกเก็บจากคลาส **IsagFtp** เพื่อสามารถนำออกมาใช้ได้เมื่อเรียกใช้งานคลาสนี้
- **SSHChannelController** คลาสนี้จะจัดการกับเกี่ยวกับการรองรับการติดต่อจากโปรแกรมเอฟทีพี เพื่อนำข้อมูลมาทำการเข้ารหัส และ ทำหน้าที่ฟอร์เวิร์ดข้อมูลนั้นผ่านทางช่องทางที่สร้างขึ้นมา โดยทำหน้าที่ควบคุมการสร้างและทำลายช่องทางที่สร้างขึ้นมา

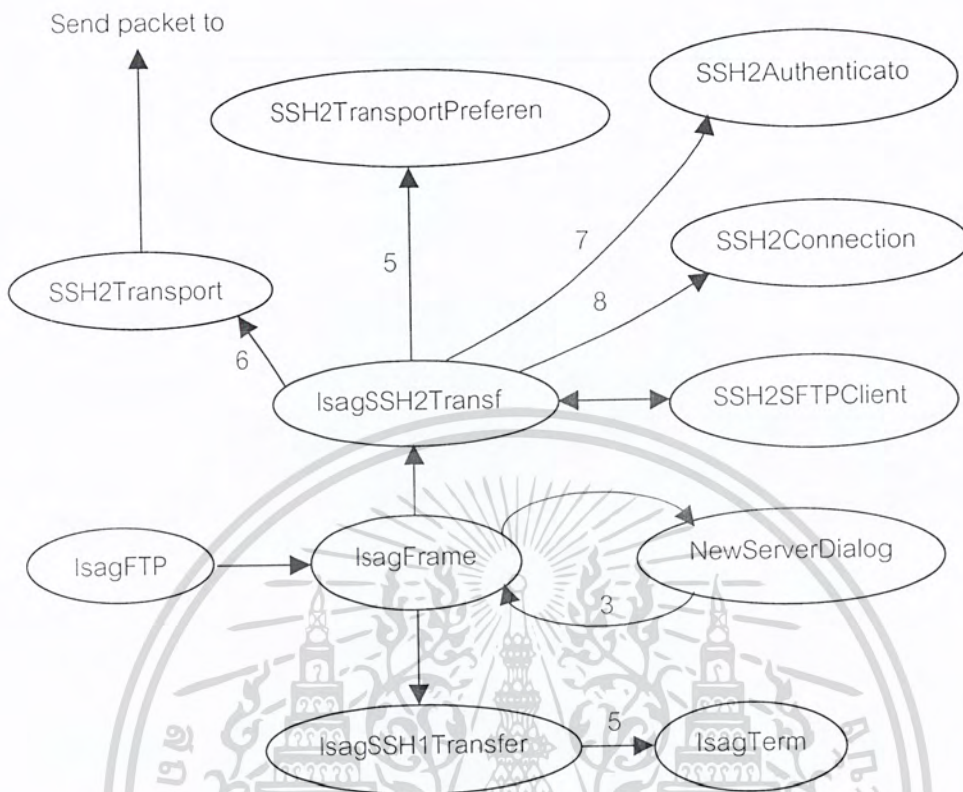


รูปที่ 11-11 แสดงการทำงานของส่วนรองรับโพรโตคอลเอสเอสเอช 1 ที่ได้พัฒนาขึ้น

11.4.2 การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอช 2

ในส่วนนี้จะทำการพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอช 2 ในการถ่ายโอนแฟ้มข้อมูล โดยใช้ระบบบ้อยเอสเอฟทีพีในการพัฒนา ซึ่งจะนำส่วนรองรับโพรโตคอลเอสเอสเอช 1 มาทำการเชื่อมต่อด้วย เพื่อให้เกิดการทรานสแพเรนต์ต่อผู้ใช้ นั่นคือผู้ใช้ไม่จำเป็นต้องรู้ว่าเซิร์ฟเวอร์ให้บริการที่ตนติดต่ออยู่ใช้โพรโตคอลเอสเอสเอชเวอร์ชันใด โปรแกรมจะทำการเลือกเวอร์ชันให้เอง โดยยึดความปลอดภัยเป็นที่ตั้ง

ในขั้นตอนแรกของการทำงานจะต้องมีการตกลงกันระหว่างไคลเอนต์กับเซิร์ฟเวอร์ตามโพรโตคอลเอสเอสเอชก่อน ไม่เช่นนั้นจะไม่มีการทำงานของโปรแกรมเอฟทีพีเลย ซึ่งโปรแกรมจะทำการตรวจสอบให้ว่า เซิร์ฟเวอร์ที่ต้องการเชื่อมต่อนั้นสนับสนุนการทำงานของโพรโตคอลเช็กเกียเวลล์เวอร์ชันใดบ้าง ถ้าเซิร์ฟเวอร์นั้นสนับสนุนการทำงานของโพรโตคอลทั้งสองเวอร์ชัน โปรแกรมจะทำการเชื่อมต่อกับโพรโตคอลเอสเอสเอช 2 ให้ หลังจากนั้นก็จะโอนการทำงานให้โพรโตคอลเอสเอฟทีพี แต่ถ้าหากเซิร์ฟเวอร์สนับสนุนการทำงานของโพรโตคอลเอสเอสเอช 1 เพียงอย่างเดียวก็จะทำการเชื่อมต่อกับโพรโตคอลเอสเอสเอช 1 ให้ จากนั้น จะทำการทำงานให้กับโปรแกรมเอฟทีพี โปรแกรมเอฟทีพีจะทำหน้าที่เหมือนกับโปรแกรมเอฟทีพีทั่วไป เพียงแต่ข้อมูลที่ส่งออกไปยังเครือข่าย จะต้องผ่านการเข้ารหัสจากโปรแกรมเสียก่อน โดยมีคลาสที่จัดการดังนี้



รูปที่ 11-12 แสดงการทำงานของส่วนรองรับโปรโตคอลเอสเอสเอช 2 ที่ได้พัฒนาขึ้น

- **IsagFtp** คลาสนี้จะทำหน้าที่จัดรูปแบบของ GUI ให้เหมาะสมกับระบบปฏิบัติการที่ใช้
- **IsagFrame** คลาสนี้จะเริ่มต้นการทำงานโดยสร้าง frame ขึ้นมาเพื่อติดต่อกับผู้ใช้ เมื่อผู้ใช้ต้องการเชื่อมต่อกับเซิร์ฟเวอร์ ก็จะมีการตรวจสอบว่า เซิร์ฟเวอร์นั้นสนับสนุนการทำงานของโปรโตคอลเช็กเคลียเซสส์เวอร์ชันใดบ้าง จากนั้นจึงจะโอนการทำงานให้กับคลาส **IsagSSH1Transfer** หรือ **IsagSSH2Transfer** ต่อไป
- **NewServerDialog** ก่อนที่ผู้ใช้จะทำการเชื่อมต่อกับเซิร์ฟเวอร์ได้นั้น ต้องทำการกำหนดคุณสมบัติต่างๆที่ต้องใช้ในการเชื่อมต่อเสียก่อน ซึ่งคลาสนี้จะทำการเก็บค่าต่างๆไว้ และสามารถให้ผู้ใช้ทำการแก้ไขเปลี่ยนแปลงค่าต่างๆได้
- **IsagSSH1Transfer** คลาสนี้จะทำการเชื่อมต่อกับเซิร์ฟเวอร์ด้วยโปรโตคอลเอสเอสเอช 1, ทำการพิสูจน์ตน, สร้าง channel ที่มีการเข้ารหัสขึ้นมา พร้อมทั้งสร้าง GUI ที่ใช้ในการถ่ายโอนแฟ้มข้อมูล
- **IsagSSH2Transfer** คลาสนี้จะทำการเชื่อมต่อกับเซิร์ฟเวอร์ด้วยโปรโตคอลเอสเอสเอช 2, ทำการพิสูจน์ตน แล้วจึงโอนการทำงานให้โปรโตคอลเอสเอสเอชที่ 2 โดยโปรโตคอลเอสเอสเอชที่ 2 จะอาศัยช่องทางที่ถูกสร้างในคลาสนี้ในการถ่ายโอนแฟ้มข้อมูลและคำสั่งต่างๆ

- **SSH2SFTPClient** คลาสนี้จะทำงานตามโพรโตคอลเอสเอฟทีพี โดยการงานจะมีลักษณะคล้ายกับโพรโตคอลเอฟทีพีทั่วไป แต่อาศัยการเชื่อมต่อที่ได้จากโพรโตคอลเอสเอสเอช 2
- **SSH2TransportPreferences** คลาสนี้จะทำการเก็บค่าคุณสมบัติต่างๆ ที่ต้องใช้ในการสร้างการติดต่อกับเซิร์ฟเวอร์
- **SSH2Transport** คลาสนี้จะนำคุณสมบัติต่างๆจากคลาส SSH2TransportPreferences มาใช้ แล้วจึงทำการสร้างซ็อกเก็ตเพื่อใช้ในการติดต่อกับเซิร์ฟเวอร์ และทำหน้าที่สร้างแพ็กเก็ตที่รับหรือส่งระหว่างเซิร์ฟเวอร์ นอกจากนี้ยังทำหน้าที่ในการแลกเปลี่ยนคีย์สาธารณะอีกด้วย
- **SSH2Authenticator** คลาสนี้ทำหน้าที่ในการพิสูจน์ตนกับเซิร์ฟเวอร์ โดยจะส่งข้อมูลต่างๆไปยังคลาส SSH2Transport เพื่อทำการสร้างแพ็กเก็ต และทำการเข้ารหัส แล้วจึงส่งไปให้เซิร์ฟเวอร์
- **SSH2Connection** คลาสนี้จะทำการสร้างการเชื่อมต่อกับเซิร์ฟเวอร์โดยใช้ การพิสูจน์ตนจากคลาส SSH2Authenticator และใช้การรับและส่งแพ็กเก็ตจากคลาส SSH2Transport



บทที่ 12

การทดลองและผลการทดลอง

หลังจากที่ได้พัฒนาโปรแกรมตามที่ได้ออกแบบไว้ในบทที่ 11 แล้ว จำเป็นต้องมีการทดสอบโปรแกรมที่ได้พัฒนาโดยทดลองติดตั้งโปรแกรมกับเครื่องที่มีระบบปฏิบัติการต่างๆ กัน แล้วทดลองใช้โปรแกรมติดต่อเพื่อขอใช้งานเอฟทีพีกับเซิร์ฟเวอร์ที่มีการเปิดบริการโปรโตคอลเอสเอสเอชและมีสิทธิ์เข้าใช้บริการการถ่ายโอนแฟ้มข้อมูล

12.1 จุดประสงค์ของการทดลอง

1. ทดสอบการใช้งานของโปรแกรมที่พัฒนาขึ้นตามโปรโตคอลมาตรฐานของโปรโตคอลเอสเอสเอชว่าสามารถทำงานได้อย่างถูกต้องหรือไม่
2. ทดสอบการใช้งานของโปรแกรมในแต่ระบบปฏิบัติการต่าง ๆ ที่มีการติดตั้ง JVM (Java Virtual Machine) ๑
3. ตรวจสอบความถูกต้องของการใช้งานและขั้นตอนการทำงานของโปรแกรม IsagFTP และประเมินผลว่าประสิทธิภาพของโปรแกรมที่ได้

12.2 การทดลองการใช้งานโปรแกรม

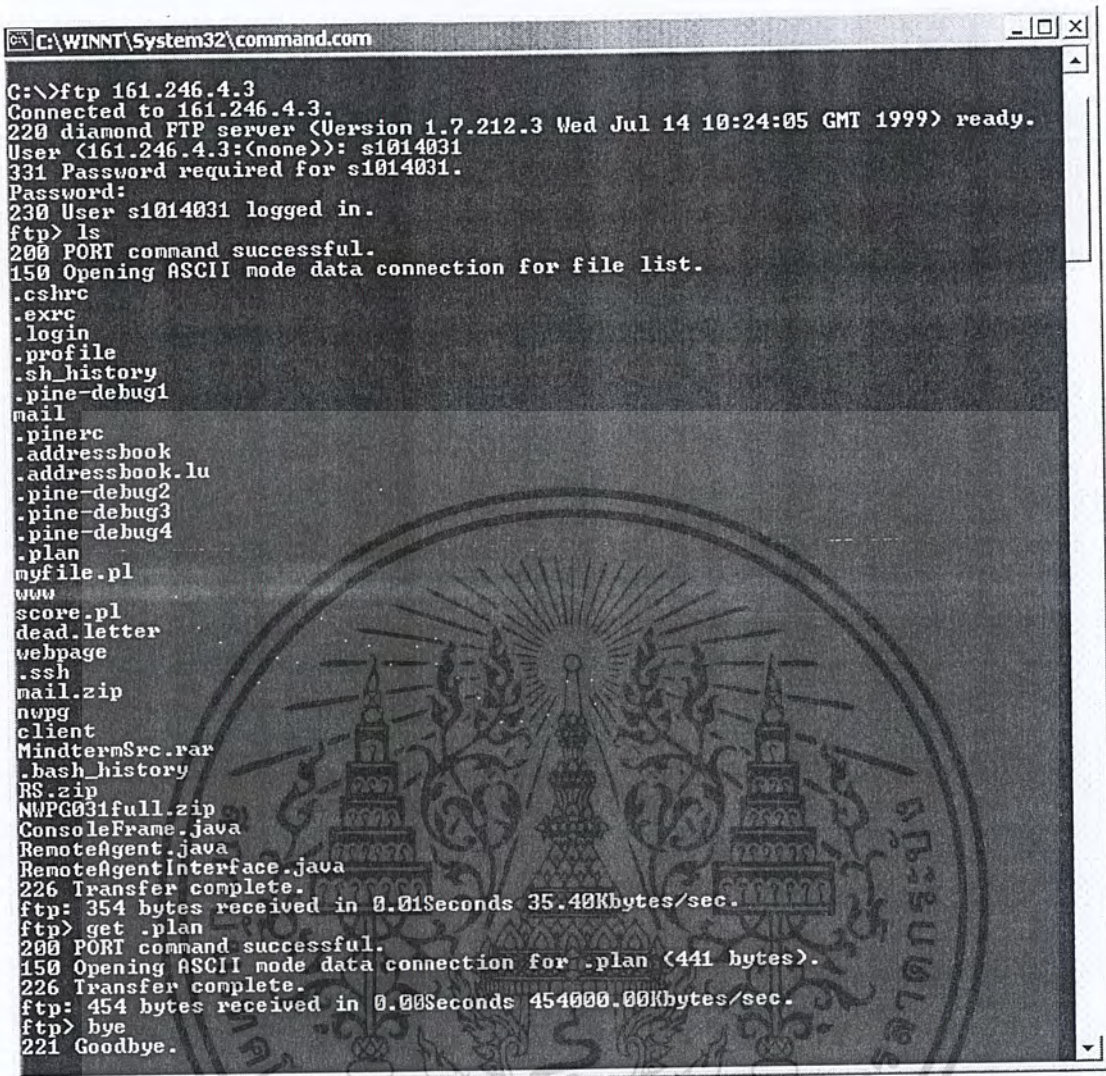
12.2.1 การทดลองใช้โปรแกรมเอฟทีพีเดิม

โปรแกรมเอฟทีพีที่ไม่มีการเข้ารหัสถูกใช้กันอย่างแพร่หลาย มีทั้งแบบที่เป็นแบบโหมดข้อความ (text mode) และแบบโหมดกราฟิก ซึ่งในที่นี้จะได้ทดลองให้เห็นถึงความไม่ปลอดภัยจากการใช้งานโปรแกรมเอฟทีพีเดิม

รูปที่ 12-1 แสดงให้เห็นตัวอย่างการใช้งานโปรแกรมเอฟทีพีในรูปแบบโหมดข้อความ จะเห็นว่าผู้ใช้จำเป็นต้องป้อนข้อความเป็นคำสั่งในการถ่ายโอนแฟ้มข้อมูล ทำให้ผู้ที่เริ่มใช้จำเป็นต้องจดจำคำสั่งเหล่านี้ ตัวอย่างโปรแกรมเอฟทีพีนี้เป็นโปรแกรมเอฟทีพีที่มากับระบบปฏิบัติการวินโดวส์

จากภาพผู้ใช้ที่มีรายชื่อ s1014031 ของเซิร์ฟเวอร์ diamond ซึ่งมีหมายเลขไอพีแอดเดรสคือ 161.246.4.3 ได้ทำการใช้โปรแกรมเอฟทีพีถ่ายโอนแฟ้มข้อมูล ไฟล์ .plan ซึ่งเป็นเท็กซ์ไฟล์ใช้ในการแสดงข้อความบรรยายของแต่ละผู้ใช้

ผู้ทดลองได้ใช้โปรแกรมในการดักจับแพ็กเก็ต (sniffer) ชื่อโปรแกรม Iris The Network Traffic Analyzer v2.0 ซึ่งสามารถดาวน์โหลดโปรแกรมทดลองใช้งานได้จากเว็บไซต์ผู้พัฒนาคือ <http://www.cceye.com>



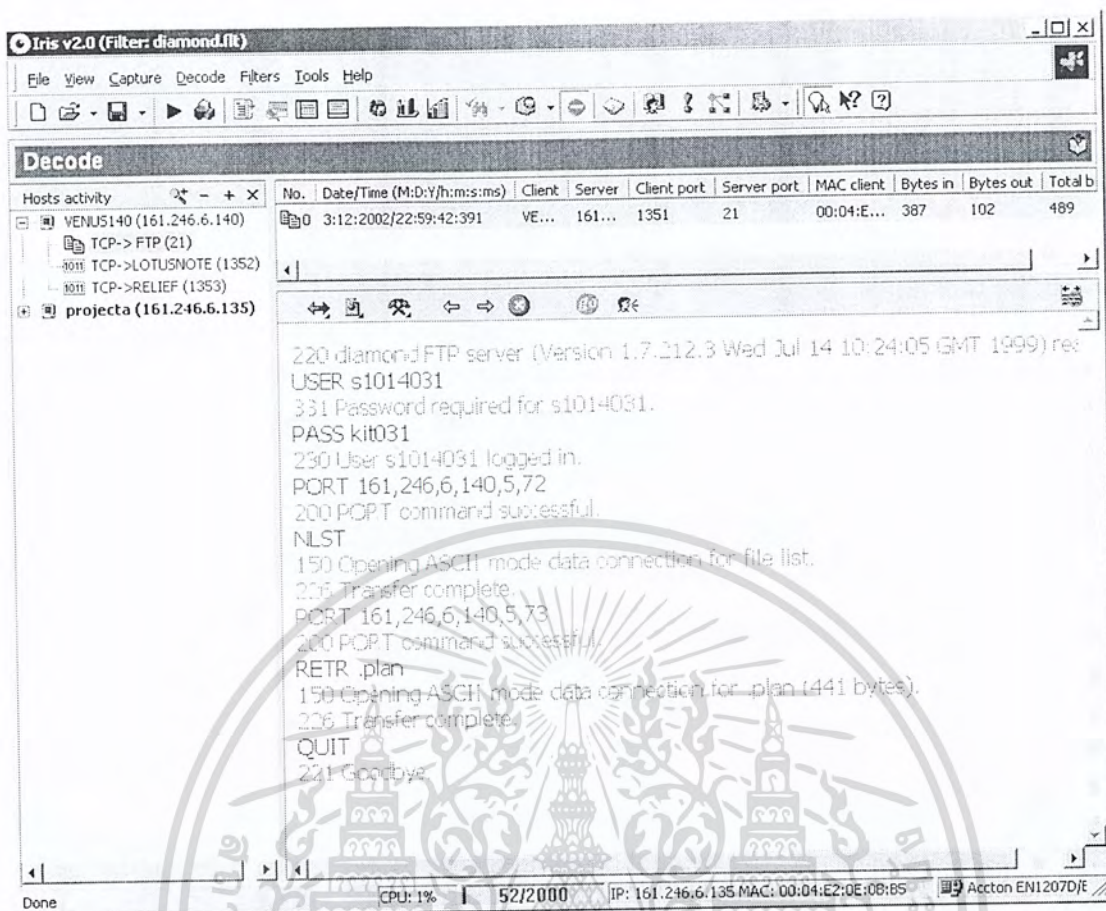
```

C:\WINNT\System32\command.com
C:\>ftp 161.246.4.3
Connected to 161.246.4.3.
220 diamond FTP server (Version 1.7.212.3 Wed Jul 14 10:24:05 GMT 1999) ready.
User (161.246.4.3:(none)): s1014031
331 Password required for s1014031.
Password:
230 User s1014031 logged in.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
.cshrc
.exrc
.login
.profile
.sh_history
.pine-debug1
mail
.pinerc
.addressbook
.addressbook.lu
.pine-debug2
.pine-debug3
.pine-debug4
.plan
myfile.pl
www
score.pl
dead.letter
webpage
.ssh
mail.zip
nupg
client
MindtermSrc.rar
.bash_history
RS.zip
NWPG031full.zip
ConsoleFrame.java
RemoteAgent.java
RemoteAgentInterface.java
226 Transfer complete.
ftp: 354 bytes received in 0.01Seconds 35.40Kbytes/sec.
ftp> get .plan
200 PORT command successful.
150 Opening ASCII mode data connection for .plan (441 bytes).
226 Transfer complete.
ftp: 454 bytes received in 0.00Seconds 454000.00Kbytes/sec.
ftp> bye
221 Goodbye.

```

รูปที่ 12-1 โปรแกรมเอฟทีพีทีพีที่มากับระบบปฏิบัติการวินโดวส์

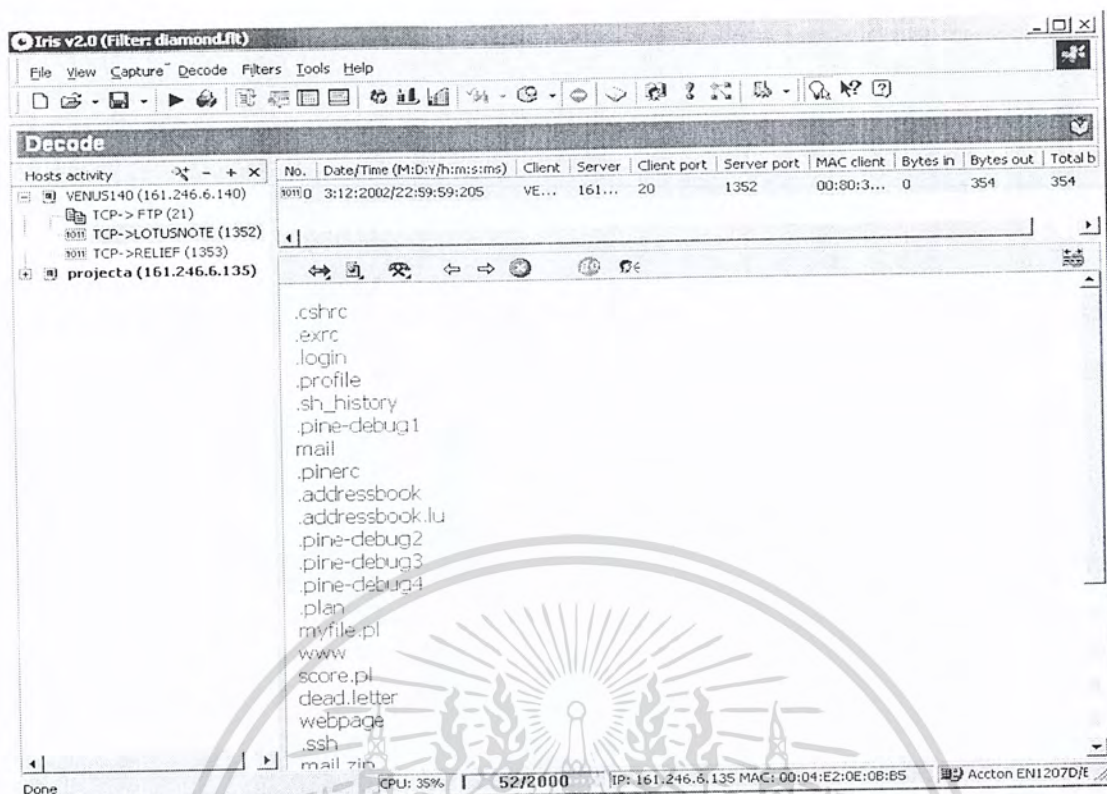
รูปที่ 12-2 แสดงให้เห็นว่า มีการสร้างการเชื่อมต่อส่วนควบคุม (control connection) ระหว่างไคลเอนต์ หมายเลขไอพีแอดเดรส 161.246.6.140 พอร์ต 1351 เข้ากับเซิร์ฟเวอร์ หมายเลขไอพีแอดเดรส 161.246.4.3 พอร์ต 21 ซึ่งข้อมูลที่ผ่านในการเชื่อมต่อนี้สามารถดักจับและอ่านได้ทันที โดยสามารถดักจับและอ่านได้ทั้งสองทิศทางการส่ง จากรูป ผู้ดักจับสามารถนำเอาชื่อผู้ใช้และรหัสผ่าน s1014031 และ kit031 ตามลำดับ ไปใช้ในทางมิชอบได้ ทั้งนี้ผู้ใช้งานก็จะสูญเสียความเป็นส่วนตัวในการใช้งานไปด้วย เพราะผู้ดักจับสามารถล่วงรู้การใช้งานต่างๆของผู้ใช้ได้หมด เช่นในตัวอย่างนี้ ผู้ดักจับจะทราบว่าผู้ใช้ได้เข้ามาดาวน์โหลดไฟล์ .plan ไป โดยทราบได้จากคำสั่งเอฟทีพี RETR .plan



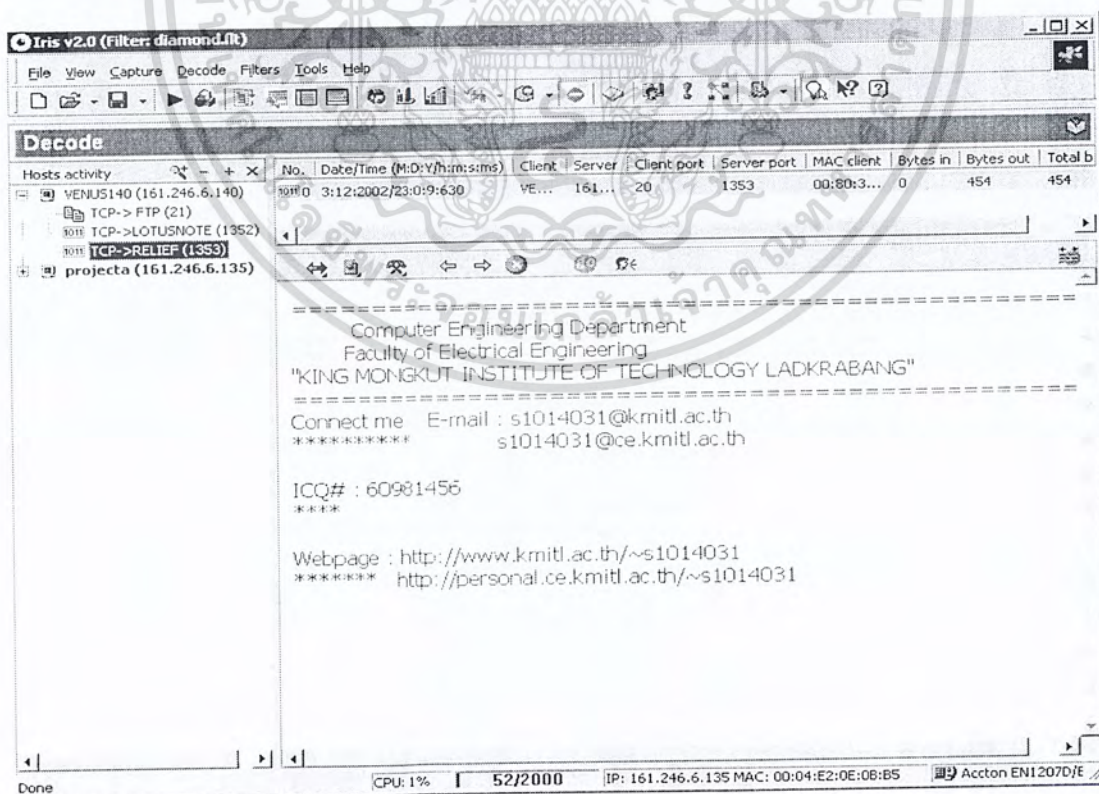
รูปที่ 12-2 แสดงข้อมูลในการเชื่อมต่อส่วนควบคุมของการถ่ายโอนแฟ้มข้อมูลที่ดักจับได้

ยิ่งไปกว่านั้น ในส่วนการเชื่อมต่อข้อมูล (data connection) นั้นก็สามารถดักจับได้เช่นกัน จากรูปที่ 12-2 จะพบว่าคำสั่ง PORT 161,256,6,140,5,72 ที่ส่งจากโปรแกรมฝั่งไคลเอ็นต์ ทำการร้องขอสร้างการเชื่อมต่อข้อมูลกับพอร์ตที่ 1352 ของไคลเอ็นต์ (คำนวณได้จากหมายเลขในการร้องขอ คือ $(256 \times 5) + 72 = 1352$) ทำให้เกิดการเชื่อมต่อข้อมูล จากนั้นโปรแกรมฝั่งไคลเอ็นต์จะนำส่วนการเชื่อมต่อข้อมูลนี้ในการร้องขอให้โปรแกรมบนฝั่งเซิร์ฟเวอร์ส่งรายการชื่อไฟล์มา ด้วยการส่งคำสั่ง NLST ไปให้โปรแกรมฝั่งเซิร์ฟเวอร์ผ่านส่วนเชื่อมต่อควบคุม รูปที่ 12-3 แสดงข้อมูลที่ถูกดักจับได้ที่ส่วนการเชื่อมต่อข้อมูล ส่งผลให้ผู้ดักจับสามารถล่วงรู้รายการไฟล์ของผู้ใช้ได้

ในกรณีคล้ายกัน ข้อมูลไฟล์ .plan ที่ผู้ใช้ร้องขอถ่ายโอนจากเซิร์ฟเวอร์ด้วยการที่โปรแกรมฝั่งไคลเอ็นต์ส่งคำสั่ง RETR .plan ก็สามารถถูกดักจับและอ่านได้ดังรูปที่ 12-4

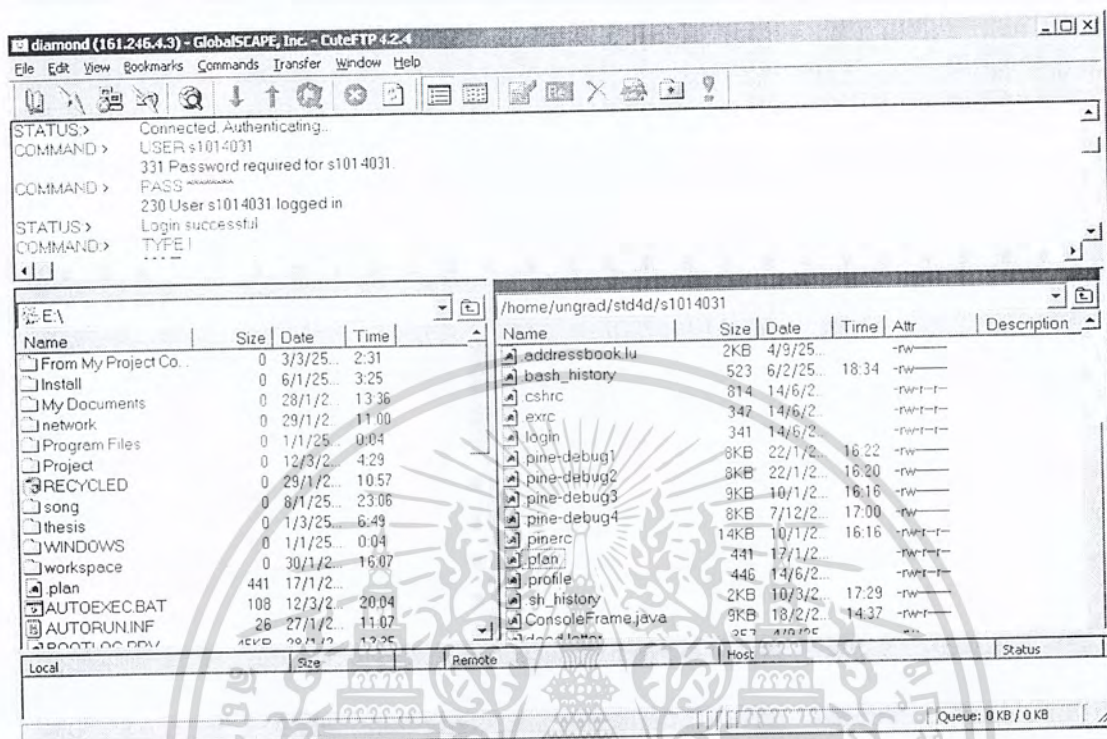


รูปที่ 12-3 แสดงรายการไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนแฟ้มข้อมูล

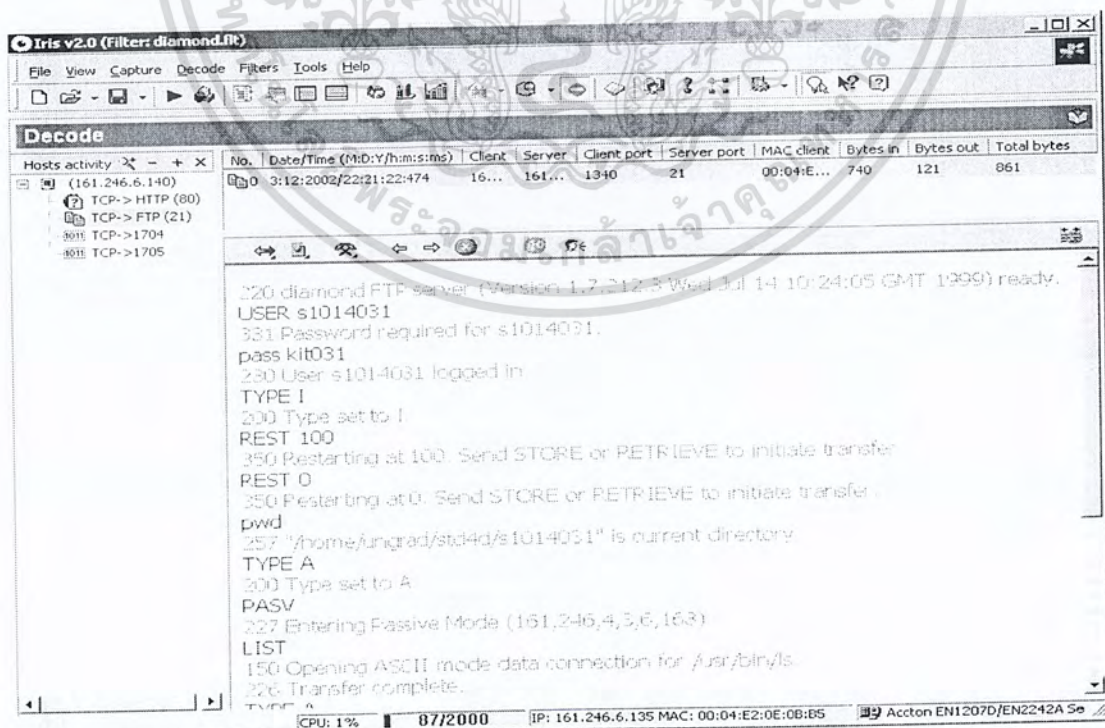


รูปที่ 12-4 แสดงข้อมูลไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนแฟ้มข้อมูล

การถ่ายโอนเพิ่มข้อมูลที่ไม่มีการเข้ารหัสแบบโหมดกราฟิก ก็สามารถถูกดักจับข้อมูล อธิบายได้ คล้ายกันกับการถ่ายโอนเพิ่มข้อมูลที่ไม่มีการเข้ารหัสแบบเท็กซ์โหมด ดังรูปที่ 12-5, 12-6, 12-7 และ 12-8 ตามลำดับ



รูปที่ 12-5 โปรแกรมออฟทีพีบีแบบกราฟิก CuteFTP



รูปที่ 12-6 แสดงข้อมูลในการเชื่อมต่อส่วนควบคุมของการถ่ายโอนเพิ่มข้อมูลที่ถูกจับได้

Iris v2.0 (Filter: diamond.ft)

File View Capture Decode Filters Tools Help

Decode

Hosts activity (161.246.6.140) TCP->HTTP (80) TCP->FTP (21) TCP->1704 TCP->1705

No.	Date/Time (M:D:Y/h:m:s)	Client	Server	Client port	Server port	MAC client	Bytes in	Bytes out	Total bytes
10110	3:12:2002/22:20:35:296	16...	161...	1341	1704	00:04:E...	2106	0	2106

total 6632

```

1 s1014031 std4d 64 Sep 4 2001 .addressbook
1 s1014031 std4d 2399 Sep 4 2001 .addressbook.lu
1 s1014031 std4d 523 Feb 6 18:34 .bash_history
1 s1014031 std4d 314 Jun 14 1999 .cshrc
1 s1014031 std4d 347 Jun 14 1999 .exrc
1 s1014031 std4d 341 Jun 14 1999 .login
1 s1014031 std4d 9131 Jan 22 16:22 .pine-debug1
1 s1014031 std4d 9131 Jan 22 16:20 .pine-debug2
1 s1014031 std4d 9335 Jan 10 16:16 .pine-debug3
1 s1014031 std4d 9703 Dec 7 17:00 .pine-debug4
1 s1014031 std4d 14803 Jan 10 16:16 .pinerc
1 s1014031 std4d 441 Jan 17 2000 .plan
1 s1014031 std4d 446 Jan 14 1999 .profile
1 s1014031 std4d 23531 Apr 18 17:29 .ssh_history
1 s1014031 std4d 1014 Aug 30 2001 .ssh
1 s1014031 std4d 3231 Sep 13 14:27 ConsoleFrame.java
1 s1014031 std4d 122493 Jan 20 14:24 MindtermSrc.rar
1 s1014031 std4d 116394 Feb 6 18:50 NWP3031full.zip
1 s1014031 std4d 3527 Feb 16 14 37 RS.zip
1 s1014031 std4d 4115 Feb 12 14 43 RemoteAgent.java
  
```

CPU: 1% 8772000 IP: 161.246.6.135 MAC: 00:04:E2:0E:0B:85 Accton EN1207D/EN2242A Se

รูปที่ 12-7 แสดงรายการไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนแฟ้มข้อมูล

Iris v2.0 (Filter: diamond.ft)

File View Capture Decode Filters Tools Help

Decode

Hosts activity (161.246.6.140) TCP->HTTP (80) TCP->FTP (21) TCP->1704 TCP->1705

No.	Date/Time (M:D:Y/h:m:s)	Client	Server	Client port	Server port	MAC client	Bytes in	Bytes out	Total bytes
10110	3:12:2002/22:21:14:403	16...	161...	1344	1705	00:04:E...	441	0	441

```

=====
Computer Engineering Department
Faculty of Electrical Engineering
"KING MONKJIT INSTITUTE OF TECHNOLOGY LADYRABANG"
=====
Connect me E-mail : s1014031@kmitl.ac.th
s1014031@ce.kmitl.ac.th

ICQ# : 60981456

Webpage : http://www.kmitl.ac.th/~s1014031
http://personal.ce.kmitl.ac.th/~s1014031
  
```

CPU: 3% 8772000 IP: 161.246.6.135 MAC: 00:04:E2:0E:0B:85 Accton EN1207D/EN2242A Se

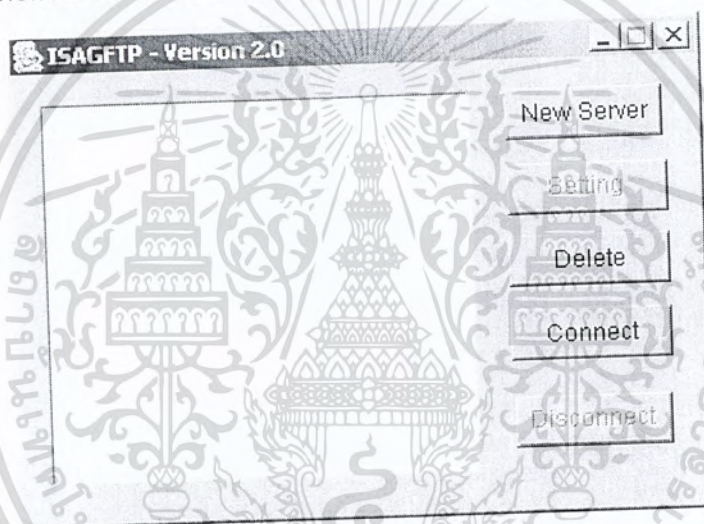
รูปที่ 12-8 แสดงข้อมูลไฟล์ที่ดักจับได้จากส่วนการเชื่อมต่อข้อมูลของการถ่ายโอนแฟ้มข้อมูล

12.2.2 การทดลองใช้งานโปรแกรม IsagFTP

โปรแกรม IsagFTP นี้มุ่งหวังให้เกิดความสะดวกในการใช้งานมากที่สุด ด้วยการเลือกเวอร์ชัน โพรโทคอลเอสเอสเอชโดยอัตโนมัติ ทำให้ผู้ใช้ไม่จำเป็นต้องเข้าใจการทำงานและจดจำความสามารถในการรองรับเวอร์ชันของโพรโทคอลของแต่ละเซิร์ฟเวอร์ที่ตนติดต่ออยู่ โดยหากเซิร์ฟเวอร์รองรับ โพรโทคอลเอสเอสเอชเวอร์ชัน 2 ซึ่งมีประสิทธิภาพในการถ่ายโอนแฟ้มข้อมูลมากกว่า ก็จะใช้โพรโทคอลเวอร์ชัน 2 นี้ แต่หากไม่รองรับ ก็จะทำการตรวจสอบความสามารถในการรองรับโพรโทคอลเอสเอสเอชเวอร์ชัน 1ต่อไป

ขั้นตอนทดลอง

1. เมื่อรันโปรแกรมจะพบกับวินโดว์ไซด์เมเนเจอร์ที่ทำการสร้างและจัดเก็บเก็บค่าคอนฟิกของแต่ละเซิร์ฟเวอร์ได้ เพื่อความสะดวกในการใช้งานครั้งต่อไป ดังรูปที่ 12-9



รูปที่ 12-9 ไซด์เมเนเจอร์ของโปรแกรม IsagFTP

2. ทำการคลิกปุ่ม New Server เพื่อเข้าสู่วินโดว์กำหนดค่าคอนฟิกในการติดต่อใหม่ ดังรูปที่ 12-10 โดยในที่นี้จะทำการทดลองติดต่อกับเซิร์ฟเวอร์ diamond ของภาควิชาวิศวกรรมคอมพิวเตอร์ ซึ่งในขณะที่ทดลองใช้งานนี้ เซิร์ฟเวอร์ diamond ของภาควิชาวิศวกรรมคอมพิวเตอร์นี้ มีความสามารถในการรองรับโพรโทคอลเวอร์ชัน 2

จะพบว่าเราสามารถเลือกอัลกอริทึมในการแลกเปลี่ยนคีย์ได้ 2 แบบคือ อัลกอริทึม RSA และอัลกอริทึม DSS ดังรูปที่ 12-11 ซึ่งโพรโทคอลเอสเอสเอช 1 จะรองรับเพียงอัลกอริทึม RSA เท่านั้น แต่หากผู้ใช้เลือกใช้อัลกอริทึม DSS ในการแลกเปลี่ยนคีย์กับเซิร์ฟเวอร์ที่รองรับเพียงโพรโทคอลเอสเอสเอช 1 โปรแกรมจะทำการแก้ไขการเลือกอัลกอริทึมให้โดยอัตโนมัติ

ISAGFTP - New Server Dialog

Server name: diamond

Address: 161.246.4.3 Port: 22

Username: s1014031 Password: *****

Key algorithm: ssh-dss

Transport Pref.	Client to Server	Server to Client
Cipher	3des-cbc	3des-cbc
Mac	hmac-sha1	hmac-sha1
Compression	none	none

Accept Cancel

รูปที่ 12-10 วินโดว์กำหนดค่าคอนฟิกการติดต่อกับเซิร์ฟเวอร์

ISAGFTP - New Server Dialog

Server name: diamond

Address: 161.246.4.3 Port: 22

Username: s1014031 Password: *****

Key algorithm: ssh-rsa

Transport Pref.	Client to Server	Server to Client
Cipher	ssh-rsa	3des-cbc
Mac	hmac-sha1	hmac-sha1
Compression	none	none

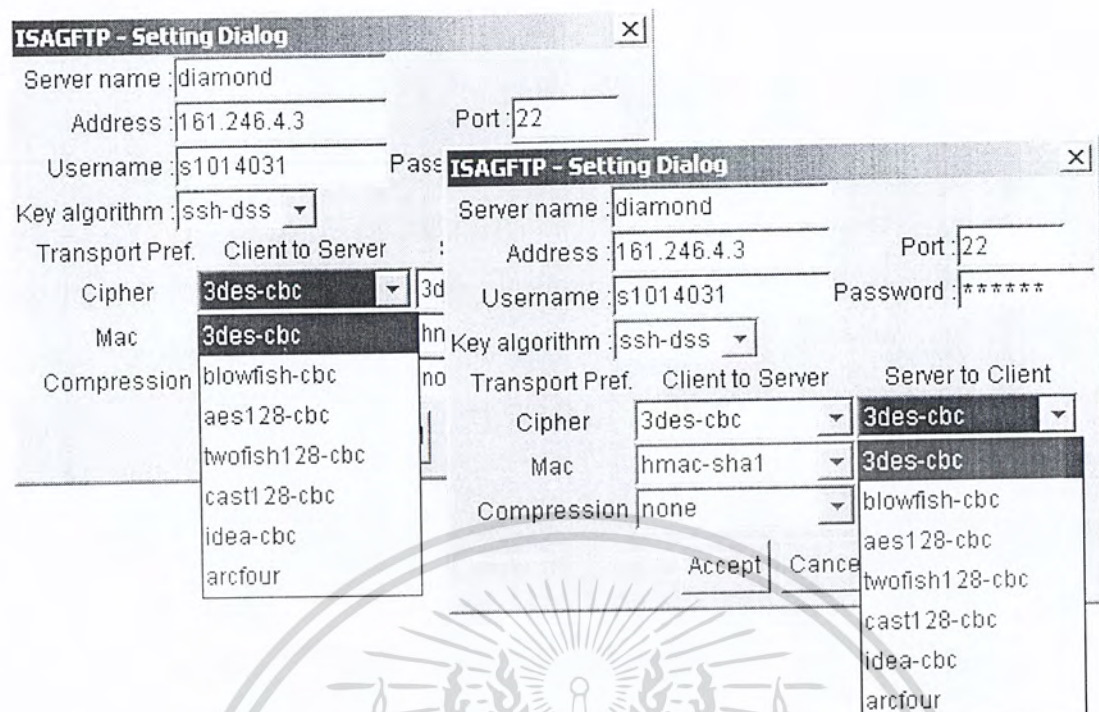
Accept Cancel

รูปที่ 12-11 แสดงอัลกอริทึมการแลกเปลี่ยนคีย์ที่สามารถเลือกใช้ได้

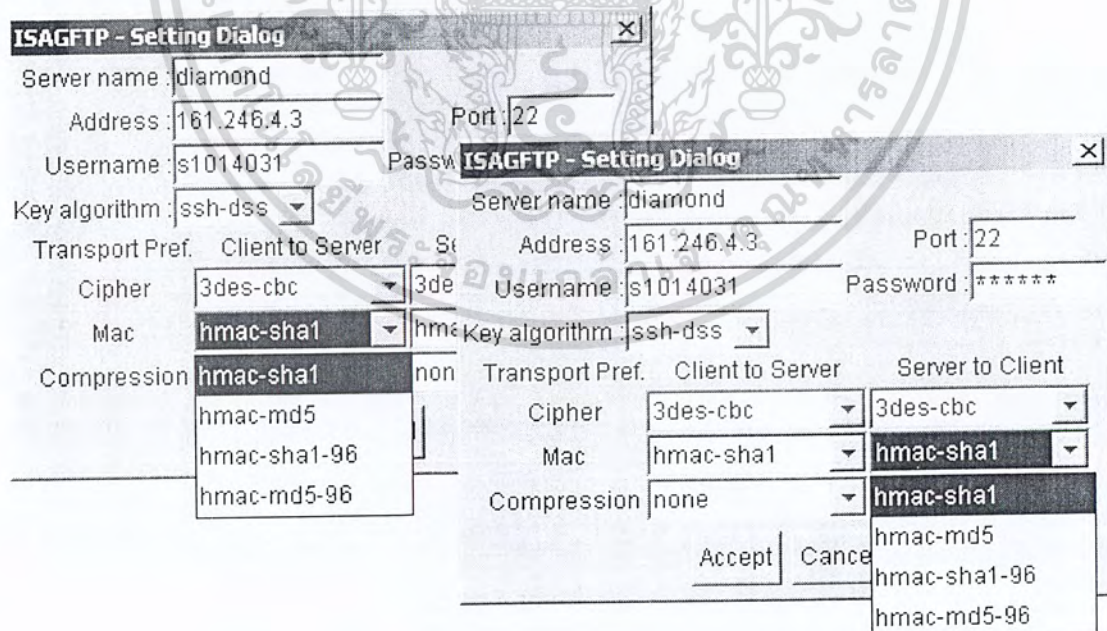
เราสามารถเลือกอัลกอริทึมการเข้ารหัสที่รองรับตามโพรโตคอลได้ โดยอัลกอริทึมการเข้ารหัสนี้ไม่ขึ้นกับทิศทางการส่งข้อมูลระหว่างไคลเอนต์กับเซิร์ฟเวอร์ เป็นอิสระต่อกัน ทำให้ฝั่งไคลเอนต์และเซิร์ฟเวอร์สามารถเลือกอัลกอริทึมที่ต่างกัน และเหมาะสมกับสภาวะการณ์ของแต่ละฝั่ง ดังรูปที่ 12-12

เช่นเดียวกับอัลกอริทึมการเข้ารหัส อัลกอริทึมแฮชที่ใช้ในการตรวจสอบความสมบูรณ์ของข้อมูลและอัลกอริทึมบีบอัดก็สามารถเลือกใช้ตามความต้องการ และเป็นอิสระต่อกันอีกด้วย ดังรูปที่ 12-13 และรูปที่ 12-14 ตามลำดับ

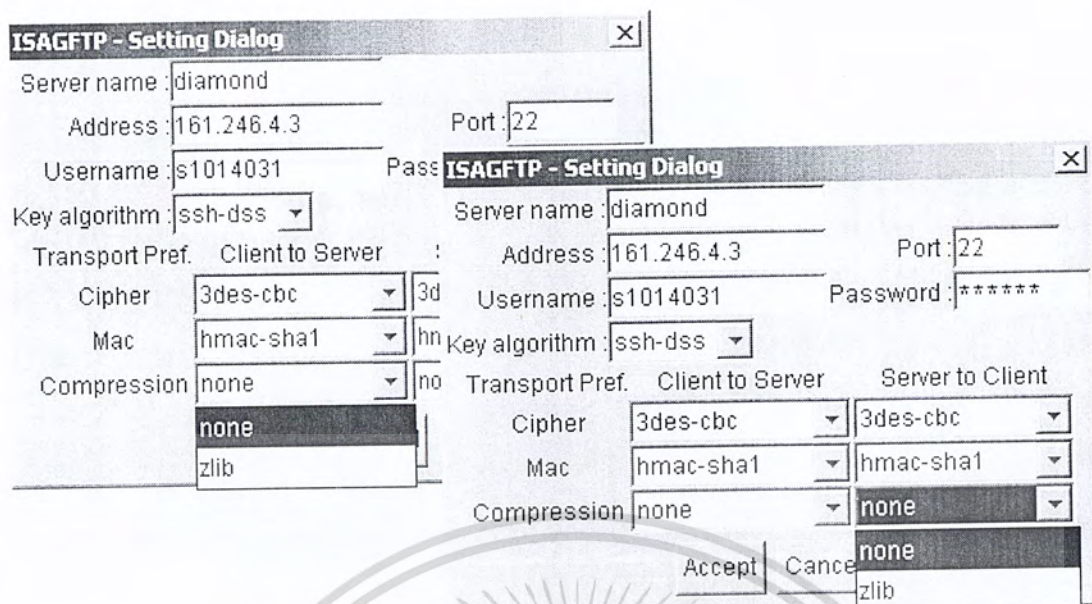
อัลกอริทึมทั้งสองนี้สามารถเลือกได้ตามความเหมาะสม แต่เพื่อความสะดวกในการใช้งาน หากอัลกอริทึมที่เลือกไม่รองรับโดยทั้งสองฝั่งผู้ติดต่อกัน อัลกอริทึมที่ใช้จริงจะถูกเลือกโดยอัตโนมัติ ตามข้อกำหนดของโพรโตคอลเอสเอสเอสซึ่งได้กล่าวมาแล้ว



รูปที่ 12-12 แสดงอัลกอริทึมการเข้ารหัสที่รองรับ และความเป็นอิสระต่อทิศทาง



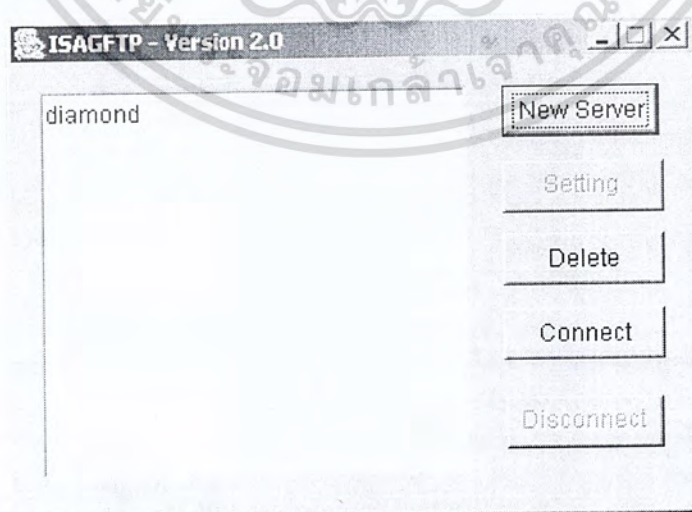
รูปที่ 12-13 แสดงอัลกอริทึมแฮชตรวจสอบความสมบูรณ์ข้อมูลที่รองรับ และความเป็นอิสระต่อทิศทาง



รูปที่ 12-14 แสดงอัลกอริทึมการบีบอัดที่รองรับ และความเป็นอิสระต่อกิตทาง

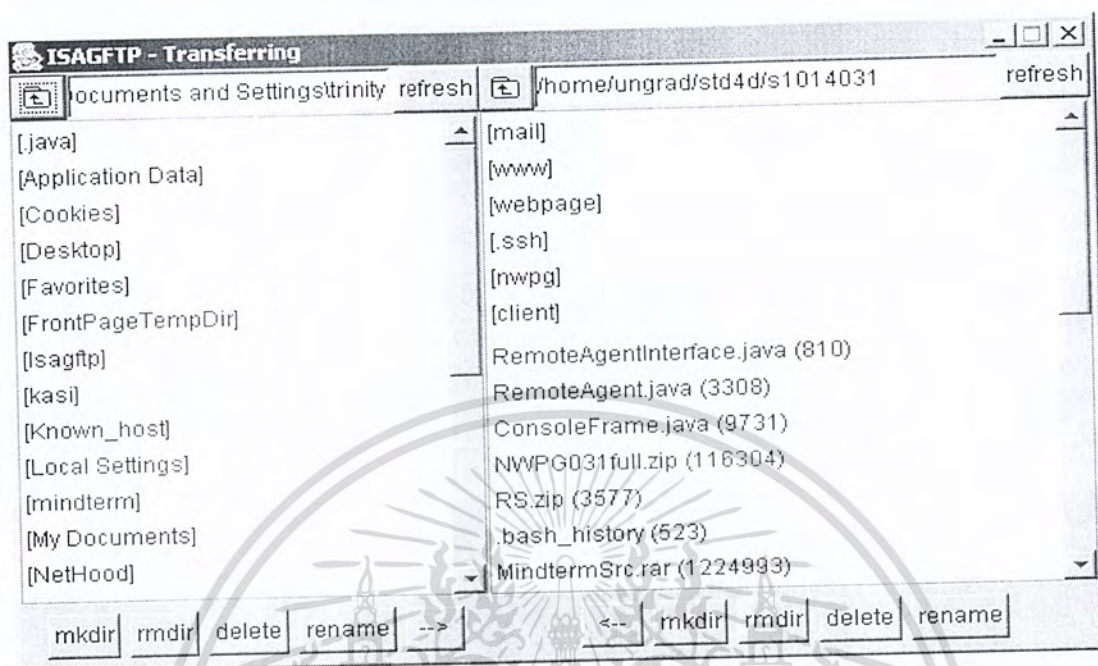
สังเกตว่าอัลกอริทึมการบีบอัดนั้นมีเพียงอัลกอริทึม zlib เท่านั้นที่รองรับ เนื่องจากมีเพียงอัลกอริทึมบีบอัดที่รองรับตามข้อกำหนดของโพรโตคอลเอสเอสเอสเอช จึงทำให้สามารถเลือกได้เพียงจะทำการบีบอัดด้วยอัลกอริทึมนี้หรือไม่เท่านั้น

- เมื่อเรากำหนดค่าทั้งหมดแล้ว จะกลับเข้าสู่ไซด์เมนเนเจอร์อีกครั้ง พร้อมกับรายการเซิร์ฟเวอร์ที่กำหนดเพิ่มขึ้นมาใหม่ ดังรูปที่ 12-15



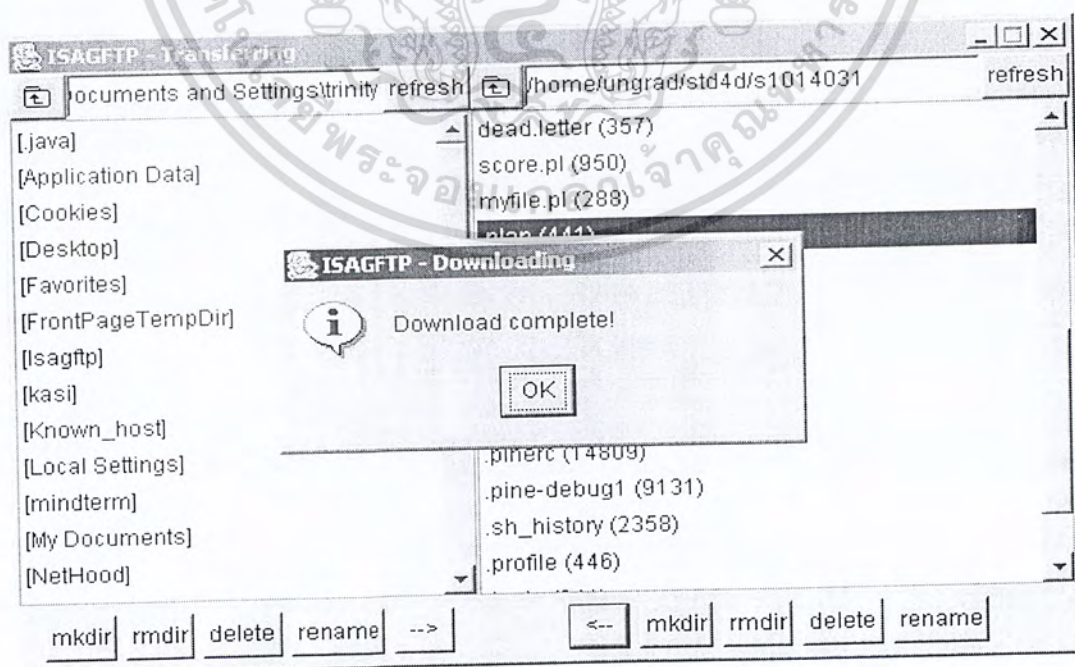
รูปที่ 12-15 ไซด์เมนเนเจอร์ที่ได้รับการกำหนดเซิร์ฟเวอร์ใหม่เพิ่ม

4. ทำการเลือกเซิร์ฟเวอร์ที่จะติดต่อจากรายการ และคลิกที่ปุ่ม Connect เพื่อทำการเชื่อมต่อ หากสำเร็จ จะขึ้นวินโดว์ในการถ่ายโอนแฟ้มข้อมูลดังรูปที่ 12-16



รูปที่ 12-15 วินโดว์สำหรับการถ่ายโอนแฟ้มข้อมูลของโปรแกรม IsagFTP

5. ทำการถ่ายโอนแฟ้มข้อมูลที่ต้องการ โดยสามารถทำการถ่ายโอนแฟ้มข้อมูลเป็นไดเรกทอรีได้ด้วย หากการถ่ายโอนแฟ้มข้อมูลในแต่ละครั้งสำเร็จ จะมีไดอะล็อกบ็อกซ์ขึ้นดังรูปที่ 12-16



รูปที่ 12-16 ไดอะล็อกบ็อกซ์แสดงความสำเร็จในการถ่ายโอนแฟ้มข้อมูล

หากผู้ใช้ทำการรันโปรแกรมจากคอนโซล จะพบว่าเมื่อข้อความการทำงานสำหรับดื่บักการทำงาน ซึ่งมีการทำงานเป็นไปตามโปรโตคอลที่ได้กำหนดไว้ในโปรโตคอลเอสเอสเอช 1 และ โปรโตคอลเอสเอสเอช 2 ดังรูปที่ 12-17

```

C:\WINNT\System32\command.com
C:\>java ISAGFTP.isagftp.IsagFTP
ssh2
** SSH2Transport : 'peer's version is 'SSH-1.99-OpenSSH_2.5.2p2''
** SSH2Transport : 'disabling key re-exchange, not implemented in peer'
** SSH2Transport : 'peer kex algorithms: diffie-hellman-group-exchange-sha1, diffie-hellman-group1-sha1'
** SSH2Transport : 'peer host key algorithms: ssh-rsa, ssh-dss'
** SSH2Transport : 'peer enc. alg. cli2srv: aes128-cbc, 3des-cbc, blowfish-cbc, cast128-cbc, arcfour, aes192-cbc, aes256-cbc, rijndael128-cbc, rijndael192-cbc, rijndael128-cbc, rijndael-cbc@lysator.liu.se'
** SSH2Transport : 'peer enc. alg. srv2cli: aes128-cbc, 3des-cbc, blowfish-cbc, cast128-cbc, arcfour, aes192-cbc, aes256-cbc, rijndael128-cbc, rijndael192-cbc, rijndael128-cbc, rijndael-cbc@lysator.liu.se'
** SSH2Transport : 'peer mac alg. cli2srv: hmac-md5, hmac-sha1, hmac-ripemd160, hmac-ripemd160@openssh.com, hmac-sha1-96, hmac-md5-96'
** SSH2Transport : 'peer mac alg. srv2cli: hmac-md5, hmac-sha1, hmac-ripemd160, hmac-ripemd160@openssh.com, hmac-sha1-96, hmac-md5-96'
** SSH2Transport : 'peer comp. alg. cli2srv: none, zlib'
** SSH2Transport : 'peer comp. alg. srv2cli: none, zlib'
** SSH2Transport : 'our kex algorithms: diffie-hellman-group1-sha1'
** SSH2Transport : 'our host key algorithms: ssh-dss'
** SSH2Transport : 'our enc. alg. cli2srv: 3des-cbc'
** SSH2Transport : 'our enc. alg. srv2cli: 3des-cbc'
** SSH2Transport : 'our mac alg. cli2srv: hmac-sha1'
** SSH2Transport : 'our mac alg. srv2cli: hmac-sha1'
** SSH2Transport : 'our comp. alg. cli2srv: none'
** SSH2Transport : 'our comp. alg. srv2cli: none'
** SSH2Transport : 'KEX algorithm chosen: diffie-hellman-group1-sha1'
** SSH2Transport : 'same KEX guessed? false'
** SSH2Transport : 'first KEX follows? false'
** SSH2Transport : 'server's signature verified'
** SSH2Transport : 'new transmitter context (3des-cbc, hmac-sha1, none)'
** SSH2Transport : 'new receiver context (3des-cbc, hmac-sha1, none)'
** SSH2UserAuth : 'server accepted: ssh-userauth'
** SSH2UserAuth : 'failure continuation: publickey, password (partial: false)'
** SSH2UserAuth : 'our remaining methods: password'
** SSH2UserAuth : 'trying method: password'
** SSH2UserAuth : 'successful authentication with password'
** SSH2Channel : 'open confirmation, ch. #0, init-winsz = 0, max-pktsz = 16384'
** SSH2Transport : 'disconnect: Disconnect by user'

```

รูปที่ 12-17 ข้อความดื่บักการทำงานของโปรแกรม IsagFTP

12.2.3 การทำงานในระบบปฏิบัติการต่างๆ

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

บทที่ 13

บทวิจารณ์และสรุป

ในการพัฒนาโปรแกรมรับส่งเพิ่มข้อมูลแบบปลอดภัยนี้ ได้นำโพรโทคอลเอสเอสเอชมาประยุกต์ใช้ เพื่อสร้างความปลอดภัยในการติดต่อสื่อสาร โดยรองรับโพรโทคอลเอสเอสเอชทั้งเวอร์ชัน 1 และ 2 ซึ่งในส่วนการรองรับเวอร์ชัน 1 นั้น ใช้หลักการการฟอร์เวิร์ดพอร์ต โดยได้สร้างโปรแกรมในส่วนของการรับส่งข้อมูล ตามมาตรฐานของโพรโทคอลเอฟทีพี ที่ได้มีการกำหนดไว้ในอาร์เอฟซี 959 (RFC 959) ส่วนการรองรับเวอร์ชัน 2 นั้น จะพัฒนาเพื่อใช้งานร่วมกับเอสเอฟทีพี จากการศึกษาและพัฒนาโปรแกรมมาพอจะวิจารณ์และสรุปได้ดังนี้

13.1 การวิเคราะห์และสรุปมาตรฐานของโพรโทคอลเอสเอสเอช

ในปัจจุบันการรักษาความปลอดภัยของข้อมูลเป็นประเด็นสำคัญ ในการติดต่อสื่อสารของระบบเครือข่าย ถึงแม้ว่าโพรโทคอลเอสเอสเอชยังไม่ได้เป็นโพรโทคอลมาตรฐาน ทางด้านความปลอดภัย แต่ก็ยังเป็นโพรโทคอลที่มีคนนิยมใช้เป็นจำนวนมากจนได้รับเป็นมาตรฐานดีแฟกโต (De facto Standard) โปรแกรมประยุกต์บนทีซีพี/ไอพีที่ต้องการความปลอดภัยของข้อมูลจะเลือกใช้ และเซิร์ฟเวอร์ส่วนใหญ่ มักจะติดตั้งโปรแกรมเอสเอสเอชไว้สำหรับผู้ใช้ที่ต้องการรักษาความปลอดภัยและความเป็นส่วนตัวของตน ข้อดีของโพรโทคอลเอสเอสเอช มีดังนี้

- การปลอมแปลงไอพี ไม่สามารถกระทำได้ เนื่องจากมีการตรวจสอบการใช้งานของผู้ใช้มีความปลอดภัยสูง โดยการเข้ารหัส คีย์โฮสต์และ ลูกที่ที่สัมพันธ์ป้องกันการปลอมแปลงไอพี
- ข้อมูลผู้ใช้และรหัสผ่านของผู้ใช้นั้นจะมีความปลอดภัยในการส่งที่สูงเนื่องจากการเข้ารหัสที่มีประสิทธิภาพสูงด้วยกรรมวิธีของ อาร์เอสเอ (RSA) ในเวอร์ชัน 1 และ ดีเอสเอส (DSS) ในเวอร์ชัน 2 ทำให้ยากต่อการถอดรหัส
- ข้อมูลที่ติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์นั้นมีการเข้ารหัสที่หลากหลายตามความต้องการของผู้ใช้ฝั่งไคลเอ็นต์ ทำให้ยากต่อการถอดรหัสข้อมูลที่ดักจับมาได้เป็นอย่างมาก
- การโจมตีของบุคคลระหว่างกลาง (Man in the middle attack) ไม่สามารถทำได้เนื่องจากการใช้ คีย์โฮสต์ และคีย์เซสชัน

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

13.2 การวิเคราะห์และสรุปมาตรฐานของโปรแกรมรับส่งเพิ่มข้อมูลแบบปลอดภัย (IsagFTP)

ตัวโปรแกรมนั้นมีการทำงานเป็นแบบกราฟิก ทำให้สะดวกต่อผู้ใช้งาน และลดเวลาการเรียนรู้การใช้งานอีกด้วย อีกทั้งยังทำให้ผู้ใช้ที่ไม่มีความรู้เกี่ยวข้องกับความซับซ้อน หรือรู้เพียงเล็กน้อย สามารถใช้งานได้ดีเหมือนเช่นผู้ที่มีความรู้ การพัฒนาโปรแกรมนั้นแบ่งออกเป็น 3 ส่วนใหญ่ ๆ คือ ส่วนที่ทำตามขั้นตอนมาตรฐานของโปรโตคอลเช็กเกียและส่วนจัดการรับส่งข้อมูล

- การพัฒนาส่วนรองรับโปรโตคอลเอสเอสเอช 1
- การพัฒนาส่วนรองรับโปรโตคอลเอสเอสเอช 2
- การพัฒนาส่วนติดต่อกับผู้ใช้งาน เป็นแบบกราฟิก

13.2.1 ส่วนรองรับโปรโตคอลเอสเอสเอช 1

เนื่องจากการพัฒนาในส่วนนี้จะใช้หลักการฟอร์เวิร์ดพอร์ต การพัฒนาจึงประกอบด้วยการสร้างส่วนจัดการการถ่ายโอนเพิ่มข้อมูลปกติ (โปรแกรมเอฟทีที) และการสร้างส่วนจัดการตามขั้นตอนของโปรโตคอลเอสเอสเอช 1

13.2.1.1 ส่วนจัดการการถ่ายโอนเพิ่มข้อมูล

ได้ออกแบบตามมาตรฐานของโปรโตคอลเอฟทีทีที่ระบุไว้ในอาร์เอฟซี 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-ดาวน์โหลดเป็นไคลเอนต์ การเลือกไฟล์เป็นกลุ่มเพื่ออัปโหลด-ดาวน์โหลด การลบทั้งไคลเอนต์ เป็นต้น

13.2.1.2 ส่วนจัดการตามขั้นตอนของโปรโตคอลเอสเอสเอช 1

ส่วนนี้จะเหมือนโปรแกรมไคลเอนต์ที่ติดต่อไปยังเซิร์ฟเวอร์ที่ให้บริการโปรโตคอลเอสเอสเอช 1 ซึ่งออกแบบตามอินเทอร์เน็ต-ดราฟต์ ที่เกี่ยวข้องกับโปรโตคอลเอสเอสเอช 1 แต่ยังมีข้อจำกัดหลายประการ คือ

- การเข้ารหัสในโปรแกรมยังไม่ครอบคลุมอัลกอริทึมการเข้ารหัสทั้งหมดของเช็กเกียเซิร์ฟเวอร์ที่มีให้บริการ แต่ได้ออกแบบไว้รองรับการเข้ารหัสประเภทอื่น ๆ แล้ว เพียงแต่นำคลาสของการเข้ารหัสประเภทนั้นมาเรียกใช้
- มีความช้าในการเข้ารหัสส่วนข้อมูล เนื่องจากโปรโตคอลเช็กเกียเซิร์ฟเวอร์ชั้น 1 นั้น ไม่ได้ออกแบบมาเพื่อการเข้ารหัสข้อมูลที่มีขนาดใหญ่ (ซึ่งมีในเวอร์ชัน 2)

13.2.2 การพัฒนาส่วนรองรับโพรโตคอลเอสเอสเอช 2

ส่วนนี้จะทำให้โปรแกรมสามารถทำงานร่วมกับโพรโตคอลเอสเอสเอช 2 ของเซิร์ฟเวอร์ได้ ซึ่งออกแบบตามอินเทอร์เน็ต-ดราฟต์ ที่เกี่ยวข้องกับโพรโตคอลเอสเอสเอช 2 สดข้อจำกัดหลายประการ คือ การคิดเงื่อนไขตามสัทธิตับตรบนัลลอกริทึมคีย์สาธารณะ RSA ที่ใช้ในโพรโตคอลเอสเอสเอชเวอร์ชัน 1 ใช้งานด้วยเอสเอฟทีพี ซึ่งออกแบบมารองรับการเข้ารหัสข้อมูลที่มีขนาดใหญ่

13.3 แนวทางการพัฒนาต่อไปในอนาคต

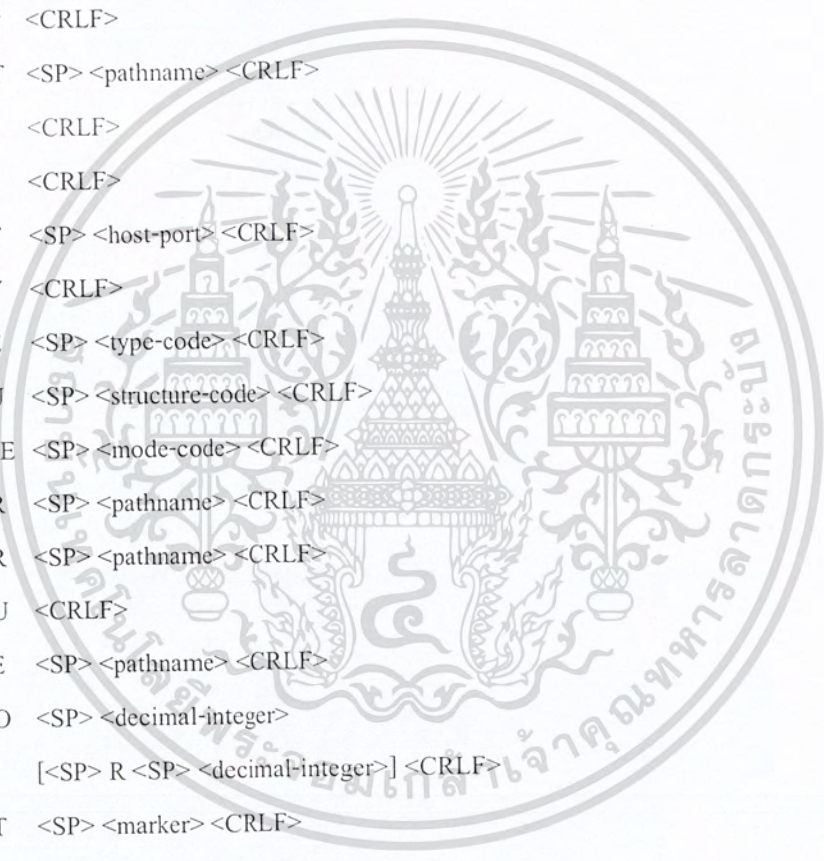
- เพิ่มอัลกอริทึมการเข้ารหัสข้อมูลที่ยังไม่ครบถ้วน เพื่อให้ยากแก่การนำข้อมูลที่ลักลอบดักจับได้ไปทำการถอดรหัส
- สร้างฟังก์ชันในการใช้งานบนอินเทอร์เน็ตเพื่อบริการที่มากขึ้น เพื่อความสะดวกในการใช้งานของแต่ละผู้ใช้
- จัดจํารูปแบบการใช้งานของผู้ใช้ เพื่อให้โปรแกรมเรียนรู้ และช่วยผู้ใช้งานได้
- พัฒนาให้มีการเลือกเข้ารหัสเฉพาะการเชื่อมต่อส่วนควบคุม เพื่อความรวดเร็วในการโอนย้ายข้อมูล ในการเชื่อมต่อส่วนข้อมูล สำหรับผู้ที่ยอมรับความเสี่ยงในการดักจับเฉพาะเนื้อข้อมูล แต่ต้องการป้องกันการดักจับรายชื่อผู้ใช้และรหัสผ่าน



FTP COMMANDS

Commands

The following are the FTP commands:



```

USER  <SP> <username> <CRLF>
PASS  <SP> <password> <CRLF>
ACCT  <SP> <account-information> <CRLF>
CWD   <SP> <pathname> <CRLF>
CDUP  <CRLF>
SMNT  <SP> <pathname> <CRLF>
QUIT  <CRLF>
REIN  <CRLF>
PORT  <SP> <host-port> <CRLF>
PASV  <CRLF>
TYPE  <SP> <type-code> <CRLF>
STRU  <SP> <structure-code> <CRLF>
MODE  <SP> <mode-code> <CRLF>
RETR  <SP> <pathname> <CRLF>
STOR  <SP> <pathname> <CRLF>
STOU  <CRLF>
APPE  <SP> <pathname> <CRLF>
ALLO  <SP> <decimal-integer>
      [<SP> R <SP> <decimal-integer>] <CRLF>
REST  <SP> <marker> <CRLF>
RNFR  <SP> <pathname> <CRLF>
RNT0  <SP> <pathname> <CRLF>
ABOR  <CRLF>
DELE  <SP> <pathname> <CRLF>
RMD   <SP> <pathname> <CRLF>
MKD   <SP> <pathname> <CRLF>
PWD   <CRLF>
LIST  [<SP> <pathname>] <CRLF>
NLST  [<SP> <pathname>] <CRLF>

```



```

SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>

```

Arguments

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

```

<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and <LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any ASCII code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number> ::= any decimal integer 1 through 255
<form-code> ::= N | T | C
<type-code> ::= A [<sp> <form-code>]
                | E [<sp> <form-code>]
                | I
                | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code> ::= S | B | C
<pathname> ::= <string>
<decimal-integer> ::= any decimal integer

```

ACCESS CONTROL COMMANDS

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

USER NAME (USER)

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

PASSWORD (PASS)

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

ACCOUNT (ACCT)

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

There are reply codes to differentiate these cases for the automation: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532

reply depending on whether it stores (pending receipt of the ACCounT command) or discards the command, respectively.

CHANGE WORKING DIRECTORY (CWD)

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

CHANGE TO PARENT DIRECTORY (CDUP)

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

STRUCTURE MOUNT (SMNT)

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

REINITIALIZE (REIN)

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

LOGOUT (QUIT)

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

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

TRANSFER PARAMETER COMMANDS

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

DATA PORT (PORT)

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

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

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

PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter

is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:

A – ASCII	}	{	N - Non-print
E – EBCDIC			T - Telnet format effectors
I – Image			C - Carriage Control
L <byte size> - Local byte Byte size			

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

FILE STRUCTURE (STRU)

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

The following codes are assigned for structure:

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

The default structure is File.

TRANSFER MODE (MODE)

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

The following codes are assigned for transfer modes:

S - Stream

B - Block

C - Compressed

The default transfer mode is Stream.

FTP SERVICE COMMANDS

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

RETRIEVE (RETR)

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

STORE (STOR)

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

STORE UNIQUE (STOU)

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

APPEND (with create) (APPE)

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

ALLOCATE (ALLO)

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

RESTART (REST)

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

RENAME FROM (RNFR)

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

RENAME TO (RNTO)

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

ABORT (ABOR)

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

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

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

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

DELETE (DELE)

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

REMOVE DIRECTORY (RMD)

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

MAKE DIRECTORY (MKD)

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

PRINT WORKING DIRECTORY (PWD)

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

LIST (LIST)

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

NAME LIST (NLST)

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

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the

protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

SYSTEM (SYST)

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

STATUS (STAT)

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

HELP (HELP)

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

NOOP (NOOP)

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

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

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

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

1. User system inserts the Telnet "Interrupt Process" (IP) signal in the Telnet stream.
2. User system sends the Telnet "Synch" signal.
3. User system inserts the command (e.g., ABOR) in the Telnet stream.
4. Server PI, after receiving "IP", scans the Telnet stream for EXACTLY ONE FTP command.

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



FTP REPLIES

Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNT0. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

A reply is defined to contain the 3-digit code, followed by Space <SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the Telnet end-of-line code. There will be cases however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the control connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions, it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus),

followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code.

For example:

123-First line

Second line

234 A line beginning with numbers

123 The last line

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram), an unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g., RNTD command without a preceding RNFR).

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply per command.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

- x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.
- x1z Information - These are replies to requests for information, such as status or help.
- x2z Connections - Replies referring to the control and data connections.
- x3z Authentication and accounting - Replies for the login process and accounting procedures.
- x4z Unspecified as yet.
- x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can

proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

Reply Codes by Function Groups

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

- Logged out if appropriate.
- 421 Service not available, closing control connection.
- This may be a reply to any command if the service knows it must shut down.
- 125 Data connection already open; transfer starting.
- 225 Data connection open; no transfer in progress.
- 425 Can't open data connection.
- 226 Closing data connection.
- Requested file action successful (for example, file transfer or file abort).
- 426 Connection closed; transfer aborted.
- 227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
- 230 User logged in, proceed.
- 530 Not logged in.
- 331 User name okay, need password.
- 332 Need account for login.
- 532 Need account for storing files.
- 150 File status okay; about to open data connection.
- 250 Requested file action okay, completed.
- 257 "PATHNAME" created.
- 350 Requested file action pending further information.
- 450 Requested file action not taken.
- File unavailable (e.g., file busy).
- 550 Requested action not taken.
- File unavailable (e.g., file not found, no access).
- 451 Requested action aborted. Local error in processing.
- 551 Requested action aborted. Page type unknown.
- 452 Requested action not taken.
- Insufficient storage space in system.
- 552 Requested file action aborted.
- Exceeded storage allocation (for current directory or dataset).
- 553 Requested action not taken.
- File name not allowed.

Numeric Order List of Reply Codes

110 Restart marker reply.

In this case, the text is exact and not left to the particular implementation; it must read:

MARK yyyy = mmmm

Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").

120 Service ready in nnn minutes.

125 Data connection already open; transfer starting.

150 File status okay; about to open data connection.

200 Command okay.

202 Command not implemented, superfluous at this site.

211 System status, or system help reply.

212 Directory status.

213 File status.

214 Help message.

On how to use the server or the meaning of a particular non-standard command.
This reply is useful only to the human user.

215 NAME system type.

Where NAME is an official system name from the list in the Assigned Numbers document.

220 Service ready for new user.

221 Service closing control connection.

Logged out if appropriate.

225 Data connection open; no transfer in progress.

226 Closing data connection.

Requested file action successful (for example, file transfer or file abort).

227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.

250 Requested file action okay, completed.

257 "PATHNAME" created.

331 User name okay, need password.

332 Need account for login.

- 350 Requested file action pending further information.
- 421 Service not available, closing control connection.
This may be a reply to any command if the service knows it must shut down.
- 425 Can't open data connection.
- 426 Connection closed; transfer aborted.
- 450 Requested file action not taken.
File unavailable (e.g., file busy).
- 451 Requested action aborted: local error in processing.
- 452 Requested action not taken.
Insufficient storage space in system.
- 500 Syntax error, command unrecognized.
This may include errors such as command line too long.
- 501 Syntax error in parameters or arguments.
- 502 Command not implemented.
- 503 Bad sequence of commands.
- 504 Command not implemented for that parameter.
- 530 Not logged in.
- 532 Need account for storing files.
- 550 Requested action not taken.
File unavailable (e.g., file not found, no access).
- 551 Requested action aborted: page type unknown.
- 552 Requested file action aborted.
Exceeded storage allocation (for current directory or dataset).
- 553 Requested action not taken.
File name not allowed.



SSH1 Packet Descriptor

ในการติดต่อกับเซิร์ฟเวอร์ที่มีโปรแกรม Secure Shell ในแต่ละแพ็กเก็ตที่ติดต่อกันจะมีชนิดที่แตกต่างกัน ขึ้นอยู่กับข้อมูลที่จะส่ง ซึ่งแพ็กเก็ตแต่ละชนิดจะมีเบอร์เฉพาะ เพื่อให้ทั้งเซิร์ฟเวอร์และไคลเอนต์เข้าใจตรงกัน

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

0	<code>SSH_MSG_NONE</code>	
		This code is reserved. This message type is never sent.
1	<code>SSH_MSG_DISCONNECT</code>	
	<i>string</i>	<i>Cause of disconnection</i>
		This message may be sent by either party at any time. It causes the immediate disconnection of the connection. The message is intended to be displayed to a human, and describes the reason for disconnection.
2	<code>SSH_SMSG_PUBLIC_KEY</code>	
	8 bytes	<i>anti_spoofing_cookie</i>
	32-bit int	<i>server_key_bits</i>
	mp-int	<i>server_key_public_exponent</i>
	mp-int	<i>server_key_public_modulus</i>
	32-bit int	<i>host_key_bits</i>
	mp-int	<i>host_key_public_exponent</i>
	mp-int	<i>host_key_public_modulus</i>
	32-bit int	<i>protocol_flags</i>
	32-bit int	<i>supported_ciphers_mask</i>
	32-bit int	<i>supported_authentications_mask</i>

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

3 *SSH_CMSG_SESSION_KEY*

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

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

4 *SSH_CMSG_USER*

string *user login name on server*

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

5 *SSH_CMSG_AUTH_RHOSTS*

string *client-side user name*

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

6 *SSH_CMSG_AUTH_RSA*

mp-int *identity_public_modulus*

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

7 *SSH_MSG_AUTH_RSA_CHALLENGE*

mp-int encrypted challenge

8 *SSH_CMSG_AUTH_RSA_RESPONSE*

16 bytes MD5 of decrypted challenge

This message is sent by the client in response to an RSA challenge. The MD5 checksum is returned instead of the decrypted challenge to deter known-plaintext attacks against the RSA key.

The server responds to this message with either *SSH_MSG_SUCCESS* or *SSH_MSG_FAILURE*.

9 *SSH_CMSG_AUTH_PASSWORD*

string plain text password

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

10 *SSH_CMSG_REQUEST_PTY*

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

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

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

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

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

n bytes tty modes encoded in binary

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

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

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

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

The server responds to the request with either *SSH_MSG_SUCCESS* or *SSH_MSG_FAILURE*. If the server does not have the concept of pseudo terminals, it should return success if it is

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

11	<i>SSH_CMSG_WINDOW_SIZE</i>
	<i>32-bit int</i> <i>terminal height, rows</i>
	<i>32-bit int</i> <i>terminal width, columns</i>
	<i>32-bit int</i> <i>terminal width, pixels</i>
	<i>32-bit int</i> <i>terminal height, pixels</i>

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

12	<i>SSH_CMSG_EXEC_SHELL</i>
	<i>(no arguments)</i>
	Starts a shell (command interpreter), and enters interactive session mode.

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

14	<i>SSH_SMSG_SUCCESS</i>
	<i>(no arguments)</i>
	This message is sent by the server in response to the session key, a successful authentication request, and a successfully completed preparatory operation.

15	<i>SSH_SMSG_FAILURE</i>
	<i>(no arguments)</i>
	This message is sent by the server in response to a failed Authentication operation to indicate that the user has not yet been successfully authenticated, and in response to a failed preparatory operation. This is also sent in response to an authentication or preparatory operation request that is not recognized or supported.

16	<i>SSH_CMSG_STDIN_DATA</i>
	<i>string</i> <i>data</i>

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

17 *SSH_MSG_STDOUT_DATA*

string *data*

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

18 *SSH_MSG_STDERR_DATA*

string *data*

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

19 *SSH_CMSG_EOF*

(no arguments)

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

20 *SSH_MSG_EXITSTATUS*

32-bit int *exit status of the command*

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

21 *SSH_MSG_CHANNEL_OPEN_CONFIRMATION*

32-bit int *remote_channel*

32-bit int *local_channel*

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

22 *SSH_MSG_CHANNEL_OPEN_FAILURE*

32-bit int *remote_channel*

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

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

23 *SSH_MSG_CHANNEL_DATA*

32-bit int *remote_channel*

string *data*

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

24 *SSH_MSG_CHANNEL_CLOSE*

32-bit int *remote_channel*

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

25 *SSH_MSG_CHANNEL_CLOSE_CONFIRMATION*

32-bit int *remote_channel*

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

26 (OBSOLETE; was unix-domain X11 forwarding)

27 *SSH_SMSG_X11_OPEN*

32-bit int *local_channel*

string *originator_string (see below)*

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

28 *SSH_CMSG_PORT_FORWARD_REQUEST*

32-bit int *server_port*

string *host_to_connect*

32-bit int *port_to_connect*

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

29 *SSH_MSG_PORT_OPEN*

32-bit int *local_channel*

string *host_name*

32-bit int *port*

string *originator_string (see below)*

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

30 *SSH_CMSG_AGENT_REQUEST_FORWARDING*

(no arguments)

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

31 *SSH_SMSG_AGENT_OPEN*

32-bit int *local_channel.*

Sent by the server in interactive session mode, this message requests opening a channel to the

authentication agent. The client should open a channel, and respond with either *SSH_MSG_CHANNEL_OPEN_CONFIRMATION* or *SSH_MSG_CHANNEL_OPEN_FAILURE*.

32 *SSH_MSG_IGNORE*

string *data*

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

33 *SSH_CMSG_EXIT_CONFIRMATION*

(no arguments)

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

34 *SSH_CMSG_X11_REQUEST_FORWARDING*

string *x11_authentication_protocol*

string *x11_authentication_data*

32-bit int *screen number (if SSH_PROTOFLAG_SCREEN_NUMBER)*

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

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

X11 authentication spoofing works by initially sending fake (random) authentication data to the server, and interpreting the first packet sent by the X11 client after the connection has been opened. The first packet contains the client's authentication. If the packet contains the correct fake data, it is replaced by the client by the correct authentication data, and then sent to the X server

35 *SSH_CMSG_AUTH_RHOSTS_RSA*

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

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

36 *SSH_MSG_DEBUG*

string *debugging message sent to the other side*

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

37 *SSH_CMSG_REQUEST_COMPRESSION*

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

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

38 *SSH_CMSG_MAX_PACKET_SIZE*

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

This message can be sent by the client in the preparatory operations phase. The server responds with *SSH_MSG_FAILURE* if it does not support limiting packet size, or with *SSH_MSG_*

SUCCESS if it has limited the maximum packet size (as determined by the value in the size field) to the specified value.

39 *SSH_CMSG_AUTH_TIS*

(no arguments)

This message starts TIS authentication. The server responds with *SSH_MSG_FAILURE* or *SSH_MSG_AUTH_TIS_CHALLENGE*.

40 *SSH_MSG_AUTH_TIS_CHALLENGE*

string *tis challenge*

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

41 *SSH_CMSG_AUTH_TIS_RESPONSE*

string *user response to tis challenge*

When client receives *SSH_MSG_AUTH_TIS_CHALLENGE* and ask users response to challenge it sends it back this message. The server answers with *SSH_MSG_FAILURE* or *SSH_MSG_SUCCESS*.

42 *SSH_CMSG_AUTH_KERBEROS*

string *authentication info*

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

43 *SSH_MSG_AUTH_KERBEROS_RESPONSE*

string *response data*

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

44 *SSH_CMSG_HAVE_KERBEROS_TGT*

string *kerberos credentials*

Client sends kerberos credentials to server and the server replies with *SSH_MSG_SUCCESS* or *SSH_MSG_FAILURE*.

บรรณานุกรม

หนังสืออ้างอิง

- [1] Gary Cornell, Cay S. Horstmann (1999) : “Core JAVA Volumn 1”, Sun Microsystem, Inc. 1999.
- [2] Gary Cornell, Cay S. Horstmann (2000) : “Core JAVA Volumn 2”, Sun Microsystem, Inc. 2000.
- [3] David Flanagan : “JAVA in a Nutshell, Second Edition”, O'Reilly & Associates, Inc., 1997.
- [4] Eliote Rusty Harold : "JavaTM Network Programming, Second Edition", O'Reilly & Associates, Inc., 2000.
- [5] ดร.วีระศักดิ์ ชิงฉาว : “Fundamental of JAVA Programming Volume 1”, Sum Publishing Department, Sum System Company Limited. 1998.
- [6] ดร.วีระศักดิ์ ชิงฉาว : “Fundamental of JAVA Programming Volume 2”, Se-Education Public Company Limited. 2000.
- [7] สุรศักดิ์ สวางพงษ์ “สถาปัตยกรรมและโพรโตคอลที่ซีพี/ไอพี”, Se-Education Public Company Limited. 2000.

เว็บไซต์อ้างอิง

- [1] <http://java.sun.com>
- [2] <http://www.appgate.com>
- [3] <http://www.employees.org/~sacth/ssh/faq/>
- [4] <http://www.ietf.com>
- [5] <http://www.ietf.org/rfc/rfc959.txt>
- [6] <http://www.javaworld.com>
- [7] <http://www.jguru.com>
- [8] <http://www.ssh.com>
- [9] <http://www.w3.org/Protocols/rfc959/>