ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ

Encryption Method Decision Software

นายจีณะ เสนจันทร์ดีไชย
นายชัยฤทธิ์ โกษศรี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต
สาขาเทคโนโลยีอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

หัวข้อปริญญานิพนธ์	ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ	
	Encryption Method Decision Software	
จัดทำโดย	นายจีณะ เสนจันทร์ฒิไชย	42015470
	นายชัยฤทธิ์ โกษศรี	42015471
อาจารย์ที่ปรึกษา	อาจารย์สุธีรา พันธุ์ธีรานุรักษ์	
ภาควิชา	เทคนิคอุตสาหกรรม	

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

..... หัวหน้าภาควิชาเทคนิคอุตสาหกรรม

( )

คณะกรรมการสอบปริญญานิพนธ์

..... ประธานคณะกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

หัวข้อปริญญานิพนธ์	ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ
นักศึกษา	นายจีณะ เสนจันทร์ฒิไชย 42015470
	นายชัยฤทธิ์ โกษศรี 42015471
อาจารย์ที่ปรึกษา	อ.สุธีรา พันธุ์ธีรานุรักษ์
ระดับการศึกษา	อุตสาหกรรมศาสตรบัณฑิต สาขาวิชาเทคโนโลยีอิเล็กทรอนิกส์
ภาควิชา	เทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา	2543

บทคัดย่อ

ปัจจุบันเทคโนโลยีสารสนเทศเป็นที่ยอมรับกันว่ามีความสำคัญต่อชีวิตประจำวันมากขึ้น โดยทุกๆคนสามารถที่จะส่งและรับรู้ข้อมูลข่าวสารได้จากทั่วทุกมุมโลกผ่านทางระบบเครือข่าย อินเตอร์เน็ต จุดนี้เองเป็นสาเหตุให้ข้อมูลข่าวสารที่สำคัญของเราสามารถรั่วไหลได้เนื่องจากการจารกรรมข้อมูลข่าวสาร ซึ่งเป็นความเสี่ยงต่อความปลอดภัยของข้อมูลข่าวสารนั้น การเข้ารหัสลับจึงเป็นวิธีที่สามารถช่วยป้องกันได้ โดยปัจจุบันระบบรหัสลับได้มีการพัฒนาขึ้นอย่างมากมายหลายรูปแบบ แต่ละวิธีก็จะมีคุณสมบัติที่แตกต่างกันไปคนละอย่าง ดังนั้น ปริญญานิพนธ์นี้ จึงได้นำเสนอการเปรียบเทียบคุณสมบัติของการเข้ารหัสแต่ละวิธี เพื่อให้ผู้ใช้สามารถเลือกวิธีการเข้ารหัสที่เหมาะสมกับความต้องการของตนเองได้มากที่สุด

Title	Encryption Method Decision Software
Student	Mr. Jeena Senchantichai 42015470
	Mr. Chairit Kotsri 42015471
Advisor	Ms. Sutheera Puntheeranurak
Level of study	Bachelor of Industrial Technology in Electronic Technology
Department	Industrial Technology Faculty of Engineering
	King Mongkut 's Institute of Technology Ladkrabang
Year	2000

Abstract

Information technology is accepted as an important part of our life at present. Everybody can send or receive the information from everywhere in the world by Internet network system. It will be caused the loss of our important data information due to data trapping, which will be remained the leading risk to data security; hence the encryption has been used for solve this problem. Technologists have developed the cryptosystems in many times, such the algorithm that will be improved. Each algorithm has been had a different property by itself, so this paper will be presented an each algorithm encryption comparison for the users will be decided what kind of encryption that suited for themselves.

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์ที่ปรึกษา อ.สุธีรา พันธุ์ธีรานุรักษ์ เป็นอย่างสูง ที่ได้ให้คำแนะนำและให้คำปรึกษาในการทำโครงงาน ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ นี้ ขอขอบพระคุณ อ.ภุชงค์ หงษ์สุวรรณ ที่ได้ให้คำปรึกษาในเรื่องการเขียนโปรแกรม

ขอขอบพระคุณ คุณพ่อคุณแม่ที่ให้ความช่วยเหลือและให้กำลังใจในการทำโครงงานนี้ตลอดจน ขอบคุณเพื่อนๆทุกคน ที่คอยให้คำปรึกษา ความช่วยเหลือ และให้กำลังใจในการทำโครงงานนี้จนสำเร็จเป็นรูปเล่มปริญญานิพนธ์ได้

จีณะ เสนจันทร์ฒิไชย

ชัยฤทธิ์ โกษศรี

สารบัญ

สารบัญ(ต่อ)

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 แนวความคิดและที่มา

ในยุคปัจจุบันที่เป็นโลกของระบบข่าวสารที่ไร้พรมแดน เพราะมนุษย์สามารถที่จะส่งและรับรู้ข่าวสารได้จากทั่วทุกมุมโลกผ่านทางระบบเครือข่ายอินเตอร์เน็ต การส่งข้อมูลข่าวสารบนระบบอินเตอร์เน็ตเป็นทางเลือกหนึ่งของการสื่อสาร ที่มีต้นทุนเป็นตัวเงินที่ต่ำที่สุด แต่ในทางกลับกันก็เป็นวิธีที่เสี่ยงต่อการถูกโจรกรรมของข้อมูลข่าวสาร หรือมีความปลอดภัยน้อยที่สุด เนื่องจากระบบอินเตอร์เน็ตเป็นระบบเปิดที่ทุกคนสามารถเชื่อมต่อเข้าไปได้ จึงทำให้การรักษาความปลอดภัยของข้อมูลข่าวสารเป็นไปได้ยาก หากว่าข้อมูลข่าวสารยังคงถูกเก็บไว้ในรูปของข้อมูลปกติ ดังนั้น การเข้ารหัสลับข้อมูล จึงเป็นวิธีป้องกันวิธีหนึ่งที่สามารถช่วยป้องกันได้ เพราะถึงแม้ว่าข้อมูลดังกล่าวถูกขโมย แต่ถ้าหากไม่ทราบรหัสผ่านก็จะไม่สามารถทำให้สามารถเปิดดูข้อมูลภายในได้

นับตั้งแต่อดีตเป็นต้นมาจนถึงปัจจุบันนี้ ได้มีนักวิทยาศาสตร์คิดค้นวิธีการเข้ารหัสแบบต่างๆขึ้นมากมาย แล้วทำการเผยแพร่ออกสู่สาธารณะ การที่ได้มีการออกแบบและพัฒนาระบบการเข้ารหัสลับแบบต่างๆขึ้นมาใหม่เรื่อยๆ ทำให้แต่ละวิธีการเข้ารหัสลับนั้นมี คุณสมบัติที่แตกต่างกัน มีข้อดี ข้อด้อยกันไปคนละอย่าง จึงทำให้เป็นที่น่าสนใจว่าการเข้ารหัสลับแบบไหน เหมาะสำหรับการเข้ารหัสลับข้อมูลที่เราจะใช้งาน

1.2 วัตถุประสงค์

เพื่อศึกษาวิธีการเข้ารหัสลับแบบต่างๆ เพื่อการนำไปใช้งานในการรักษาความปลอดภัยของข้อมูลข่าวสาร และเปรียบเทียบหาข้อดีข้อด้อยของการเข้ารหัสลับแต่ละแบบในด้านต่างๆ เพื่อต้องการให้ผู้ใช้สามารถที่จะใช้งานการข้ารหัสลับที่เหมาะสมกับความต้องการของตนเองมากที่สุด

1.3 ขอบเขต

โครงงาน ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกการเข้ารหัส จะมีลักษณะเป็นซอฟต์แวร์โปรแกรม ที่ช่วยในการวิเคราะห์ว่าข้อมูลที่ผู้ใช้ต้องการเข้ารหัสลับ จะเหมาะสมที่จะใช้วิธีการเข้ารหัสลับแบบใด โดยการเปรียบเทียบให้ผู้ใช้ได้เห็นถึงข้อแตกต่างของการเข้ารหัสลับแต่ละวิธีว่ามีประสิทธิภาพเป็นอย่างไร มีข้อดีหรือข้อด้อยต่างกันเช่นไร

ตัวอย่างเช่น ความเร็วในการเข้ารหัสข้อมูลในแต่ละวิธีนั้นจะมีความเร็วที่ไม่เท่ากัน ขึ้นอยู่กับรูปแบบโครงสร้างของการเข้ารหัสข้อมูลแบบนั้น ว่า แต่ละระบบจะมีวิธีการที่ซับซ้อนเพียงใด

หากว่าผู้ใช้ต้องการที่จะใช้สำหรับการเข้ารหัสข้อมูลที่มีขนาดความจุมากๆ ก็จะเห็นความแตกต่างกันได้อย่างมากเลยทีเดียว หรือในด้านความปลอดภัยของข้อมูล จำนวนรหัสลับที่ใช้ในการเข้ารหัสก็มีส่วนสำคัญ ในการที่จะสามารถให้ความเชื่อมั่นได้ส่วนหนึ่งว่า วิธีการเข้ารหัสลับนั้นโอกาสที่จะถูกเจาะรหัสได้ยากง่ายมากน้อยเพียงใด หากว่าผู้ใช้ต้องการความแน่ใจในเรื่องความปลอดภัยมากๆ โปรแกรมก็จะทำการตัดสินใจเลือกวิธีการที่เหมาะสมให้จากการวิเคราะห์หาค่าความน่าจะเป็นว่าจะมีโอกาสในการถูกเจาะรหัสได้มากน้อยเพียงใด ส่วนในด้านการกระจายของข้อมูลโปรแกรมจะทำการวิเคราะห์หาค่าการกระจาย ข้อมูลของการเข้ารหัสในแต่ละแบบ ซึ่งจะบอกได้ว่าวิธีการเข้ารหัสลับข้อมูลในแต่ละแบบนั้น ข้อมูลดิบถูกกระจายรหัสไปมากน้อยเพียงใด ซึ่งจะทำให้ผู้ที่ต้องการถอดรหัสข้อมูลโดยไม่ได้รับอนุญาต ทำได้ยากขึ้นเพราะไม่สามารถเดาได้ว่าข้อมูลเดิมนั้นมีลักษณะเป็นเช่นไร ซึ่งทั้งหมดที่กล่าวมานี้เป็นองค์ประกอบที่จะช่วยให้ผู้ใช้โปรแกรมสามารถเลือกใช้วิธีการเข้ารหัสได้อย่างเหมาะสมตรงกับความต้องการในการใช้งานมากที่สุด

1.4 เนื้อหาของปริญญานิพนธ์

ปริญญานิพนธ์ ฉบับนี้ได้นำเสนอถึง โครงงานระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ ซึ่ง เนื้อหาของปริญญานิพนธ์นี้จะประกอบไปด้วย

บทที่ 2 ได้นำเสนอถึงมาตรฐานการเข้ารหัสลับแบบต่างๆ ที่ได้ถูกคิดค้นมาด้วยนักเทคโนโลยีท่านต่างๆ ซึ่งได้นำมาใช้งานในโครงงานนี้

บทที่ 3 ได้นำเสนอถึงวิธีการที่ใช้ในการที่ใช้ในการช่วยวิเคราะห์ข้อมูล เพื่อแสดงผลเป็นประสิทธิภาพของการเข้ารหัสลับ

บทที่ 4 ได้นำเสนอ อัลกอริธึมของการทำงานแต่ส่วน รวมทั้งการนำเสนอวิธีการออกแบบโปรแกรมส่วนต่างๆ ที่ได้ใช้งานในโครงงานนี้ เช่น ส่วนการเข้ารหัสลับ ส่วนการเปรียบเทียบวิธีการเข้ารหัส

บทที่ 5 เป็นการนำเสนอผลการทำงานของโปรแกรมส่วนต่างๆของโครงงานนี้ เช่น ส่วนการเข้ารหัสลับ ส่วนการเปรียบเทียบวิธีการเข้ารหัส

บทที่ 6 ได้นำเสนอถึงบทสรุป โครงงาน แนวทางการแก้ไขและพัฒนาโครงงานต่อไป

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ด้านวิชาการ

1 มีความรู้ความเข้าใจวิธีการเข้ารหัสลับแบบต่างๆ

2 มีความรู้ความเข้าใจถึงข้อแตกต่างของวิธีการเข้ารหัสแต่ละแบบ

3 มีความรู้ความเข้าใจในการเขียนโปรแกรมแนว visual

4 สามารถวิเคราะห์ได้ว่าการเข้ารหัสแต่ละแบบมีความเหมือนหรือแตกต่างกันอย่างไร และสามารถเขียนโปรแกรมวิเคราะห์ได้

ด้านการใช้งานทั่วไป

1 สามารถนำโปรแกรมไปใช้งานในการเข้ารหัสรูปแบบต่างๆได้

2 สามารถใช้โปรแกรมเพื่อช่วยการตัดสินใจเลือกวิธีการเข้ารหัสที่ดีที่สุดได้

บทที่ 2
มาตรฐานการเข้ารหัสข้อมูล (Data Encryption)

ไม่ว่าคอมพิวเตอร์จะมีระบบความปลอดภัยอยู่ในระดับใดก็ตาม หากว่าข้อมูลที่เก็บอยู่ในเครื่องยังคงอยู่ในรูปข้อมูลปกติ หรือตัวอักษรธรรมดา (Plain Text) ความปลอดภัยของข้อมูลก็ยังต่ำมาก เนื่องจากเมื่อผู้อื่นรู้ชื่อผู้ใช้และรหัสผ่านที่ถูกต้องแล้ว ก็สามารถลักลอบนำข้อมูลที่เก็บไว้ไปใช้ได้ทันที ซึ่งเท่ากับว่าความปลอดภัยของข้อมูลนี้ขึ้นอยู่กับรหัสผ่านเพียงอย่างเดียว ดังนั้นการเข้ารหัสข้อมูล หรือ Data Encryption จึงเป็นอีกวิธีหนึ่งที่จะป้องกันไม่ให้ข้อมูลอันมีค่าของเราถูกลักลอบนำไปใช้

การเข้ารหัสข้อมูล คือ การนำข้อมูลปกติที่อยู่ในรูปของตัวอักษรธรรมดา มาผ่านการเข้ารหัสทำให้ข้อมูลออกมาในอีกรูปแบบหนึ่ง ซึ่งจะไม่มีความหมาย อ่านไม่ออกและแปลกลับไปหาข้อมูลเดิมไม่ได้หากไม่มีรหัสที่ถูกต้อง ดังนั้นถึงแม้ใครจะแอบสำเนาข้อมูลที่เข้ารหัสลับนี้ไปได้ แต่ถ้าไม่รู้รหัสลับก็จะไม่สามารถใช้ประโยชน์จากข้อมูลนั้นได้ ดังนั้นการเข้ารหัสข้อมูลจึงเป็นมาตรการสำหรับการรักษาความปลอดภัยของข้อมูลที่เพิ่มขึ้นจากระบบความปลอดภัยของคอมพิวเตอร์ ซึ่งการเข้ารหัสข้อมูลจะแบ่งออกเป็น 2 ประเภท คือ Symmetric Encryption และ Asymmetric Encryption

2.1 Symmetric Encryption

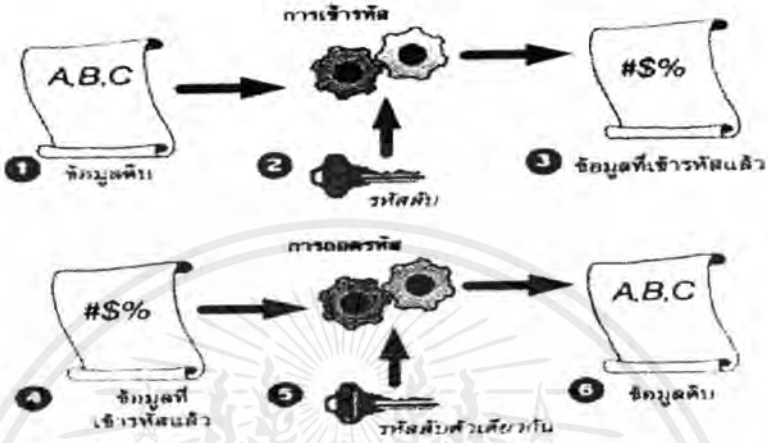
เป็นการเข้ารหัสข้อมูลพื้นฐาน โดยนำข้อมูลตัวอักษรและรหัสลับที่กำหนดขึ้นมาเข้ารหัสทางคณิตศาสตร์ เพื่อให้ได้ผลลัพธ์ออกมาเป็นข้อมูลที่เข้ารหัสแล้ว ซึ่งสามารถทำให้อยู่ในรูปของฟังก์ชันทางคณิตศาสตร์ได้ดังนี้

ข้อมูลที่เข้ารหัสแล้ว = ฟังก์ชันการเข้ารหัส(ข้อมูลดิบ,รหัสลับ)

และในการถอดรหัสข้อมูลกลับคืนมาให้อยู่ในรูปเดิม ก็ต้องใช้รหัสตัวเดิมมาทำเป็นขบวนการย้อนกลับกับการเข้ารหัส โดยมีฟังก์ชันทางคณิตศาสตร์เป็นดังนี้

ข้อมูลดิบ = ฟังก์ชันการถอดรหัส(ข้อมูลที่เข้ารหัสแล้ว,รหัสลับ)

ถึงแม้ว่าฟังก์ชันการเข้ารหัสและถอดรหัสจะมีขบวนการที่ชัดเจน และทราบดีว่ามีขั้นตอนรายละเอียดอย่างไร ก็อาจจะถอดรหัสข้อมูลไม่ได้ ถ้าไม่รู้รหัสลับที่ใช้ในการเข้ารหัสข้อมูลนั้นๆ ดังแสดงในรูป



รูปที่ 2.1 แสดงการเข้ารหัสและถอดรหัสแบบ Symmetric Encryption

หรือ การเข้ารหัสลับแบบคีย์เดี่ยว

1) เริ่มต้นจากข้อมูลดิบที่ต้องการเข้ารหัส
2) ทำการเข้ารหัสข้อมูลโดยใช้รหัสลับ
3) ได้ข้อมูลที่เข้ารหัสออกมาส่งให้ผู้รับ
4) ผู้รับนำข้อมูลที่ได้มาถอดรหัส
5) ข้อมูลจะถอดรหัสโดยใช้รหัสลับตัวเดียวกัน
6) ผู้รับจะได้ข้อมูลดิบกลับมาตามเดิม

ข้อดีของ Symmetric Encryption ก็คือ สามารถทำการเข้ารหัสลับที่ใช้สำหรับเข้ารหัสและถอดรหัสข้อมูลได้อย่างรวดเร็ว แต่มีข้อเสีย คือ การรักษารหัสลับที่ใช้สำหรับเข้าและถอดรหัสนั้นทำได้ยาก โดยเฉพาะในระบบใหญ่ๆที่มีผู้ใช้เป็นจำนวนมาก หากมีใครรู้รหัสลับที่ใช้ ก็สามารถถอดรหัสมาอ่านข้อความและแก้ไขข้อความนั้นได้โดยไม่มีใครรู้ นอกจากนี้การที่ผู้ส่งและผู้รับใช้รหัสลับในการเข้ารหัสและถอดรหัสตัวเดียวกัน และการที่ผู้ส่งแจ้งรหัสลับไปให้ผู้รับทราบโดยไม่ให้คนอื่นรู้นั้นเป็นสิ่งที่ลำบากที่สุดของการเข้ารหัสข้อมูลในแบบ Symmetric Encryption ทีเดียว โดยทั่วไปก็จะส่งรหัสลับกันเป็นเอกสารแท็กซี่ หรือการใช้วิธีโทรศัพท์แจ้งรหัสลับให้ผู้รับทราบ ซึ่งจะเห็นได้ว่าในแต่ละวิธีนั้นก็มีความเสี่ยงในการรั่วไหลของรหัสลับได้ทั้งนั้น

การเข้ารหัสข้อมูลแบบ Symmetric Encryption นี้มีชื่อเรียกอีกชื่อหนึ่งว่า "Secret Key Encryption" คือ รหัสลับที่ใช้ในการเข้ารหัสและถอดรหัสนั้นเป็นรหัสตัวเดียวกัน โดยผู้ส่งและผู้รับจะต้องทราบรหัสลับนี้ร่วมกัน จึงสามารถรับส่งข้อมูลที่เข้ารหัสลับแล้วนี้ได้อย่างถูกต้อง ซึ่งการเข้ารหัสข้อมูลแบบนี้ สามารถทำได้ทีละกลุ่ม เรียกว่า Block Cipher คือจะทำการอ่านข้อมูลดิบมาทีละกลุ่ม โดยมีความยาวของข้อมูลคงที่แล้วทำการเข้ารหัสข้อมูลของกลุ่มนั้น เมื่อได้ผลลัพธ์ออกมาจะเป็นข้อมูลที่เข้ารหัสแล้วจะมีความยาวคงที่เช่นเดียวกัน ส่วนอีกแบบหนึ่งคือ การเข้ารหัสแบบต่อเนื่อง หรือ Stream Cipher จะอ่านข้อมูลดิบเข้ามาแล้วทำการเข้ารหัสทีละบิตต่อเนื่องกันไป เมื่อได้ผลลัพธ์ออกมาก็จะเป็นข้อมูลที่เข้ารหัสแล้ว ทีละส่วนต่อเนื่องกันไป โดยจะทำการเข้รหัสเช่นนั้นต่อไปเรื่อยๆ จนกระทั่งจบข้อมูล ซึ่งการเข้ารหัสข้อมูลแบบ Symmetric Encryption ที่นิยมใช้กันมากนั้นมีอยู่หลายชนิดเช่น

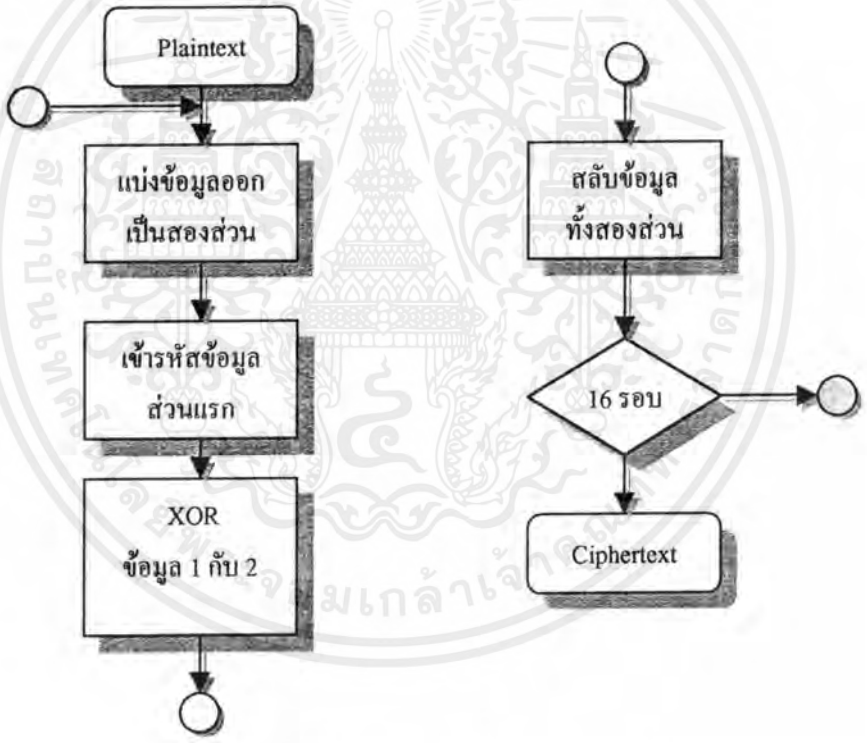
2.1.1 การเข้ารหัสลับแบบ Data Encryption Standard (DES)

DES เป็นมาตรฐานการเข้ารหัสข้อมูลชนิดหนึ่งของ Symmetric Encryption ที่คิดค้นโดยบริษัทไอบีเอ็ม เมื่อ ค.ศ. 1977 ต่อมาได้มีการปรับปรุงโดยกระทรวงกลาโหมของสหรัฐฯ และกำหนดเป็นมาตรฐาน ANSI X3.92 และ X3.106 โดย DES ถูกออกแบบมาให้ทำงานโดยใช้ฮาร์ดแวร์เป็นตัวเข้ารหัสและถอดรหัส หรือจะใช้ซอฟต์แวร์ทำงานในลักษณะเดียวกันก็ได้ การทำงานของ DES จะเป็นแบบ Block Cipher คือการเข้ารหัสข้อมูลทีละกลุ่ม กลุ่มละ 64 บิต โดยใช้รหัสลับขนาด 56 บิต ซึ่งเป็นการเข้ารหัสข้อมูล DES นี้ ได้ผ่านการทดสอบจนเป็นที่ยอมรับว่า เป็นการเข้ารหัสข้อมูลที่ปลอดภัยมากที่สุดชนิดหนึ่ง

การเข้ารหัสของ DES สามารถเข้ารหัสข้อมูลได้ 2 วิธี คือ Electronic Code Block (ECB) ซึ่งจะเป็นการเข้ารหัสข้อมูลทีละครั้ง ครั้งละ 64 บิต โดยใช้รหัสลับขนาด 56 บิต ดังนั้นข้อมูล 64 บิตที่นำมาเข้ารหัส จะไม่เกี่ยวข้องกับข้อมูลส่วนอื่นๆที่เหลือเลย ส่วนอีกวิธีหนึ่งคือ Cipher Block Chaining (CBC) จะเป็นการเข้ารหัสข้อมูลครั้งละ 64 บิตเช่นเดียวกัน และใช้รหัสลับขนาด 56 บิต ต่างกันตรงที่ว่าข้อมูลที่นำมาเข้ารหัสจะถูก Exclusive OR (XOR) กับข้อมูล 64 บิตก่อนหน้านั้นเสียก่อนแล้วจึงนำมาเข้ารหัส ทำให้ข้อมูล 64 บิตที่มีข้อความเหมือนกัน ถูกเข้ารหัสข้อมูลแล้วได้ผลลัพธ์ออกมาไม่เหมือนกัน เป็นผลให้การเดาถอดรหัสข้อมูลยากยิ่งขึ้น

จากจำนวนรหัสลับขนาด 56 บิตนี้จะมีจำนวนรหัสที่เป็นไปได้ทั้งหมดประมาณ 72,000 ล้านล้านรหัส ซึ่งในการเข้ารหัสนั้นเราจะสุ่มรหัสใดรหัสหนึ่งมาเป็นรหัสลับสำหรับการเข้ารหัสของ DES ทำให้การเดาทำได้ยากมาก และนำรหัสลับที่เลือกไว้มาประมวลผลการเข้ารหัสข้อมูลกลับไปกลับมาถึง 16 ครั้ง จึงถือว่าการเข้ารหัสของ DES เป็นการเข้ารหัสที่มีความปลอดภัยสูง ยากแก่การเดาถอดรหัส หากมีผู้ต้องการถอดรหัสต้องเดารหัสหลายหมื่นล้านรหัสกว่าจะพบรหัสที่ถูกต้อง

DES มีการทำงานที่รวดเร็วและสามารถทำในฮาร์ดแวร์ได้ แต่การจำหน่ายซอฟต์แวร์หรือฮาร์ดแวร์ที่ใช้ในการเข้ารหัสข้อมูลตามมาตรฐาน DES นอกสหรัฐอเมริกาเป็นสิ่งที่ผิดกฎหมาย เนื่องจากรัฐบาลสหรัฐฯ ถือว่าเป็นผลิตภัณฑ์ควบคุม ห้ามนำออกจำหน่ายนอกประเทศ ดังนั้นสำหรับงานเข้ารหัสที่ต้องการความปลอดภัยสูง จะใช้การเข้ารหัสข้อมูลที่เรียกว่า Triple-DES โดยมีหลักการทำงานคือ ใช้ DES และรหัสลับลำดับที่ 1 ทำการเข้ารหัสข้อมูลตามปกติ แล้วนำผลลัพธ์ที่ได้มาเข้ารหัสตามมาตรฐาน DES ครั้งที่ 2 โดยใช้รหัสลับลำดับที่ 2 ในการเข้ารหัส จากนั้นจึงนำผลลัพธ์ที่ได้มาเข้ารหัส DES อีกครั้งเป็นครั้งที่ 3 โดยใช้รหัสลับลำดับที่ 3 ในการเข้ารหัส ซึ่งรหัสลับทั้ง 3 ตัวนี้จะไม่เกี่ยวข้องกันเลย ทำให้ข้อมูลมีความปลอดภัยสูงขึ้นกว่าการเข้ารหัสตามปกติ ส่วนทางผู้รับก็จะถอดรหัสข้อมูลนี้ทีละขั้น ย้อนกลับไปจนได้ข้อมูลเดิมกลับมา



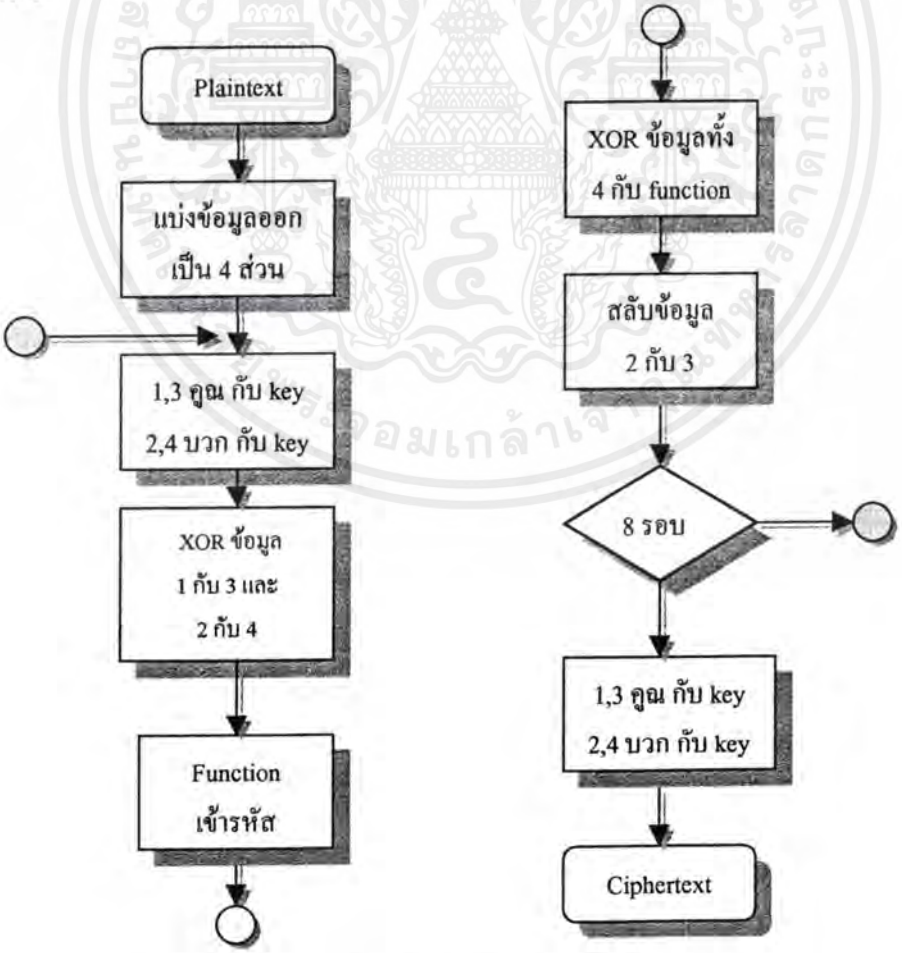
รูปที่ 2.2 แสดง Flow Chart การทำงานของ DES

2.1.2 การเข้ารหัสลับแบบ International data Encryption Algorithm (IDEA)

International data Encryption Algorithm หรือ IDEA เป็นการเข้ารหัสข้อมูลประเภท Symmetric Encryption แบบ Block Cipher อีกชนิดหนึ่ง ซึ่งผู้ส่งและผู้รับจะใช้รหัสตัวเดียวกันในการเข้ารหัสและถอดรหัสข้อมูล IDEA เป็นมาตรฐานของทางยุโรป เริ่มใช้ในราวปี ค.ศ. 1990 เป็นต้นมา โดย IDEA จะใช้รหัสลับขนาด 128 บิตในการเข้ารหัสข้อมูล และเป็นมาตรฐานที่ผ่านการทดสอบจนเป็นที่ยอมรับในด้านความปลอดภัยเช่นเดียวกับ DES ทั้งทางด้านความเร็วในการทำงาน และความ

ปลอดภัยของข้อมูลที่เข้ารหัสแล้ว เนื่องจาก IDEA ถูกคิดค้นขึ้นโดย Xuejai Lai และ James Massy ที่ Swiss Federal Institute of Technology เมืองซูริค ประเทศสวิสเซอร์แลนด์ จึงไม่เกี่ยวข้องกับการควบคุมของสหรัฐฯ ทำให้ ถูกใช้ในการเข้ารหัสข้อมูลแพร่หลายกว่า DES เพราะไม่ถูกควบคุมการส่งออก ที่เข้มงวดเหมือนกับ DES นอกจากนี้รหัสลับขนาด 128 บิต ที่ใช้ใน IDEA ก็มีความปลอดภัยของข้อมูลสูงกว่า DES ที่ใช้รหัสลับเพียง 56 บิตมาก

ข้อดีของ IDEA คือ มีความเร็วในการทำงานมากกว่า DES ซึ่งเมื่อใช้ซอฟต์แวร์ในการเข้ารหัสและถอดรหัส DES ข้อมูล จะทำงานได้ช้ากว่า IDEA ดั้งนั้นแม้ว่า IDEA จะใช้จำนวนบิตสำหรับเข้ารหัสข้อมูลถึง 128 บิตก็ตาม แต่ไม่มีผลทำให้การเข้ารหัสช้าลงเลย และยังมีความปลอดภัยมากกว่าด้วย และเมื่อเทียบกับการเข้ารหัสตามมาตรฐานอื่นๆเช่น RSA แล้ว การเข้ารหัสของ IDEA มีความเร็วในการทำงานที่สูงกว่า ถึง 4,000 เท่า ในจำนวนบิตของรหัสลับที่เท่ากัน นอกจากนี้การเข้ารหัสที่ซับซ้อนของ IDEA ทำให้รหัสลับขนาด 128 บิต ของ IDEA มีผลความปลอดภัยเทียบเท่ากับการเข้ารหัส RSA ที่ใช้รหัสถึง 3,100 บิต ทำให้ IDEA เป็นการเข้ารหัสข้อมูลที่ได้รับการยอมรับอย่างแพร่หลายอีกประเภทหนึ่ง



รูปที่ 2.3 แสดง Flow Chart การเข้ารหัสแบบ IDEA

2.1.3 การเข้ารหัสลับแบบ RC2 และ RC4

RC2 และ RC4 เป็นมาตรฐานการเข้ารหัสข้อมูลประเภท Symmetric Encryption ที่คิดค้นโดย Ronald L. Rivest หนึ่งในผู้ก่อตั้งบริษัท RSA Data Security เพื่อใช้เป็นทางเลี่ยงของ DES ในการส่งออกไปจำหน่ายนอกสหรัฐอเมริกา โดยคำว่า RC ย่อมาจาก "Rivest's Cipher" โดย RC2 ได้ถูกออกแบบมาให้ใช้แทน DES เพื่อส่งออกไปจำหน่ายนอกสหรัฐอเมริกา ซึ่ง RC2 จะทำงานได้เร็วกว่า 2-3 เท่า เมื่อใช้ซอฟต์แวร์เป็นตัวเข้ารหัส

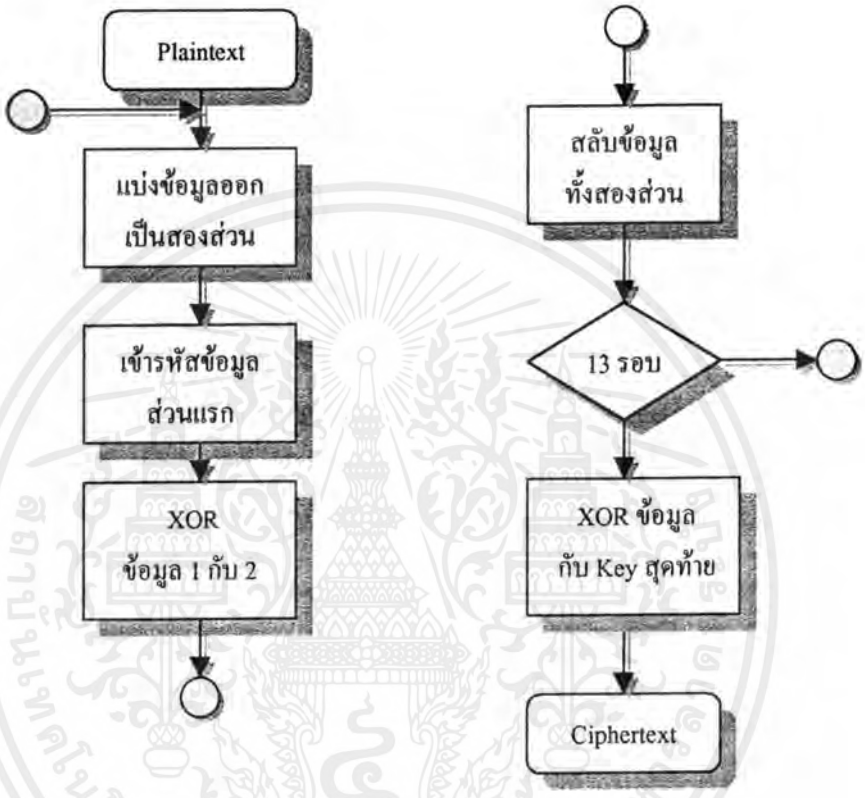
RC2 และ RC4 จะทำการเข้ารหัสคล้ายกับ DES แต่รหัสลับที่ใช้จะมีขนาดไม่แน่นอน สามารถกำหนดขนาดความยาวของรหัสลับในการเข้ารหัสข้อมูลได้สูงสุดไม่เกิน 40 บิต ซึ่ง RC2 จะเป็นการเข้ารหัสข้อมูลแบบ Block Cipher เหมือนกับ ECB ของ DES คืออ่านข้อมูลครั้งละ 64 บิต และใช้รหัสมีความยาวไม่เกิน 40 บิต สำหรับรหัสลับที่มีความยาว 56 บิต จะอนุญาตให้ใช้เฉพาะบริษัทในเครือข่ายของบริษัทแม่หรือสำนักงานสาขาของสหรัฐอเมริกาเท่านั้น ส่วน RC4 จะเป็นการเข้ารหัสข้อมูลแบบ Streamer Cipher คือ อ่านข้อมูลมาทำการเข้ารหัสอย่างต่อเนื่อง และมีรหัสลับความยาวไม่เกิน 40 บิต เช่นเดียวกัน ซึ่ง RC4 จะใช้ในการเข้ารหัสที่เรียกว่า "Random Permutation" ทั้ง RC2 และ RC4 มีการทำงานที่รวดเร็วเช่นเดียวกัน เมื่อใช้ซอฟต์แวร์ในการเข้ารหัสและถอดรหัส การเข้ารหัสชนิด RC4 สามารถใช้งานได้ทั้งการเข้ารหัสไฟล์และการเข้ารหัสเชื่อมต่อรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ ซึ่งทั้ง RC2 และ RC4 สามารถนำออกจำหน่ายนอกสหรัฐอเมริกาได้โดยไม่ถูกควบคุม ส่วนซอฟต์แวร์ที่ใช้ในการเข้ารหัส DES นั้นแทบจะไม่เคยได้รับอนุญาตให้จำหน่ายนอกสหรัฐอเมริกาเลย

2.1.4 การเข้ารหัสลับแบบ Blowfish

Blowfish เป็นการเข้ารหัสข้อมูลชนิด Symmetric Encryption แบบ Block Cipher ที่สามารถใช้แทนการเข้ารหัสแบบDES หรือ IDEA ได้ ซึ่ง Blowfish เป็นการเข้ารหัสที่จะทำการเข้ารหัสข้อมูลครั้งละ 64 บิต ด้วยรหัสลับที่มีขนาดไม่แน่นนอน ซึ่งอาจสูงถึง 448 บิตสามารถกระจายออกไปเป็นรหัสลับย่อยได้ถึง 4168 ไบต์ Blowfish ถูกคิดค้นโดย Bruce Schneier เมื่อปี คศ. 1993เพื่อเป็นทางเลือกในวิธีการการเข้ารหัสที่รวดเร็วและอิสระมากกว่าการเข้ารหัสแบบอื่น

การเข้ารหัสแบบBlowfish เป็นการเข้ารหัสที่ผู้คิดค้นไม่ได้จดลิขสิทธิ์ จึงอนุญาตให้ใช้ได้อย่างอิสระ ด้วยการเข้ารหัสข้อมูลจะทำการเข้ารหัสข้อมูลถึง 16 รอบนี้ และจะกระทำเป็นบล็อกข้อมูลครั้งละ 32 บิต เช่นเดียวกับ DES ซึ่ง Blowfish จะใช้ ค่าของ พาย π จำนวน 16 หลักในการจัดเรียง 4 S-box และ อะเรย์ p อีก 1 ชุด

ข้อดีของการเข้ารหัสแบบ Blowfish คือการที่ Blowfish เข้ารหัสข้อมูลถึง 16 ครั้ง ทำให้มีความปลอดภัยมากต่อการที่จะเดาถอดรหัส และสามารถป้องกันการถอดรหัสแบบอาศัยความแตกต่างได้เป็นอย่างดี เพราะถ้าระบบที่ไม่ปลอดภัยนัก Blowfish จะทำการรวม algorithm ของมันลงไปกับระบบนั้นๆด้วย



รูปที่ 2.4 แสดง Flow Chart การเข้ารหัสแบบ Blowfish

2.1.5 การเข้ารหัสแบบ CAST128

CAST128 หรือที่รู้จักว่า CAST5 หรือ CAST 5 ถูกพัฒนาขึ้นโดย Carlisle Adame ที่ Entrust Technology เป็นการเข้ารหัสแบบ Symmetric ชนิด Block Cipher ด้วยการเข้ารหัสทีละกลุ่ม กลุ่มละ 64 บิต โดยใช้รหัสลับขนาด 128 บิต ซึ่งรหัสลับนี้สามารถใช้รหัสลับอย่างสั้น ตั้งแต่ 40 ถึง 128 บิต (โดยเพิ่มขึ้นทีละ 8 บิต) สำหรับรหัสลับนี้ถ้าเพิ่มขึ้นถึง 80 บิต จะใช้การเข้ารหัส 12 รอบ และ 16 รอบ สำหรับรหัสอย่างยาว และ สำหรับการเข้ารหัสลับแบบ CAST128 นี้ เราสามารถเลือก mode การทำงานได้เป็น ECB, CBC, PCBC, OFB และ CFB รวมทั้ง mode การทำงานที่มีการเพิ่มเติมขึ้นอีกหลายๆ mode

การทำงานของ CAST128 ใน mode ECB (Electronic Code Book) จะทำงานอย่างอิสระ โดยใช้ codebook และ block replay ซึ่ง codebook จะทำการเข้ารหัสลับของข่าวสารให้ตรงกับบล็อกข่าว

สาร ciphertext เพื่อที่จะได้สามารถทำการสร้างข่าวสารนั้นกลับคืนมาเมื่อทำการถอดรหัสลับคืนได้ โดยปราศจากการที่จะมีผู้ล่วงรู้รหัสลับนั้นๆ ส่วนการเข้ารหัสใน mode CBC จะสามารถที่จะทำการเข้ารหัสข้อมูลดิบโดยจะเข้ารหัสเป็นบล็อกของข่าวสารตามข้อมูลดิบเดิม

2.1.6 การเข้ารหัสลับแบบ SHARK

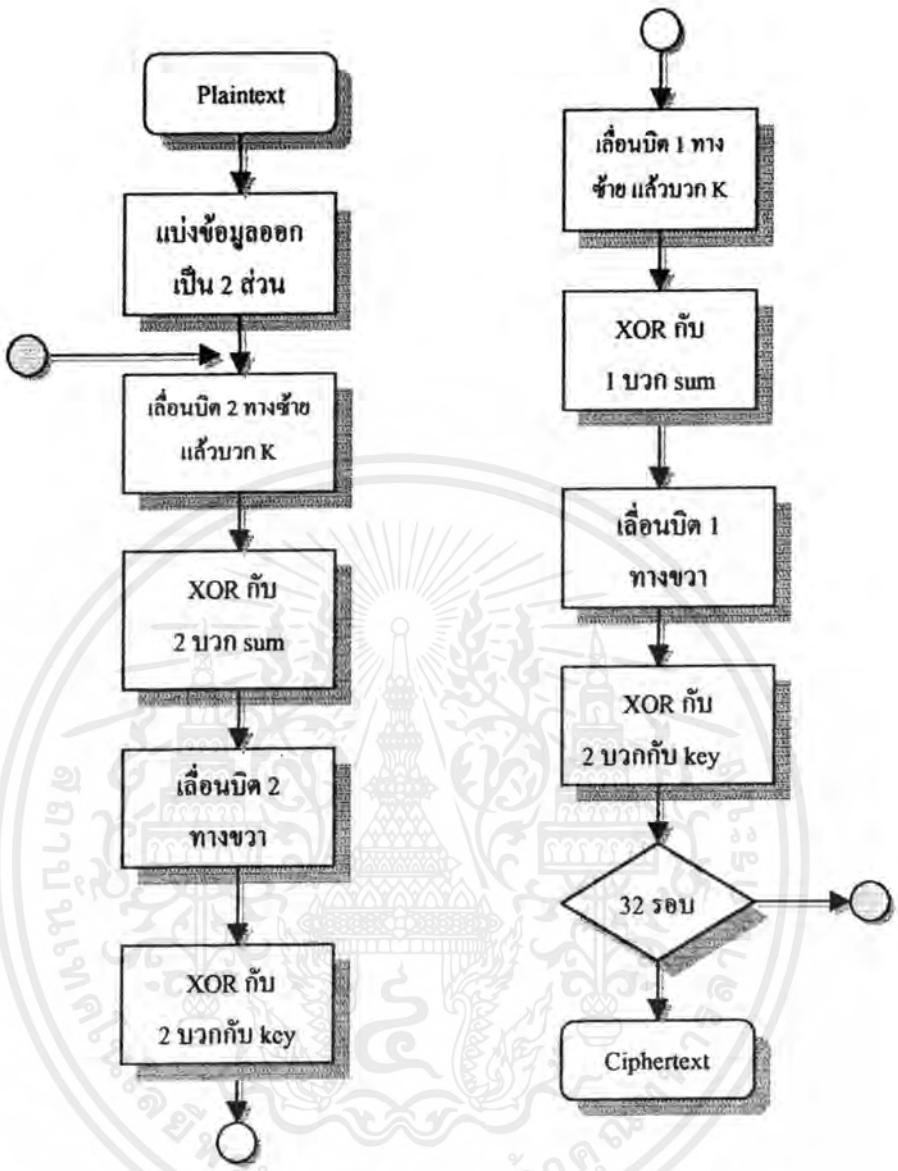
SHARK เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher ที่ถูกออกแบบโดย Joan Daeman, Vincent Rijmen, Bart Preenel, Anton Bosselaers และ Erik de Win โดย SHARKจะเป็นการเข้ารหัสลับที่นำเอา box ย่อยที่ไม่เป็นเชิงเส้น กับ ความแตกต่างของรหัสที่แก้รหัสที่ผิดพลาด (MDS-Code) ซึ่งจะทำให้เกิดการกระจายของข้อมูลได้เป็นอย่างดี

การเข้ารหัสแบบ SHARK นี้ จะสามารถต่อต้าน การถอดรหัสแบบอาศัยความแตกต่าง และการถอดรหัสแบบเชิงเส้นได้ด้วยการเข้ารหัสเพียงน้อยรอบเท่านั้น โดยโครงสร้างของ SHARK จะใช้ซอฟต์แวร์ที่มีความรวดเร็วสูงในการเข้าและถอดรหัสข้อมูล ซึ่งจะทำงานได้มากกว่าถึง 4 ครั้ง เมื่อทำงานบนเครื่องคอมพิวเตอร์ขนาด 64 บิต ทำให้มีความเร็วมากว่า SAFER และ IDEA

2.1.7 การเข้ารหัสแบบ TEA (Tiny Encryption Algorithm)

TEA เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher เป็นหนึ่งในอัลกอริทึมของการเข้ารหัสลับที่เร็วที่สุดและมีประสิทธิภาพมาก ถูกพัฒนาขึ้นโดย David Wheeler และ Roger Needham ที่ห้องปฏิบัติการคอมพิวเตอร์ มหาวิทยาลัยเคมบริดจ์ เป็นการออกแบบโปรแกรมขนาดสั้นๆที่สามารถทำงานได้ในฮาร์ดแวร์เกือบทุกๆชนิด ที่ต้องการใช้รหัสพิเศษหรือจำเป็นต้องให้ผู้ใช้สามารถจดจำรูปแบบโครงสร้างของอัลกอริทึมและรหัสลับของมันได้ เมื่อต้องการให้เครื่องคอมพิวเตอร์ตัดสินใจเลือกในภายหลัง ซึ่งTEA นี้ เป็นการเข้ารหัสลับข้อมูลขนาด 64 บิต โดยใช้รหัสลับขนาด 128 บิต ที่มีรอบการทำงานถึง 32 รอบ

โดยฟังก์ชันของ TEA จะใช้การเลื่อนบิต และการบวกแบบมีตัวทด และมี golden ratio คือ $(\sqrt{5}-1)\times 2^{31}$ ที่จะใช้ในการบวกค่าของทุกๆรอบเพื่อป้องกันการเลือกโจมตีข้อมูลดิบได้ ซึ่งเป็นการป้องกันการเจาะรหัสแบบอาศัยความแตกต่างและสามารถที่จะกระจายข้อมูลได้อย่างสมบูรณ์แบบ (โดยที่ข้อมูลดิบ 1 บิต จะถูกกระจายข้อมูลเป็นข้อมูลมูลที่เข้ารหัสลับที่แตกต่างกันได้ถึง 32 บิต) จึงทำให้เป็นที่มั่นใจได้ว่าเป็นการเข้ารหัสลับที่ปลอดภัยมาก



รูปที่ 2.5 แสดง Flow Chart การเข้ารหัสแบบ TEA

2.1.8 การเข้ารหัสลับแบบ GOST

GOST เป็นมาตรฐานการเข้ารหัสที่รัฐบาลอดีตสหภาพโซเวียตใช้เป็นมาตรฐานของระบบความปลอดภัยของข้อมูล ด้วยมาตรฐานเลขที่ 28147-89 เช่นเดียวกับสหรัฐอเมริกาที่มีมาตรฐานการเข้ารหัสแบบ DES เป็นระบบความปลอดภัย

GOST เป็นการเข้ารหัสลับข้อมูลแบบ Symmetric ชนิด block cipher ที่เป็น Cipher Block Chaining (CBC) ที่ทำการเข้ารหัส ข้อมูลเป็นกลุ่ม กลุ่มละ 64 บิต ด้วยการใช้รหัสลับขนาด 256 บิต โดยที่สามมารถเปลี่ยนค่าใน S-box และเก็บไว้เป็นความลับได้ ทำให้ GOST นี้สามารถที่จะป้องกันการถอดรหัสแบบอาศัยความแตกต่างได้

การเข้ารหัสแบบ GOST นี้มีข้อเหมือนและข้อแตกต่างจากการเข้ารหัสแบบ DES ดังจะเห็นได้ดังต่อไปนี้

- GOST ของสหภาพโซเวียต จะทำการเข้ารหัสลับถึง 32 รอบ มากกว่า DES ที่ทำการเข้ารหัสเพียง 16 รอบเท่านั้น
- การเข้ารหัสในแต่ละรอบของ GOST จะเหมือนกับการเข้ารหัสแบบ DES โดยใช้ฟังก์ชัน F
- GOST จะใช้รหัสลับขนาด 256 บิต เมื่อใช้รหัสลับตัวแรก และ 512 บิต สำหรับรหัสลับตัวที่สอง ซึ่ง DES จะใช้เพียง 32 บิตสำหรับรหัสลับตัวแรก

2.1.9 การเข้ารหัสลับแบบ Square

Square เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher ที่ถูกออกแบบโดย Joan Daeman และ Vincent Rijmen ซึ่งเป็นการเข้ารหัสที่มีความรวดเร็วโดยจะแบ่งข้อมูลออกเป็นบล็อกๆ ละ 128 บิต และใช้รหัสลับขนาด 128 บิต เช่นเดียวกัน โดยมีแนวคิดการออกแบบมาเพื่อที่จะใช้กับเครื่องคอมพิวเตอร์รุ่นใหม่ที่สามารถประมวลผลได้อย่างรวดเร็ว การเข้ารหัสจะใช้รหัสลับที่ได้ค่าจากเมตริกซ์ GF (2^8) ซึ่งจะทำให้การประมวลเป็นไปได้อย่างรวดเร็วยิ่งขึ้น และมีความสามารถในการต่อต้านการถอดรหัสโดยอาศัยความต่าง (differential cryptanlysis)

โครงสร้างของ Square จะเป็นการเข้ารหัสลับที่ทำงานแบบซ้ำ ไม่ใช่การทำงานแบบ Feistel แทนที่ทุกๆรอบการทำงานจะกระทำกับข้อมูลทั้งหมด 16 ไบต์ทีเดียว แต่หนึ่งรอบการทำงานของ Square นั้นจะประกอบไปด้วยการทำงานที่แตกต่างกัน 4 อย่าง รวมอยู่ในตาราง lookup และการเอ็กซ์คลูซีฟ ซึ่งทั้ง 4 กระบวนการทำงานคือ

การแปลงแบบเชิงเส้น จะทำงานโดยแบ่งข้อมูลขนาด 16 ไบต์ออกเป็นอะเรย์ขนาด 4×4แล้วหาสัมประสิทธิ์ของทั้ง 4 ส่วน

การแปลงแบบไม่เชิงเส้น γ โดยข้อมูลขนาด 16 ไบต์จะถูกสลับที่โดยอ้างอิงจาก S-box 8 บิต

การสลับที่ของข้อมูลด้วยค่า π โดยจะทำการเรียงข้อมูลใหม่ในแถวและคอลัมน์นั้นๆ

การบวกรหัสลับเพิ่ม โดยทำการเอ็กซ์คลูซีฟด้วยรหัสลับที่ใช้ในรอบนั้นๆ

ทำให้รอบกระบวนการทำงานที่สามารถเขียนได้เป็น $p[k^i]$ โดยที่

$$p[k^i] = \sigma[k^i] \pi \gamma \theta$$

โดย Square สามารถที่จะทำงานได้ที่ความเร็ว 15 MByte/s บนเครื่อง Pentium II 333MHz หรือ ไมโครคอนโทรลเลอร์ M68HC05 ที่ขนาด ROM 547 ไบต์ ด้วยความเร็ว 7500 รอบ ที่ความเร็วสัญญาณนาฬิกา 4 MHz

มาตรฐานการเข้ารหัสแบบใหม่ (Advance Encryption Standard, AES)

มาตรฐานการเข้ารหัสแบบใหม่ หรือ AES เป็นการมาตรฐานของการประมวลผลข่าวสาร (Federal Information Processing Standard, FIPS) ที่ใช้สำหรับการเข้ารหัสลับข้อมูล ซึ่งถูกกำหนดขึ้นโดย สถาบันมาตรฐานและเทคโนโลยีแห่งชาติของสหรัฐ(National Institute of Standard and Technology, NIST) เพื่อใช้เป็นมาตรฐานการเข้ารหัสลับข้อมูลใหม่ ทดแทนการเข้ารหัสลับแบบ DES หรือ Triple DES ซึ่งค่อนข้างจะล้าสมัย แต่ในเวลานี้ (ปี ค.ศ. 2000) ก็ยังไม่ได้มีการประกาศรับรองมาตรฐานการเข้ารหัสแบบ AES เป็นมาตรฐานการเข้ารหัสของรัฐบาลสหรัฐแต่อย่างใด โดยยังคงต้องรอการผลสรุปที่แน่นอนอีกครั้งหนึ่ง ซึ่งอาจจะมีการรับรองในช่วงประมาณเดือนเมษายนถึงเดือนมิถุนายน ปี ค.ศ. 2001 นี้ก็เป็นได้

ขนาดของการเข้ารหัสลับแบบ AES นี้ AES จะกำหนดขนาดของรหัสลับเป็น 128, 192 และ 256 รหัสซึ่งสามารถแสดงค่าการสุ่มที่จะสุ่มได้ ดังนี้

ค่าการสุ่มที่สุ่มได้ 3.4×10^{38} ตัวอย่าง จากรหัส 128 บิต

ค่าการสุ่มที่สุ่มได้ 6.2×10^{57} ตัวอย่าง จากรหัส 192 บิต

ค่าการสุ่มที่สุ่มได้ 1.1×10^{77} ตัวอย่าง จากรหัส 256 บิต

และจากการเปรียบเทียบรหัสลับของการเข้ารหัสแบบ DES ที่ใช้รหัสลับจำนวน 56 บิต หรือสุ่มค่าได้ประมาณ 7.2×10^{16} ตัวอย่าง ซึ่งจะเห็นได้ว่า ต้องมีการสั่งให้ การเข้ารหัสแบบ DES ทำงานอีก 10^{21} ครั้ง กว่าจะได้การสุ่มค่าตัวอย่างที่เท่ากับ การเข้ารหัสแบบ AES

และถ้าให้เครื่องคอมพิวเตอร์ทำการค้นหารหัสลับของการเข้ารหัสแบบ DES (สมมุติให้สุ่มค่าด้วยอัตรา 2^{35} รหัสต่อวินาที) ก็ต้องใช้เวลาประมาณไม่ต่ำกว่า 20 ล้านปี ในการเจาะรหัส แต่ถ้าเป็นการเข้ารหัสแบบ AES ที่ใช้รหัสลับถึง 128 บิตก็คงต้องใช้เวลาประมาณ 149 พันล้านปี (149 trillion) ถึงสามารถจะเจาะรหัสได้

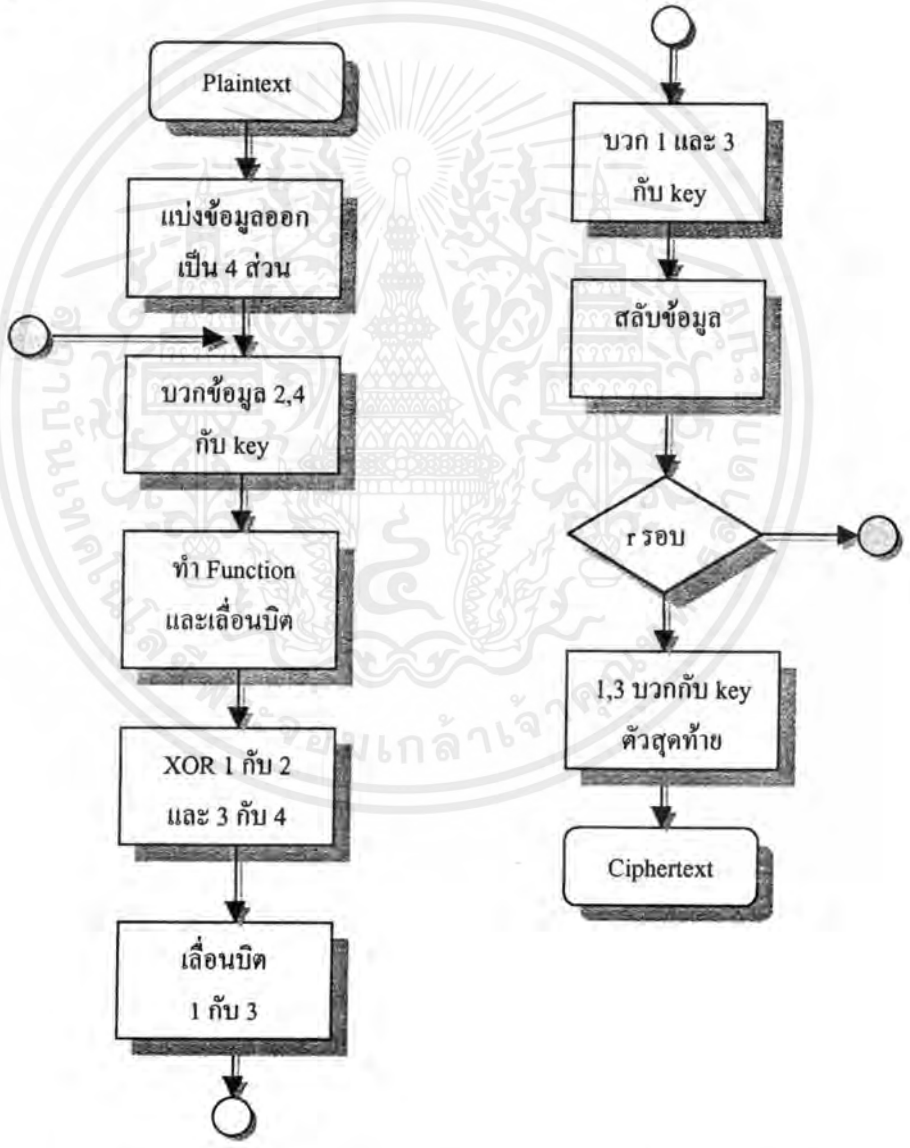
วิธีการการเข้ารหัสข้อมูลที่ได้ถูกคัดสรรไว้ว่าเป็นไปตามมาตรฐานการเข้ารหัสแบบใหม่ หรือ AES โดย NIST ก็เช่น

2.1.10 มาตรฐานการเข้ารหัสข้อมูลแบบ RC6

RC6 เป็นมาตรฐานการเข้ารหัส ที่ถูกออกแบบ โดย Dr. Ron Rivest เมื่อปี ค.ศ. 1995 ให้ได้เป็นไปตามมาตรฐานการเข้ารหัสแบบ Advance Encryption Standard (AES) ซึ่งเป็นการเข้ารหัสที่สามารถเข้ารหัสคำขนาด 128 บิตได้ โดยอาศัยการเข้ารหัสเป็นบล็อกๆละ 32 บิต หรือ 64 บิต โดยใช้รหัสลับขนาดที่ไม่แน่นอน ขนาด 16 -,32-,หรือ 64-ไบต์ต่อรหัส ซึ่งอาจสูงถึง 2048 บิต โดยข้อมูลดิบ

จะถูกแบ่งเป็นบล็อกขนาด 32 บิต จำนวน 4 บล็อก จนกว่าข้อมูลจะจบ แล้วถูกเข้ารหัสเป็นจำนวน r รอบด้วยกัน

การเข้ารหัสแบบ RC6 นี้สามารถป้องกันการถอดรหัสแบบ brute force หรือ การถอดรหัสแบบอาศัยความแตกต่าง (differential cryptanalysis) ได้เป็นอย่างดี ด้วยกระบวนการเข้ารหัสข้อมูลทำการเข้ารหัสเป็นจำนวน r รอบด้วยกันโดยจะแบ่งข้อมูลออกเป็นบล็อกละ 32 บิต จำนวน 4 บล็อกนี้เอง ทำให้ใช้เวลาในกระบวนการเข้ารหัสที่รวดเร็วกว่าการเข้ารหัสแบบ RC5 ที่ได้ถูกพัฒนาขึ้นก่อนซึ่งใช้บล็อกข้อมูลในการเข้ารหัสเพียง 2 บล็อกเท่านั้น ทำให้การเข้ารหัสแบบ RC6 มี ความเร็วกว่าถึงสองเท่าด้วยกัน



รูปที่ 2.6 แสดง Flow Chart การทำงานของการเข้ารหัสลับแบบ RC6

2.1.11 การเข้ารหัสลับแบบ Twofish

Twofish เป็นการเข้ารหัสลับชนิด Symmetric Encryption แบบ Block Cipher ที่ถูกพัฒนาขึ้นโดย Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hill และ Niels Ferguson ตามมาตรฐานการเข้ารหัสลับแบบใหม่ AES (Advance Encryption Standard) โดยที่ทำการเข้ารหัสด้วยรหัสลับขนาด 128 บิต ซึ่งรหัสลับนี้สามารถเพิ่มให้ยาวได้ถึง 256 บิต ซึ่งเป็นไปตามความต้องการของมาตรฐาน AES ที่ต้องการเข้ารหัสลับด้วยรหัสขนาด 128, 192, และ 256 บิต

การเข้ารหัสจะทำการเข้ารหัสลับจำนวน 16 รอบ ด้วยฟังก์ชัน F ที่สร้างจาก รหัสลับที่ขึ้น อยู่กับ S-box ขนาด 8×8 บิต ซึ่ง รหัสลับขนาด 4×4 บิตที่ได้ขึ้นมานี้จะได้ค่าจากเมตริกซ์ GF (2^8), การแปลงฮาร์มาร์ดแบบสุ่ม, กระบวนการหมุนวนของรหัสลับ และรหัสที่ถูกออกแบบไว้จากตารางรหัส โดยการเข้ารหัสแบบ Twofish นี้ สามารถใช้งานบนเครื่อง Pentium Pro ที่ 17.8 clock cycle ต่อ ไบต์ และไมโครโปรเซสเซอร์ smart card ขนาด 8 บิต เมื่อเข้ารหัสที่ ความเร็ว 1820 clock cycle ต่อ ไบต์ และ Twofish จะสามารถใช้งานใน hardware ขนาด 14,000 gates ได้ รวมทั้งการออกแบบฟังก์ชันของการเข้ารหัสและตารางคีย์ สามารถที่จะใช้ได้กับเครื่องที่มีความเร็วต่างๆกัน, ขนาดของโปรแกรม, เวลาในการติดตั้ง และขนาดของหน่วยความจำที่แตกต่างกันได้

2.1.12 การเข้ารหัสลับแบบ MARS

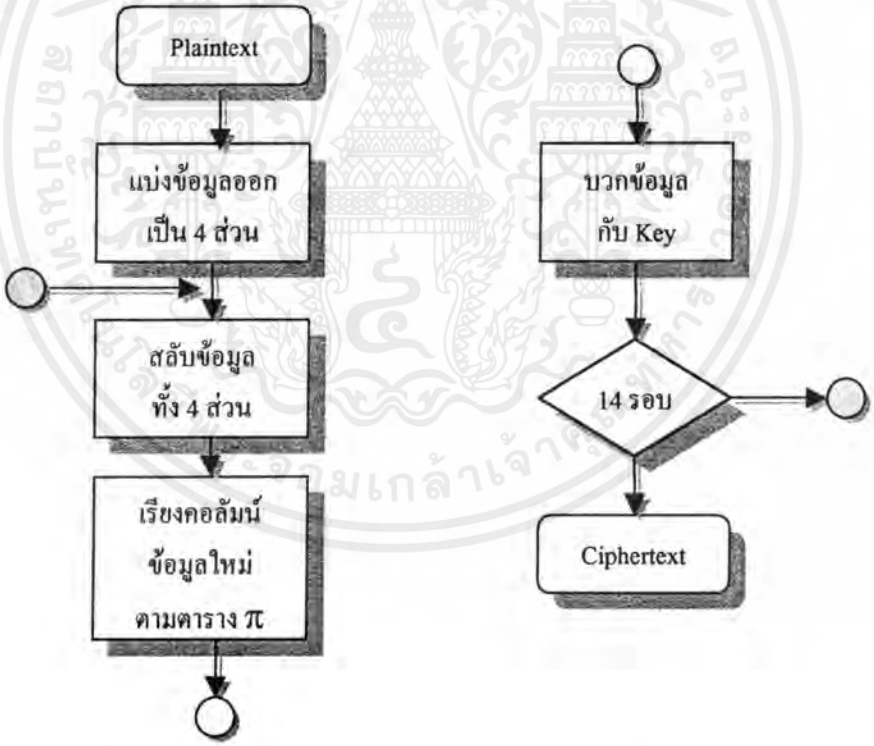
MARS เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher ที่ถูกพัฒนาขึ้นโดย IBM เพื่อให้มีรูปแบบเป็นไปตามมาตรฐานการเข้ารหัสแบบใหม่ หรือ AES และก็ได้รับเลือกเป็น 1 ใน 5 รูปแบบสุดท้ายจาก NIST โดย MARS เป็นการเข้ารหัสลับที่มีระดับความปลอดภัยสูงและมีความสามารถมากกว่าอัลกอริทึมแบบเดิมๆ โดยสามารถป้องกันการโจมตีแบบ shortcut ได้ และ MARS ยังเป็นการออกแบบระบบเพื่อรองรับกับคอมพิวเตอร์รุ่นใหม่ๆให้สามารถทำงานได้อย่างเต็มประสิทธิภาพสูงสุด โดยการรวมเอาเทคนิคของการเข้ารหัสต่างๆเข้าไว้ด้วยกัน แต่ก็เป็นการง่ายที่จะเข้าใจได้ ซึ่งโครงสร้างของระบบจะประกอบด้วย 2 ส่วนสำคัญๆ ที่แตกต่างกัน ทั้งนี้ก็เพื่อถ้าหากว่ามีการโจมตีเกิดขึ้นที่ส่วนหนึ่งก็จะไม่กระทบกับอีกส่วนหนึ่งด้วย และ การรวมอัลกอริทึมต่างๆเข้าด้วยกันนั้นก็จะสามารถป้องกันการโจมตีแบบใหม่ๆ ในอนาคตได้

MARS เป็นการเข้ารหัสข้อมูลชนิด block cipher ที่ทำการเข้ารหัสข้อมูลเป็นกลุ่ม กลุ่มละ 128 บิต โดยใช้รหัสลับที่มีความยาวไม่แน่นอน จาก 128 บิต ไปจนถึง 400 บิต แต่ก็เป็นไปตามความต้องการของ AES คือใช้รหัสลับขนาด 128, 192, หรือ 256 บิต อย่างหนึ่งอย่างใด โดยการเข้ารหัสลับจะทำการเข้ารหัสจำนวน 16 รอบ หรือมากกว่านั้น ซึ่งจะทำให้การเจาะรหัสลับเป็นไปได้ยาก ทั้งนี้ MARS จะใช้รหัสลับย่อยขนาด 448 บิต ซึ่งจะทำให้แน่ใจได้ว่ามีความปลอดภัยสูงขึ้น

2.1.13 การเข้ารหัสลับแบบ Rijndael

Rijndael เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher ที่ถูกออกแบบโดย Joan Daeman และ Vincent Rijmen เพื่อให้เป็นไปตามเงื่อนไขของมาตรฐานการเข้ารหัสแบบใหม่ หรือ AES และเป็น 1 ใน 5 ของวิธีการเข้ารหัสที่ได้รับเลือกจาก NIST ส่วนชื่อของวิธีการเข้ารหัสชนิดนี้ก็พบว่ามาจากชื่อสกุลของผู้ออกแบบทั้งสอง Daeman และ Rijmen โดยการเข้ารหัสลับแบบ Rijndael นี้ เป็นการออกแบบขึ้นมาโดยอ้างอิงวิธีการเข้ารหัสแบบ Square ซึ่งทั้งสองได้พัฒนาขึ้นมาก่อน เพื่อต้องการให้การเข้ารหัสลับแบบ Rijndael นี้เป็นการเข้ารหัสข้อมูลที่มีประสิทธิภาพสูงมากสำหรับ processor และ hardware แบบต่างๆ

Rijndael จะเป็นการเข้ารหัสข้อมูล โดยใช้รหัสลับที่มีความยาวไม่คงที่ ซึ่งอาจเป็น 128, 192, หรือ 256 บิต ต่อรหัสลับรหัสหนึ่ง และทำการเข้ารหัสข้อมูลทีละกลุ่ม กลุ่มละ 128, 192, หรือ 256 บิต เช่นกัน โดยทั้งขนาดของกลุ่มข้อมูลและความยาวของรหัสลับจะสามารถขยายเพิ่มขึ้นอย่างง่ายๆ โดยเพียงคูณกับ 32 บิต ทั้งนี้เพื่อให้รอบของการเข้ารหัสอยู่ที่ประมาณ 10 ถึง 14 รอบ



รูปที่ 2.7 แสดง Flow Chart การเข้ารหัสลับแบบ Rijndael

2.1.14 การเข้ารหัสลับแบบ Serpent

Serpent เป็นการเข้ารหัสลับแบบ symmetric ชนิด block cipher ที่ถูกออกแบบโดย Ross Anderson, Eli Biham และ Lars Knudsen เพื่อให้เป็นไปตามเงื่อนไขของมาตรฐานการเข้ารหัสแบบ

ใหม่ โดยการเข้ารหัสแบบ Serpent นี้ จะมีความเร็วในการเข้ารหัสมากกว่าการเข้ารหัสลับแบบ DES และมีความปลอดภัยสูงกว่า Triple DES สำหรับอัลกอริทึมของการเข้ารหัสลับแบบ Serpent นี้จะถูกออกแบบมาให้เป็นเสมือนระบบกลไกการเข้ารหัสลับ(cryptography mechanism) เพื่อที่จะป้องกันเทคนิคการเจาะรหัสลับแบบใหม่ในอนาคต

และสำหรับมาตราฐานการเข้ารหัสลับแบบใหม่ AES จะกำหนดให้รหัสลับของแต่ละวิธีคือ 128, 192, หรือ 256 บิต โดย Serpent นี้จะใช้รหัสลับที่มีขนาดที่ไม่แน่นอนอาจจะเป็น 128, 192, หรือ 256 บิต และจะทำการเข้ารหัสถึง 32 รอบ ซึ่งสูงกว่าการเข้ารหัสที่เป็นมาตรฐานของ AES แบบอื่นๆ จึงทำให้เป็นที่มั่นใจได้ว่ามีความปลอดภัยสูงเพียงพอ

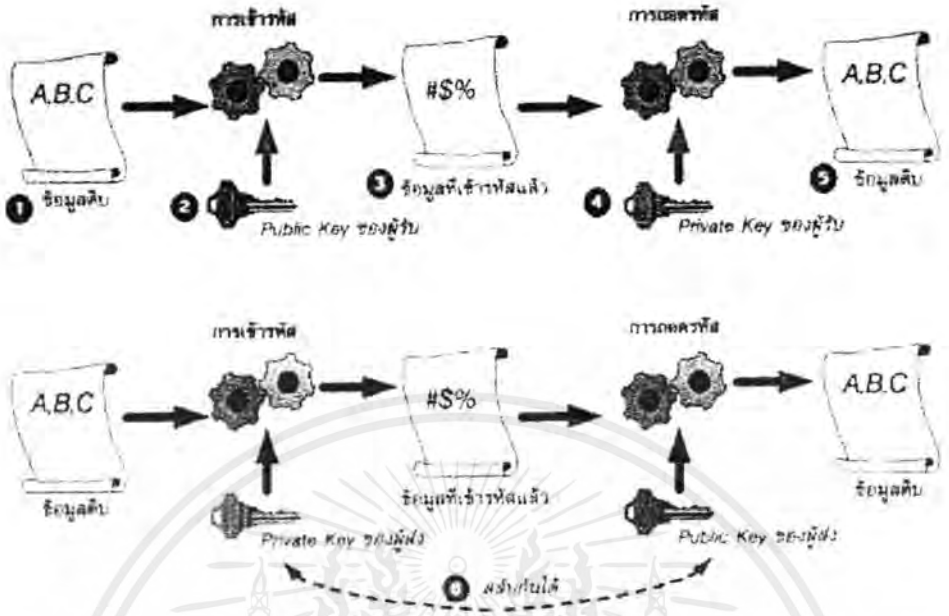
Serpent ได้ถูกแก้ไขเพิ่มเติมโดย 5th International Workshop ด้วยซอฟต์แวร์การเข้ารหัสที่มีรวดเร็วสูงมาก เราเรียก Serpent ที่ใช้ DES S-box เพื่อเปลี่ยนตารางรหัสลับและเปลี่ยนค่าที่ได้ใน S-box ด้วยว่า Serpent 1 และเรียก Serpent ที่ทำงานตามวิธีการที่ได้ออกแบบไว้แต่แรกว่า Serpent 0

รูปที่ 2.8 แสดง Flow Chart การเข้ารหัส แบบ Serpent

2.2 Asymmetric Encryption

Asymmetric Encryption ถูกคิดค้นขึ้นในปี ค.ศ. 1976 โดย Whitfiled Diffie และ Martin Hellman เพื่อแก้ไขข้อเสียของการเข้ารหัสแบบ Symmetric Encryption ที่ใช้รหัสลับตัวเดียวในการเข้ารหัสและถอดรหัส โดย Asymmetric Encryption จะเป็นการเข้ารหัสข้อมูลโดยมีรหัสที่เกี่ยวข้องอยู่ 2 รหัส คือ Public key และ Private key ซึ่งรหัสนี้จะต้องเข้าคู่กันเสมอ โดยจับเป็นคู่ของใครของมัน การเข้ารหัสข้อมูลจะนำข้อมูลดิบมาเข้ารหัสโดยใช้ Public key เป็นรหัสลับ เมื่อได้ข้อมูลที่เข้ารหัสแล้ว ผู้รับจะถอดรหัสได้โดยใช้ Private key ที่เข้าคู่กันกับ Public key มาทำการถอดรหัส ซึ่งจะไม่สามารถใช้ Public key ตัวเดิมหรือ Private key ของคู่อื่นมาถอดรหัสข้อมูลได้ เนื่องจาก Public key และ Private key นี้ถูกเชื่อมโยงกันด้วยสูตรทางคณิตศาสตร์ เพื่อให้เข้าคู่กันได้โดยไม่ซ้ำกับคู่อื่น ในการเข้ารหัสข้อมูลเราควรเปิดเผย Public key ให้คนอื่นๆทราบด้วย เพื่อจะได้นำไปใช้ในการเข้ารหัสข้อมูลแล้วส่งมาให้เราได้ ซึ่งอาจจะมีเพียงเราคนเดียวเท่านั้นที่ทราบ Private key ที่จะใช้ในการถอดรหัสข้อมูลได้ ด้วยวิธีการนี้จะเห็นว่าเราไม่จำเป็นต้องส่ง Private key ไปให้ผู้อื่นใช้ในการเข้ารหัส เพราะการเข้ารหัสจะใช้ Public key ที่เปิดเผยได้เท่านั้น ดังนั้นรหัสลับของเราจึงปลอดภัยแน่นอนเนื่องจากไม่ต้องส่งออกไปให้ผู้อื่นทราบ

สิ่งที่น่าสนใจของ Asymmetric Encryption คือ Public key และ Private key จะใช้งานสลับกันได้ หมายถึงถ้าเรามีรหัสลับ Public key และ Private key หนึ่งคู่ เราอาจใช้ Private key เป็นรหัสลับในการเข้ารหัสข้อมูลและใช้ Public keyที่เป็นคู่ของมันมาทำการถอดรหัสข้อมูลก็ได้และคุณสมบัติอีกอย่างหนึ่งของ Asymmetric Encryption ก็คือถึงแม้ว่าเราจะทราบ Public keyที่ใช้ในการเข้ารหัสข้อมูลแต่ก็เป็นการยากที่จะนำ Public key นั้นมาคำนวณย้อนกลับไปหา Private key ที่เป็นคู่ของมัน ดังนั้นถึงแม้ว่าเราจะรู้ Public key และมีข้อมูลที่เข้ารหัสแล้วอยู่ในมือ แต่เราก็ไม่สามารถย้อนกลับไปคำนวณหา Private key และไม่สามารถถอดรหัสข้อมูลนั้นได้ ในการใช้งานการเข้ารหัสข้อมูล เราจึงสามารถเปิดเผย Public key ที่ใช้เข้ารหัสได้ มีเพียงผู้รับข้อมูลเท่านั้นที่จะต้องทราบ Private key เพื่อใช้ในการถอดรหัสข้อมูล ดังรูป



รูปที่ 2.9 แสดงการเข้ารหัสลับแบบ Asymmetric Encryption หรือ การเข้ารหัสลับแบบคีย์คู่

1) เริ่มจากข้อมูลดิบที่ต้องการเข้ารหัสลับ
2) ทำการเข้ารหัสข้อมูลโดยใช้ Public key ของผู้รับ
3) ได้รับข้อมูลที่เข้ารหัสออกมาเพื่อส่งให้ผู้รับ
4) ผู้รับนำข้อมูลที่ได้รับมาถอดรหัสโดยใช้ Private key ของผู้รับ
5) ผู้รับจะได้ข้อมูลดิบกลับมาตามต้องการ
6) ในการเข้ารหัสลับแบบ Asymmetric Encryption เราสามารถสลับเอา Private key มาเข้ารหัสข้อมูลแล้วใช้ Public key ถอดรหัสข้อมูลก็ได้

ในส่วนที่เราสลับรหัสลับที่ใช้ในการเข้ารหัส คือใช้ Private key เป็นรหัสลับในการเข้ารหัสข้อมูล และใช้ Public key เป็นตัวถอดรหัส เราจะทราบได้ทันทีว่าใครเป็นผู้ส่งข้อมูลนี้มา เนื่องจากมีเพียงผู้ส่งเท่านั้นที่จะทราบ Private key ของตัวเอง ดังนั้นหากเราได้รับข้อมูลจากใครคนหนึ่ง แล้วใช้ Public key ของเขาถอดรหัสข้อมูลออกมาได้แสดงว่าผู้ที่ส่งข้อมูลมาให้เราก็คือเจ้าของ Private key นั่นเอง ซึ่งคนอื่นๆที่รู้ Public key ก็สามารถถอดรหัสอ่านข้อมูลได้เช่นเดียวกัน แต่จะไม่ใช่ผู้เข้ารหัสข้อมูลนี้อย่างแน่นอน

ข้อเสียของการใช้ Asymmetric Encryption ก็คือการเข้ารหัสและถอดรหัสจะเสียเวลานานกว่าการเข้ารหัสแบบ Symmetric Encryption เนื่องจากมีความซับซ้อนสูงกว่า และหากเราเลือกใช้รหัสลับ

ที่มีความยาวน้อยเกินไป อาจมีผู้นำ Public key และข้อมูลที่เข้ารหัสแล้วไปคำนวณย้อนหา Private key ได้ การเข้ารหัสข้อมูลแบบ Asymmetric Encryption ที่นิยมแพร่หลายได้แก่

2.2.1 การเข้ารหัสลับแบบ RSA

RSA เป็นมาตรฐานการเข้ารหัสข้อมูลที่คิดค้นโดยบุคคล 3 คน คือ Ron Rivest, Adi Shamir และ Leonard Adleman ดังนั้น ชื่อ RSAจึงได้มาจากอักษรตัวแรกในนามสกุลของทั้งสามคนนั่นเอง ปัจจุบันได้มีการจัดตั้งเป็นบริษัท RSA Data Security ซึ่งมีผลิตภัณฑ์ทางด้านซอฟต์แวร์เกี่ยวกับการเข้ารหัสข้อมูลต่างๆ และบริการให้คำปรึกษาเกี่ยวกับความปลอดภัยของข้อมูลและอื่นๆอีกมาก

RSA จะประกอบด้วยรหัส 2 ตัว คือ Public key และ Private key ที่ใช้สำหรับเข้ารหัสและถอดรหัสตามลำดับ ซึ่ง Public key นี้สามารถเปิดเผยได้ แต่ Private key จะเป็นความลับเฉพาะผู้ส่งเท่านั้น นอกจากนี้ Public key และ Private key จะต้องเป็นคู่ของมันเท่านั้นจึงจะสามารถถอดรหัสได้ RSAได้กำหนดรหัสลับของข้อมูลโดยใช้ Exponentiation Modulo (การหารเอาแต่เศษ)ของเลขจำนวนเต็มที่เรียกว่า "จำนวนเฉพาะ" (prime number) ซึ่งเป็นเลขจำนวนเต็มที่หารด้วยเลขอื่นไม่ลงตัว ยกเว้น 1 กับตัวเลขนั้นเองเท่านั้น เช่น 13,17,19, 29,..... โดยเลือก prime number สองตัวมาคูณกัน เพื่อหารหัสของ Public key และ Private key จากผลคูณที่ได้นี้ ซึ่ง Public key และ Private key ก็จะเป็นตัวเลข prime number จำนวนใดจำนวนหนึ่งที่ได้จากผลคูณของ Exponentiation Modulo ข้างต้น และใช้หลักการว่าการหารหัสลับของ Private key นั้น ยากเท่ากับการหาค่าตัวเลขตัวใดตัวหนึ่งที่ถูกต้องเพียงตัวเดียว จากผลคูณของ prime number สองตัว

RSA ผ่านการทดสอบจนเป็นที่ยอมรับว่ามีความปลอดภัยในการเข้ารหัสข้อมูลสูงมาก เมื่อเราเลือกความยาวของรหัสลับที่เหมาะสม เช่น ความยาวของรหัสลับขนาด 512 บิต นั้นถือว่ายังไม่มีความปลอดภัยมากนัก หากต้องการความปลอดภัยสูงความจะเป็น 1,024 บิต เป็นต้น แม้จะมีผู้กว่าว่าคอมพิวเตอร์ในปัจจุบันมีความเร็วสูงขึ้นมาก ทำให้การหา Private key ของ RSAโดยไล่ไปทีละค่าๆ สามารถทำได้ง่ายขึ้นในเวลาอันสั้น แต่ทว่าความเร็วของคอมพิวเตอร์ที่เพิ่มขึ้นนี้ ก็อาจจะเป็นตัวนำไปใช้เพิ่มความยาวของรหัสก็ได้ ซึ่งก็ทำให้การเดารหัสยากขึ้นตามไปด้วยเช่นกัน

2.2.2 การเข้ารหัสลับแบบ Rabin

Rabin เป็นมาตรฐานการเข้ารหัสข้อมูลแบบ Asymmetric ประกอบด้วยรหัส 2 ตัว คือ Public key และ Private key ที่ใช้สำหรับเข้ารหัสและถอดรหัสตามลำดับ ซึ่ง Public key หาได้จากการเทียบเคียงตัวประกอบที่คิดค้นโดย Michael Rabin ซึ่งได้อาศัยทฤษฎีของ Fermat's Little Theorem และ Euclid's Lemma เรียกว่า Miller's Test ซึ่งมีความแตกต่างจากของการเข้ารหัสแบบ RSA โดยสามารถรู้ค่าของ private key หรือค่าแปลกปลอมอื่นได้ เพราะได้กำหนดรหัสลับของข้อมูลโดยใช้การหารเอา

แต่เศษของเลขจำนวนเต็มที่เรียกว่า "จำนวนเฉพาะ" (prime number) โดยจะเลือกจำนวนเฉพาะสองจำนวนมาคูณกัน เป็น private key

และค่าของ Public key ซึ่งสามารถเปิดเผยให้ผู้รับข้อมูลทราบได้ สามารถหาได้จากรากที่สองของข่าวสาร ทำการหารเอาแต่เศษ หรือ modulo กับค่า n ซึ่งจะได้จากการเลือกจำนวนเฉพาะสองจำนวน P และ Q มาคูณกัน แต่อย่างไรก็ตามก็มีโอกาสเป็นไปได้แค่ ¼ ที่จะหาค่าของรากที่สองของข่าวสาร ทำการหารเอาแต่เศษ หรือ modulo กับค่า n ได้

และเป็นที่มั่นใจได้ว่าการเข้ารหัสแบบ Rabin มีความปลอดภัยสูงพอ ถึงแม้ว่าจะมีคนอื่นทราบค่า n ที่ใช้ในการเข้ารหัส แต่ก็ไม่สามารถทราบตัวประกอบที่ใช้ได้ เพราะ Rabin ใช้หลักการว่าการหารหัสลับของ private key นั้น ยากเท่ากับการหาค่าตัวเลขตัวใดตัวหนึ่งที่ถูกต้องเพียงตัวเดียวจากผลคูณของ prime number สองตัว

2.2.3 การเข้ารหัสลับแบบ LUC

Luc เป็นระบบการเข้ารหัสลับที่ถูกพัฒนาขึ้นโดยกลุ่มนักวิจัยในออสเตรเลียและนิวซีแลนด์ ที่ได้นำเสนอออกมาโดย Dr. Dobb's Journal เมื่อ ค.ศ. 19993 (See P. Smith and M. Lennon, "LUC: A new public-key system", Proceedings of IFIP/Sec '93, Elsevier Science Publications, 1993.)

ในการเข้ารหัสลับแบบ LUC นี้จะใช้ ลำดับของลูคัส (Lucus Sequences) ในการเข้ารหัสลับ ซึ่งจะให้ผลการเข้ารหัสออกมาคล้ายกันกับการเข้ารหัสลับแบบ ElGamal, Diffie-Hellman รวมทั้งแบบ RSAด้วย และจะพบคำว่า LUCELG ซึ่งก็จะหมายถึงการเข้ารหัสแบบ LUC ที่ให้ผลของการเข้ารหัสออกมาคล้ายกันกับการเข้ารหัสลับแบบ ElGamal และก็เช่นเดียวกัน LUCDIF และ LUCRSA ก็คือการเข้ารหัสลับแบบ LUC ที่ให้ผลออกมาเหมือนกับการเข้ารหัสแบบ Diffie-Hellman หรือ DH และการเข้ารหัสลับแบบ RSA

ลำดับของลูคัส (Lucus Sequences)ที่ใช้ในการเข้ารหัสลับแบบนี้จะหาค่าได้จากความสัมพันธ์เชิงเส้นของลำดับอันดับสอง คือ $T_n = PT_{n-1} - QT_{n-2}$ ซึ่ง P และ Q ก็คือจำนวนเฉพาะที่สัมพันธ์กัน โดยการเข้ารหัสลับแบบ LUC นี้จะคำนวณค้าที่เกิดขึ้นซ้ำๆกัน แทนที่จะใช้สมการเอ็กซ์โพเนนเชียลเหมือนกับการเข้ารหัสลับแบบ Diffie-Hellman หรือ DH และ RSA ซึ่งทำให้กระบวนการคำนวณเป็นได้ง่ายยิ่งขึ้น

บทที่ 3
ทฤษฎีที่ช่วยในการวิเคราะห์ข้อมูล

วิธีการเข้ารหัสลับแต่ละวิธี จะมีคุณสมบัติในการเข้ารหัสลับที่แตกต่างกัน ทำให้แต่ละวิธีการมีความสามารถในการเข้ารหัสที่แตกต่างกัน เช่น ความเร็วในการเข้ารหัสลับ, จำนวนของรหัสลับที่ใช้ในการเข้ารหัสแต่ละวิธี ,ขนาดของไฟล์ข้อมูลหลังการเข้ารหัสลับ และการกระจายข้อมูลที่ผิดพลาด ซึ่งต้องทำการวัดหาเปอร์เซ็นต์ความคลาดเคลื่อนที่เกิดขึ้น

1 การวัดความคลาดเคลื่อน

การวัดความคลาดเคลื่อนของค่าจริงและค่าที่เปลี่ยนแปลงไปหลังการเข้ารหัส โดยใช้สัมประสิทธิ์หรือข้อมูลต่างๆ มีความแตกต่างกัน ดังนั้น ค่าสัมประสิทธิ์ หรือจำนวนข้อมูลใดที่ทำให้ค่าผิดพลาดไปต่างจากค่าจริงมากที่สุด หรือทำให้เกิดความคลาดเคลื่อนของข้อมูลมากที่สุด ก็เหมาะสมกับการตัดสินใจเลือกวิธีการเข้ารหัสลับที่ดีที่สุดด้วย ซึ่งก็จะทำให้เห็นได้ว่าข้อมูล บิต 0 และ บิต 1 กระจายไปมากน้อยเพียงใด เพราะถ้าหากว่าข้อมูลมี ข้อมูลที่เปลี่ยนแปลงจากบิต 0 เป็น บิต 1 หรือ บิต 1 เป็น บิต 0 เป็น 0 ก็เสมือนว่าข้อมูลเพียงถูกกลับลอจิกกันเท่านั้น ทำให้สามารถเดาข้อมูลก่อนทำการเข้ารหัสได้ง่ายขึ้น

การวัดความคลาดเคลื่อน สามารถวัดได้จาก

$$e = Y_n - X_n$$

เมื่อ e คือ ความคลาดเคลื่อน absolute error

Y_n คือ ค่าที่แท้จริง

X_n คือ ค่าที่เปลี่ยนแปลงไป

และเปอร์เซ็นต์ ความคลาดเคลื่อน Percent Error หาได้จาก

$$\%e = \frac{Y_n - X_n}{Y_n} \times 100$$

ซึ่งในการเปรียบเทียบการเข้ารหัสแต่ละวิธี ถ้ามี%e มากก็แสดงว่ามีการเปลี่ยนแปลงของข้อมูลที่สูง ทำให้สามารถแน่ใจว่ามีความปลอดภัยของข้อมูลที่เข้ารหัสสูง และ เหมาะสมเป็นวิธีการเข้ารหัสลับที่ผู้ใช้จะเลือกนำไปใช้ หากต้องการการกระจายข้อมูลที่สูง

2 การกระจายข้อมูลที่ผิดพลาด (Error Propagation)

การกระจายข้อมูลไบนารี่ ของ plaintext Caligula และผลลัพธ์ที่ได้จากการเข้ารหัสลับข้อมูลด้วย mini-DES เป็น B'8821}$

ตารางที่ 3.1 แสดงการกระจายข้อมูลที่ผิดพลาดของการเข้ารหัสด้วยของ plaintext **Caligula** และผลลัพธ์ที่ได้จากการเข้ารหัสลับข้อมูลด้วย mini-DES เป็น **B'8821}$**

Bit	Plaintext		Ciphertext		% Change on plaintext
	Number	%	Number	%	
0	33	51.6	40	62.5	+21.2
1	31	48.4	24	37.5	-22.6

การเปลี่ยนแปลงของอัตราส่วนของ บิต 0 และ บิต 1 นี้จะเรียกว่า การกระจายข้อมูลที่ผิดพลาด (Error Propagation) ทั้งนี้เพราะ เป็นการทำงานแบบทางเดียว และอีกอย่างหนึ่ง คือ การผิดพลาดของข้อมูล 1 บิต จะทำให้เกิดความแตกต่างที่เอาต์พุตเป็นอย่างมาก โดยที่ หากเปลี่ยนอินพุตใดอินพุตหนึ่งเพื่อเลือกฟังก์ชันการทำงาน ค่าของตาราง E, ฟังก์ชันเลือกการทำงาน และค่า P ซึ่งถูกออกแบบมาก็จะเพิ่มการกระจายข้อมูลที่ผิดพลาดให้มากขึ้น เช่น การสุ่มรหัสลับขึ้นมาหนึ่งบิต เพื่อค้นหาข้อมูล ก็คงไม่ต่างอะไรกับการเดาข้อมูลนั่นเอง เพราะเป็นการยากที่จะเดารหัสลับได้ถูก และคงจะเอาไปใช้ประโยชน์อะไรไม่ได้

3 วิธีการตัดสินใจเลือกวิธีการเข้ารหัสลับ

วิธีการเลือกวิธีการเข้ารหัสที่เหมาะสม แตกต่างกันไปแล้วแต่กรณี แต่ในที่นี้จะนำเสนอวิธีการ คือ

3.1 วิธีให้คะแนน (Factor Rating Plan) วิธีนี้เป็นการชั่งน้ำหนักปัจจัยต่างๆ คือ ปัจจัยที่มีความสำคัญต่อการตัดสินใจเลือกวิธีการเข้ารหัสลับมากที่สุด เช่น สมมุติว่า ต้องการความเร็ว เป็นปัจจัยที่สำคัญมากที่สุด เราก็จะให้น้ำหนัก(weight)มากที่สุด และปัจจัยที่สำคัญน้อยๆลงมาเราก็จะให้น้ำหนักน้อยลงตามเช่นกัน

หลังจากกำหนดน้ำหนักของแต่ละปัจจัยแล้ว ก็จะพิจารณาให้คะแนนวิธีการเข้ารหัสลับแต่ละแบบ ตามความสำคัญที่ต้องการ เช่น ถ้าต้องการความเร็วสูงสุด ในทั้งหมดทุกวิธีที่เลือกมา วิธีการที่

เร็วที่สุด ก็จะได้คะแนนเต็ม คือ 10 และวิธีการอื่นๆ ก็จะได้ คะแนนลดหลั่นกันลงไป ดังตัวอย่างต่อไปนี้

หากต้องการตัดสินใจเลือกวิธีการเข้ารหัสลับข้อมูล โดยกำหนดว่า ได้รวบรวมข้อมูลจากการทำงานของวิธีการเข้ารหัสลับแต่ละวิธีเพื่อนำมาเป็นปัจจัยในการตัดสินใจ ค่าที่ได้ดังตารางข้างล่างนี้

ตารางที่ 3.2 แสดงการรวบรวมข้อมูลการทำงานของการเข้ารหัสลับที่ต้องการและกำหนดน้ำหนักความสำคัญของปัจจัยการเข้ารหัสลับแต่ละอย่าง

ปัจจัยในการเลือก	น้ำหนัก	ผลการทำงาน		
		DES	IDEA	Blowfish
ความเร็วในการเข้ารหัส	0.7	9.945	11.286	7.091
จำนวนรหัสลับ(byte)	0.2	8	16	16
การกระจายข้อมูล(%)	0.1	21.2	30.5	25.2

ทำการกำหนดคะแนนให้วิธีการเข้ารหัสแต่ละวิธีจากข้อมูลการทำงานของการเข้ารหัสลับวิธีนั้นๆ โดยจะกำหนดการให้คะแนนตามแต่ละปัจจัยว่าให้ข้อมูลเรียงจากมากไปหาน้อย หรือ น้อยไปหามาก

ตารางที่ 3.3 แสดงการให้คะแนนการเข้ารหัสลับแต่ละวิธี

ปัจจัยในการเลือก	น้ำหนัก	ผลการทำงาน		
		DES	IDEA	Blowfish
ความเร็วในการเข้ารหัส	0.7	9	8	10
จำนวนรหัสลับ(byte)	0.2	9	10	10
การกระจายข้อมูล(%)	0.1	8	10	9

ตารางที่ 3.4 แสดงคะแนนที่ทำการถ่วงน้ำหนักข้อมูลแล้ว

ปัจจัยในการเลือก	คะแนนที่ถ่วงน้ำหนักแล้ว		
	DES	IDEA	Blowfish
ความเร็วในการเข้ารหัส	6.3	5.6	7.0
จำนวนรหัสลับ	1.8	2.0	2.0
การกระจายข้อมูล	0.8	1.0	0.9
รวม	8.9	8.6	9.9

ผลการให้คะแนนปรากฏว่า วิธีการ DES ได้ 8.9 คะแนน IDEA ได้ 8.6 คะแนน และ Blowfish ได้คะแนน 9.9 คะแนน ดังนั้น วิธีการเข้ารหัสที่น่าจะเหมาะสมกับความต้องการของผู้ใช้มากที่สุด คือ Blowfish ที่ได้คะแนน 9.9 คะแนน

3.2 วิธีวิเคราะห์มิติ(Dimension Analysis)

ในวิธีนี้เราจะพิจารณาปัจจัยทั้งหมดที่วัดได้ และปัจจัยที่ไม่สามารถวัดออกมาเป็นตัวเลขได้ ปัจจัยต่างๆเหล่านี้จะมีความสำคัญมากหรือน้อยขึ้นอยู่กับการตัดสินใจให้น้ำหนักของเรา แต่วิธีวิเคราะห์มิตินี้มีข้อเสียคือ เปรียบเทียบความแตกต่างระหว่างของสองสิ่งได้เท่านั้น

$$\frac{\text{วิธี B}}{\text{วิธี A}} = \left(\frac{c_{A1}}{c_{B1}}\right)^{W_1} \times \left(\frac{c_{A2}}{c_{B2}}\right)^{W_2} \times \ldots \times \left(\frac{c_{An}}{c_{Bn}}\right)^{W_n}$$

โดยที่ $C_{A1},C_{A2},\ldots,C_{An}$ และ $C_{B1},C_{B2},\ldots,C_{Bn}$ คือ ข้อมูลสำหรับปัจจัยต่างๆที่ได้มาจากวิธีการเข้ารหัสลับ แต่ละวิธี ตามลำดับ และ W_i คือ ตัวเน้นความสำคัญของปัจจัยเหล่านั้น

ตารางที่ 3.5 แสดงการรวบรวมข้อมูลการทำงานของการเข้ารหัสลับที่ต้องการและกำหนดน้ำหนักความสำคัญของปัจจัยการเข้ารหัสลับแต่ละอย่าง

ปัจจัยในการเลือก	น้ำหนัก	ผลการทำงาน		
		DES	IDEA	Blowfish
ความเร็วในการเข้ารหัส	0.7	9.945	11.286	7.091
จำนวนรหัสลับ(byte)	0.2	8	16	16
การกระจายข้อมูล(%)	0.1	21.2	30.5	25.2

จากตารางการรวบรวมข้อมูลของการเข้ารหัสแต่ละวิธี เรานำมาพิจารณา เพื่อหาวิธีการเข้ารหัสที่ดีที่สุดโดยทำการวิเคราะห์ ด้วยวิธีวิเคราะห์มิติ ดังนี้

$$\frac{\text{IDEA}}{\text{DES}} = \left(\frac{9.945}{11.286}\right)^{0.7} \times \left(\frac{8}{16}\right)^{0.2} \times \left(\frac{21.2}{30.5}\right)^{0.1} = 0.768321$$

จากผลการวิเคราะห์มิติของ IDEA กับ DES จะได้ค่าออกมา คือ 0.768321 ซึ่งมีค่าน้อยกว่า 1 ดังนั้น เราจึงทำการเลือกวิธีเข้ารหัสที่อยู่ข้างล่าง คือ DES

แล้วทำการวิเคราะห์มิติต่อไป ระหว่าง Blowfish กับ DES จะได้

$$\frac{\text{Blowfish}}{\text{DES}} = \left(\frac{9.945}{7.091}\right)^{0.7} \times \left(\frac{8}{16}\right)^{0.2} \times \left(\frac{21.2}{25.2}\right)^{0.1} = 1.08293$$

ผลการวิเคราะห์มิติของ Blowfish กับ DES จะได้ค่าออกมา คือ 1.08293 ซึ่งมีค่ามากกว่า 1 ดังนั้น เราจึงทำการเลือกวิธีเข้ารหัสที่อยู่ข้างบน คือ Blowfish

และเพื่อให้แน่ใจว่า IDEA นั้นมีคุณสมบัติจากปัจจัยที่เรากำหนดน้ำหนักให้ ด้อยกว่าการเข้ารหัสแบบ Blowfish จึงหรือไม่ จึงทำการวิเคราะห์มิติต่อ

$$\frac{\text{IDEA}}{\text{Blowfish}} = \left(\frac{7.091}{11.286}\right)^{0.7} \times \left(\frac{16}{16}\right)^{0.2} \times \left(\frac{25.2}{30.5}\right)^{0.1} = 0.708643$$

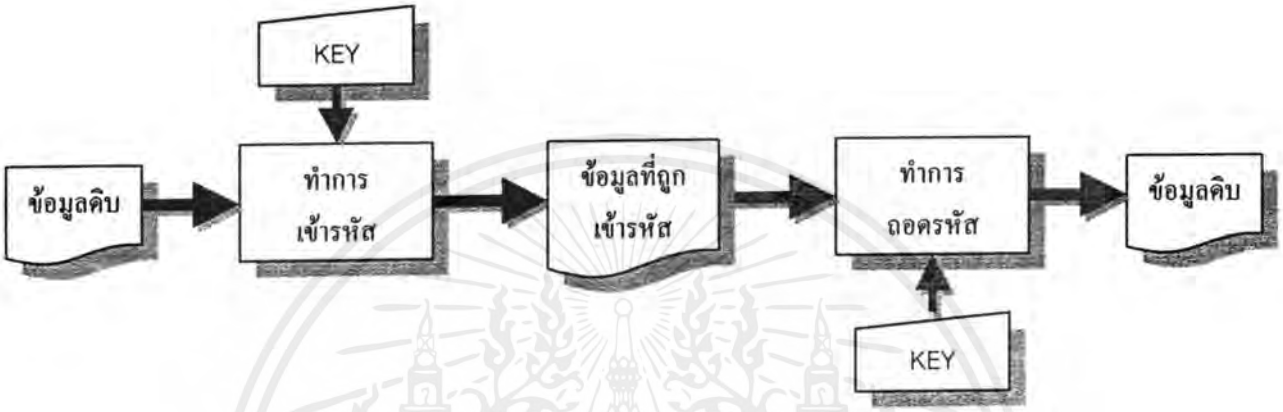
ผลการวิเคราะห์มิติของ IDEA กับ Blowfish จะได้ค่าออกมา คือ 0.708643 ซึ่งมีค่าน้อยกว่า 1 ดังนั้น เราจึงทำการเลือกวิธีเข้ารหัสที่อยู่ข้างล่าง คือ Blowfish

และจากการวิเคราะห์ทั้งสามครั้ง จะได้ว่า วิธีการเข้ารหัสแบบ Blowfish มีคุณสมบัติเหมาะสมตามปัจจัยที่เราได้กำหนดไว้ ตามมาด้วย DES และ IDEA ตามลำดับ ซึ่งได้ผลเป็นไปตามวิธีการให้คะแนนข้างต้น ดั้งนั้น จึงอาจจะพอสรุปได้ว่า วิธีการให้คะแนน ที่ใช้ในการตัดสินใจเลือกวิธีการเข้ารหัสลับ มีความแม่นยำพอสมควร

บทที่ 4

การออกแบบระบบ

4.1 System Flow Diagram



รูปที่ 4.1 แสดง System Flow Diagram ของระบบ

จาก System Flow Diagram สามารถอธิบายการทำงานได้ดังนี้

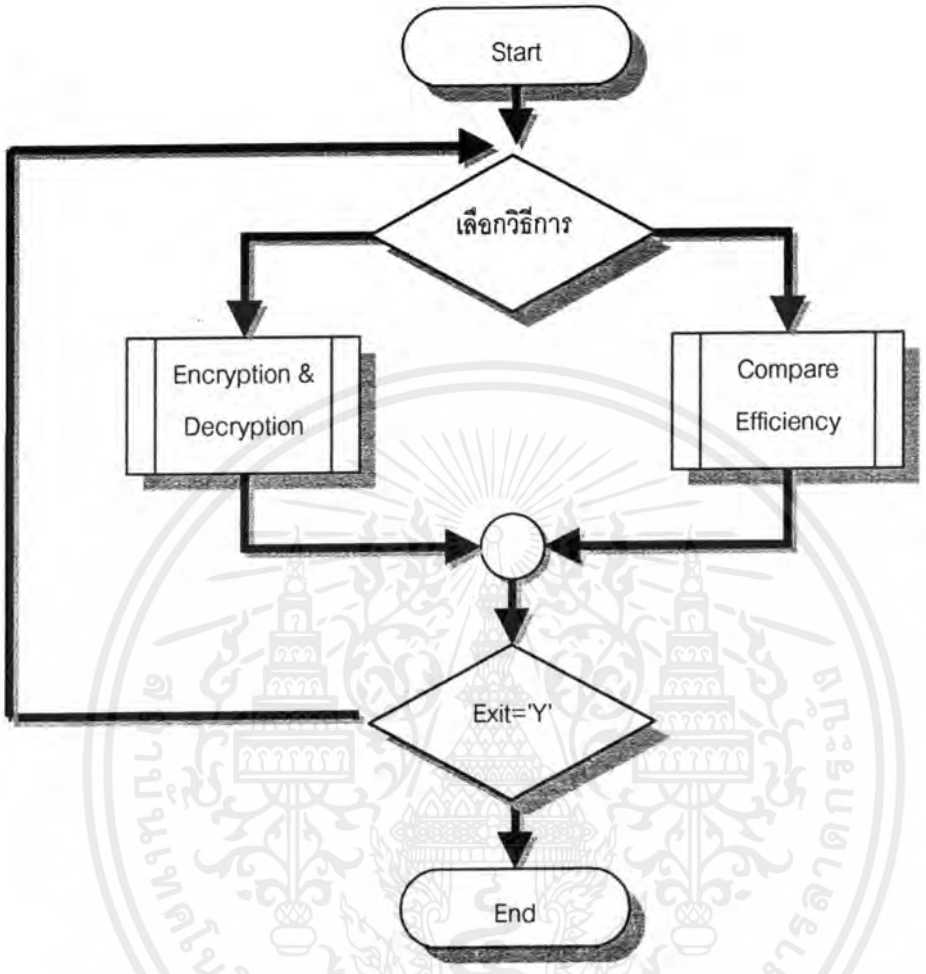
1 เริ่มที่ผู้ใช้กำหนดไฟล์ข้อมูล (plain text) จะทำการเข้ารหัสมาให้ และป้อนค่า Key ที่จะใช้ในการเข้ารหัส

2 ทำการเข้ารหัสไฟล์จาก Key ที่กำหนดให้ ผลลัพธ์ที่ได้จะถูกเขียนลงไฟล์ output ที่ได้ทำการเข้ารหัสแล้ว

3 เมื่อจะทำการถอดรหัส ก็นำไฟล์ข้อมูลที่ได้เข้ารหัสไว้แล้ว และรับ Key จากผู้ใช้ไปทำการถอดรหัสข้อมูล

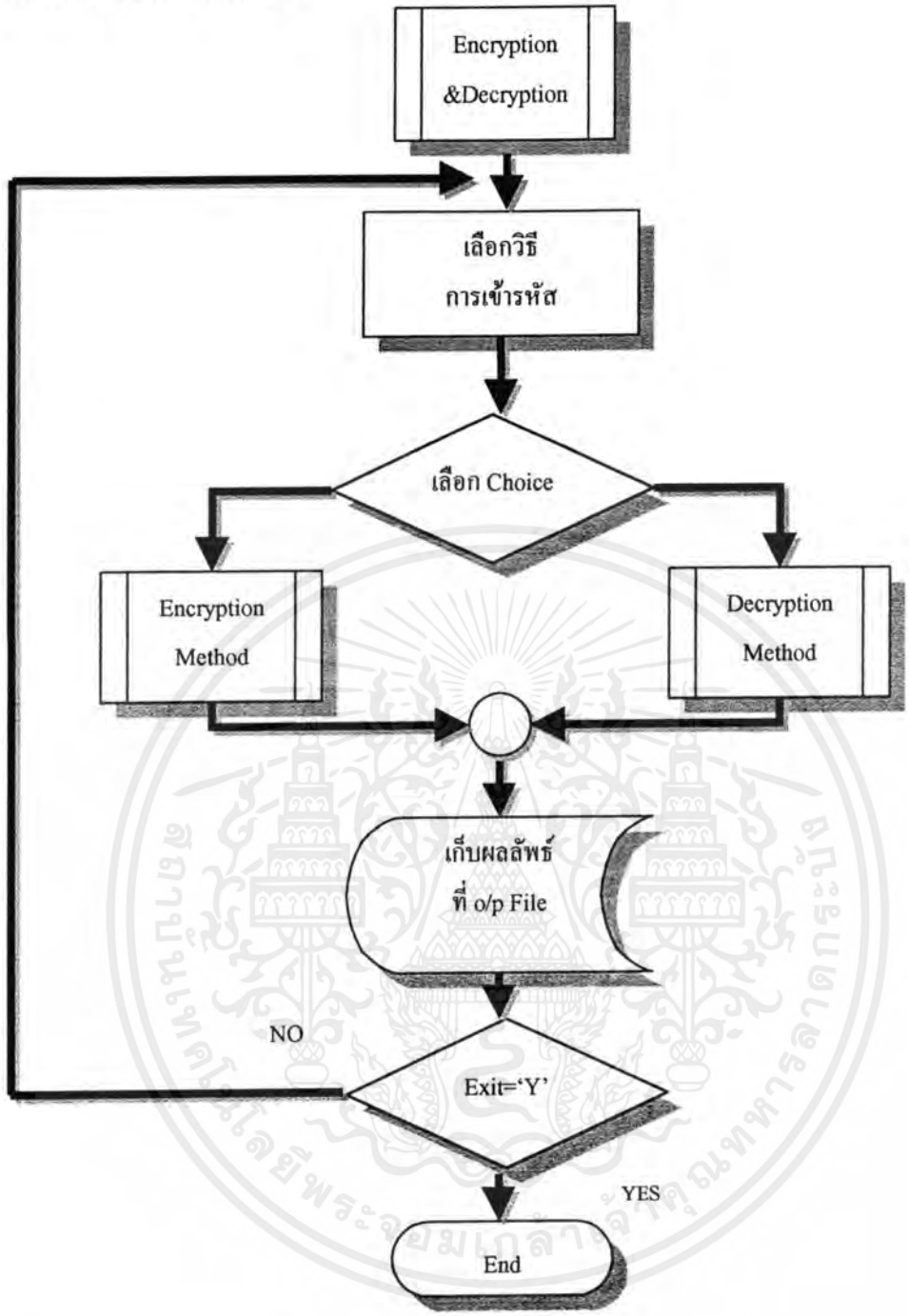
4 ผลลัพธ์ที่ได้ จะถูกเก็บไว้เป็นไฟล์ข้อมูลปกติ

4.2 Flow Chart



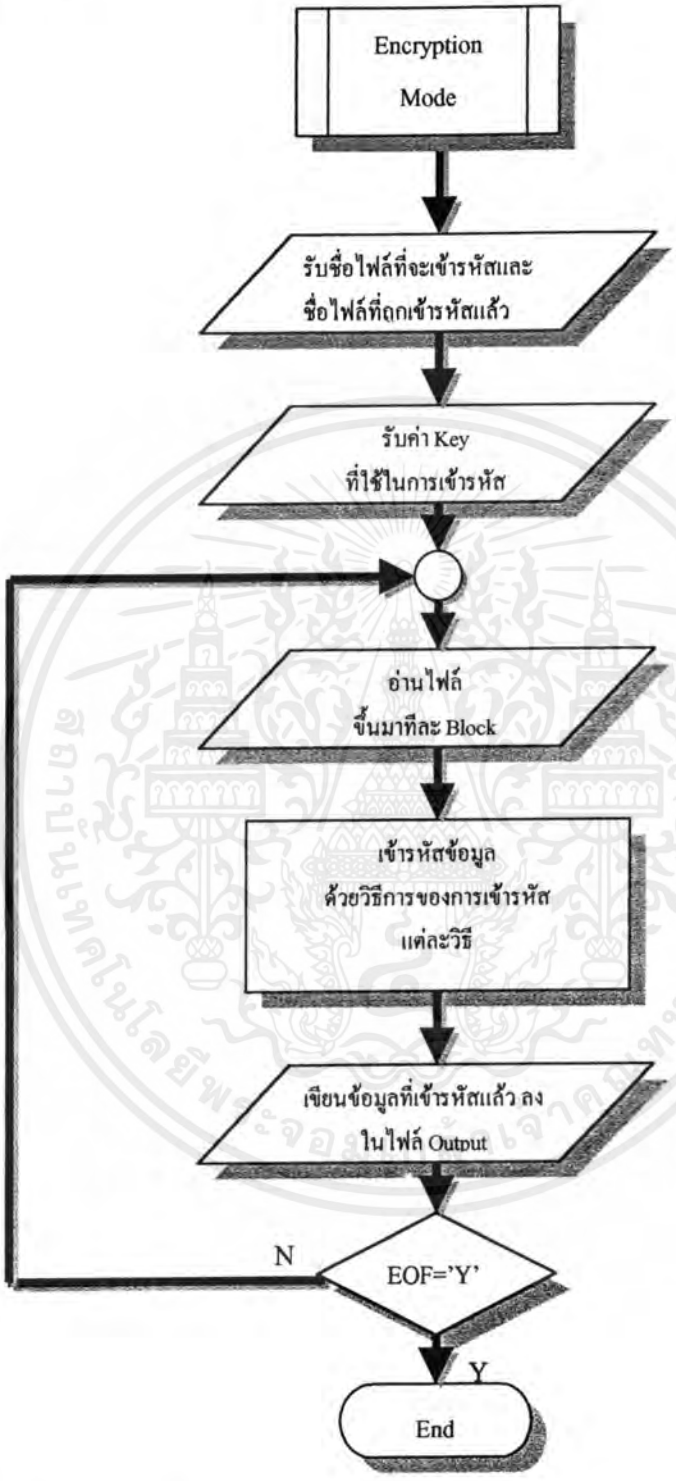
รูปที่ 4.2 แสดง Flow Chart ของระบบ

ส่วนการเข้ารหัสและถอดรหัส



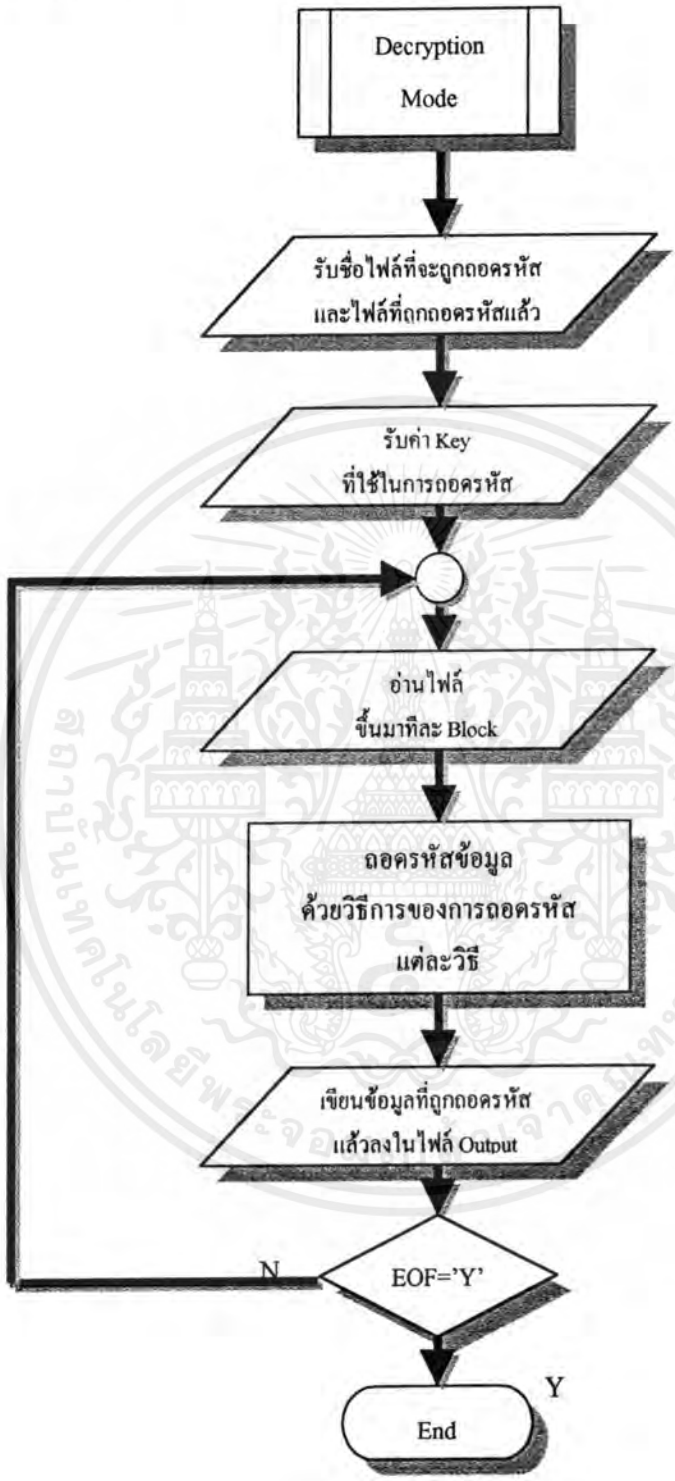
รูปที่ 4.3 แสดง Flow Chart ของการเข้ารหัส และการถอดรหัส

ส่วนการเข้ารหัส



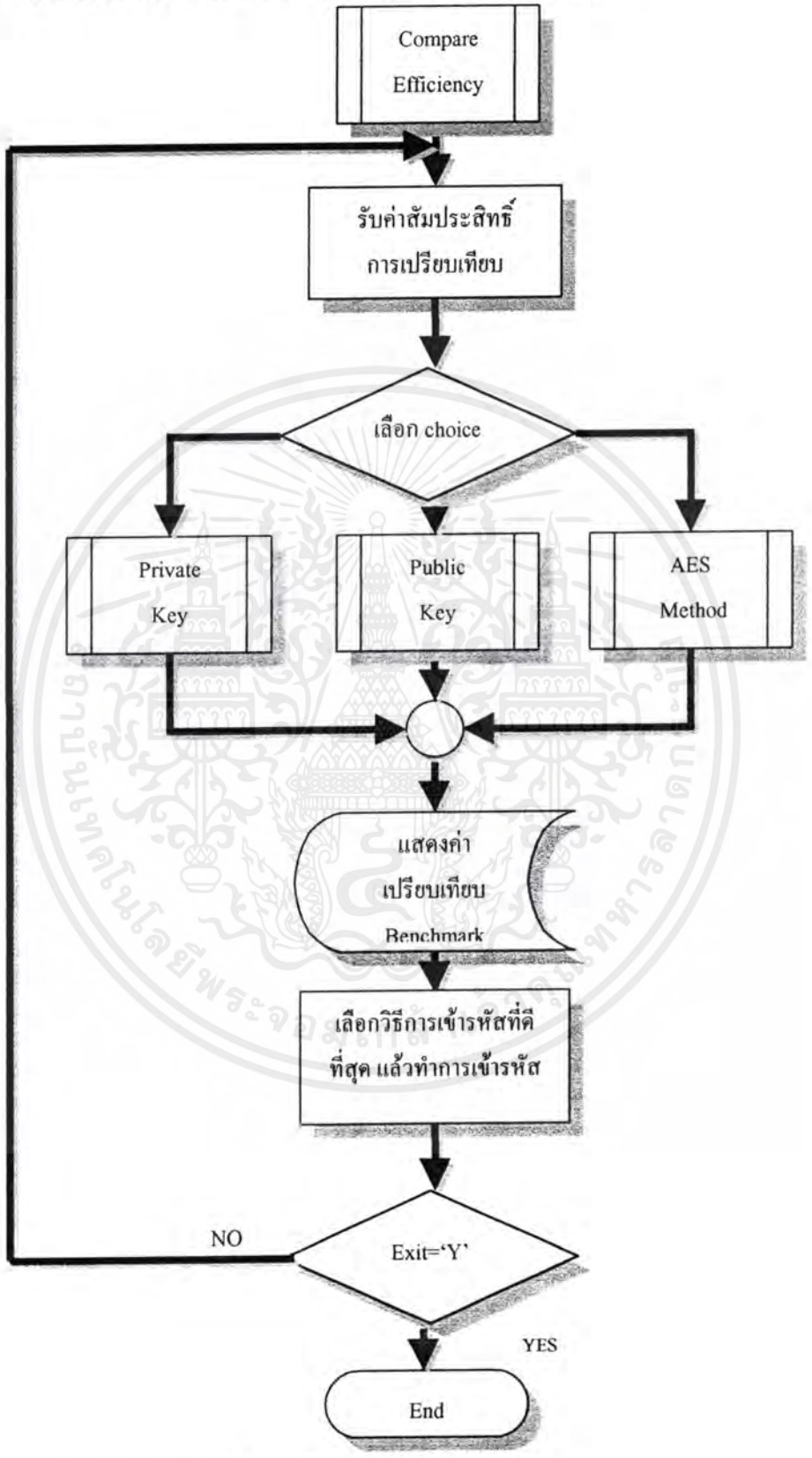
รูปที่ 4.4 แสดง Flow Chart ของการเข้ารหัส

ส่วนของการถอดรหัส



รูปที่ 4.5 แสดงส่วนของการถอดรหัส

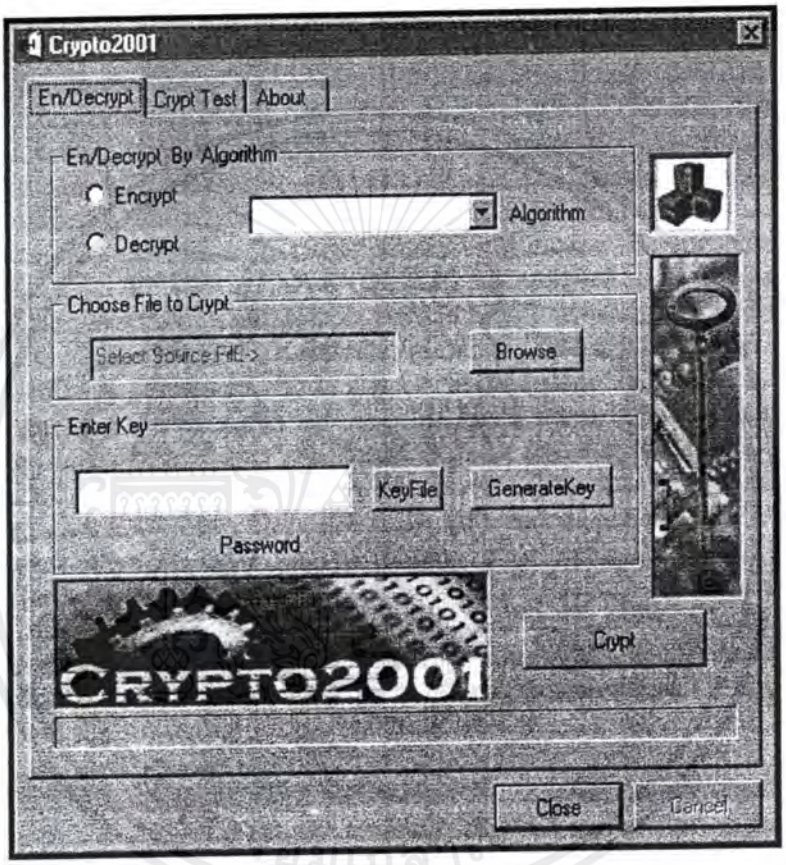
ส่วนการเปรียบเทียบสัมประสิทธิ์และการตัดสินใจเลือกวิธีการเข้ารหัส



รูปที่ 4.6 แสดง Flow Chart ของส่วนเปรียบเทียบการเข้ารหัส

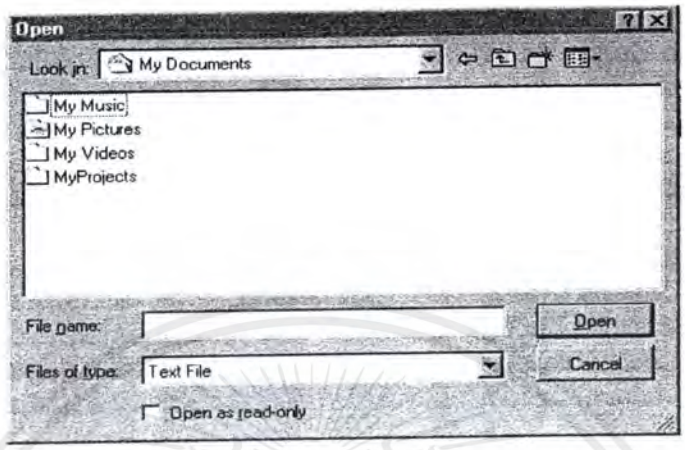
หน้าจอของโปรแกรมที่ติดต่อกับผู้ใช้

1 หน้าจอสำหรับการเข้ารหัสลับและถอดรหัสลับ (Encryption and Decryption) เมื่อทำการเลือกวิธีการเข้ารหัสลับและถอดรหัสลับ โดยจะแสดง ส่วนการรับชื่อไฟล์ที่จะทำการเข้ารหัสและถอดรหัสข้อมูล ส่วนของการเลือกว่าจะทำการเข้ารหัสหรือถอดรหัสข้อมูลด้วยวิธีการเข้ารหัสข้อมูลแบบต่างๆ รวมทั้งแสดงหน้าจอการรับ Key รหัสลับ และ process bar



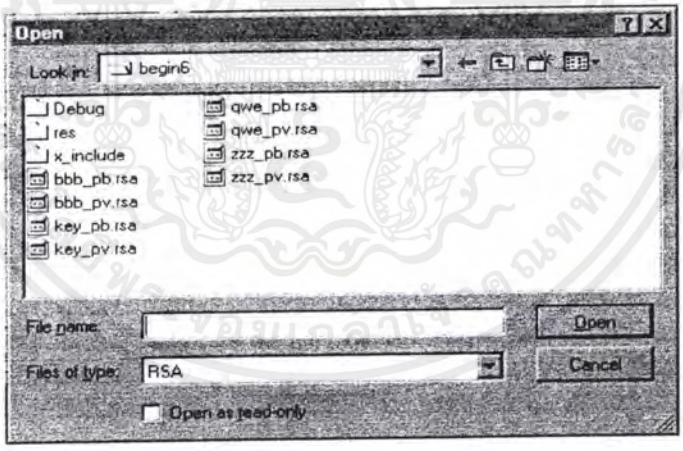
รูปที่ 4.7 แสดงส่วนของหน้าจอที่ใช้ในการเข้ารหัสและถอดรหัสลับข้อมูล

2 เมนูไฟล์(File) เมื่อเลือก Selected File



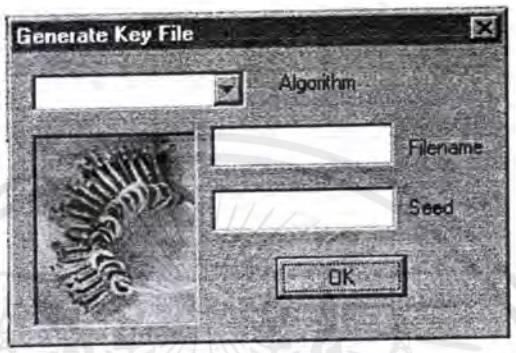
รูปที่ 4.8 แสดงหน้าจอการเลือกไฟล์ข้อมูลที่จะเข้ารหัสลับ

3 เมนูไฟล์เมื่อเลือก Key File



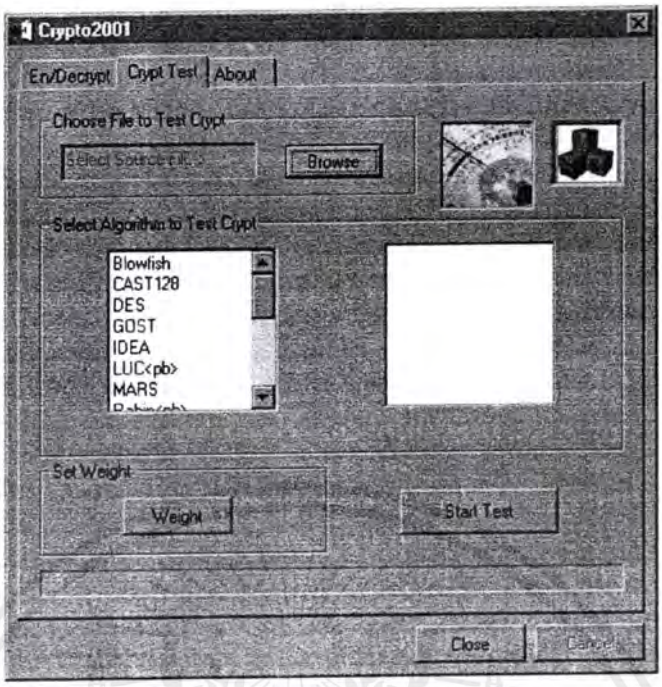
รูปที่ 4.9 แสดงหน้าจอการสร้างไฟล์ข้อมูลที่ใช้เป็นรหัสลับ

4 เมนู Key เมื่อต้องการเข้ารหัสลับแบบ Asymmetric Encryption และ Decryption โดยเมื่อทำการ Generate Key จะแสดงส่วนของการกำหนดชื่อไฟล์ของ private key และ public key รวมทั้ง ค่า Seed



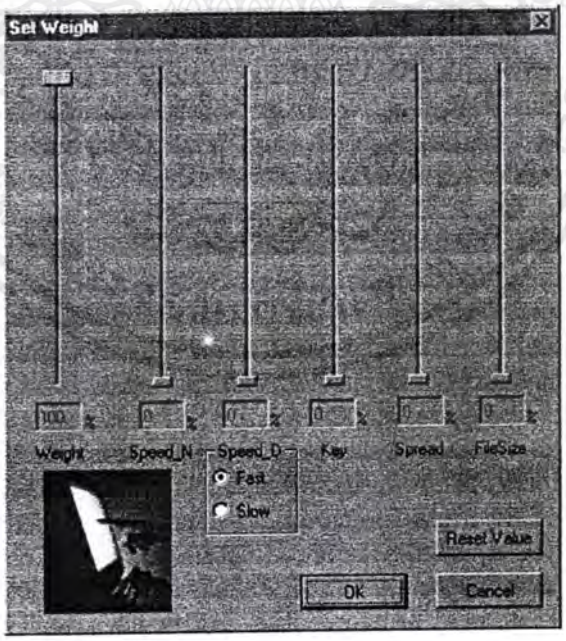
รูปที่ 4.10 แสดงหน้าจอการแสดง Public key, Private key และค่า Seed ของการเข้ารหัสแบบ Asymmetric

5 หน้าจอสำหรับส่วนการเปรียบเทียบวิธีการเข้ารหัสลับแต่ละวิธี Compare Efficiency เมื่อเลือกที่จะทำการเปรียบเทียบ โดยจะแสดงส่วนการเลือกกลุ่มของวิธีการเข้ารหัสที่ต้องการ รวมทั้งแสดงหน้าจอการรับชื่อไฟล์ที่จะนำมาใช้ในการเข้ารหัส



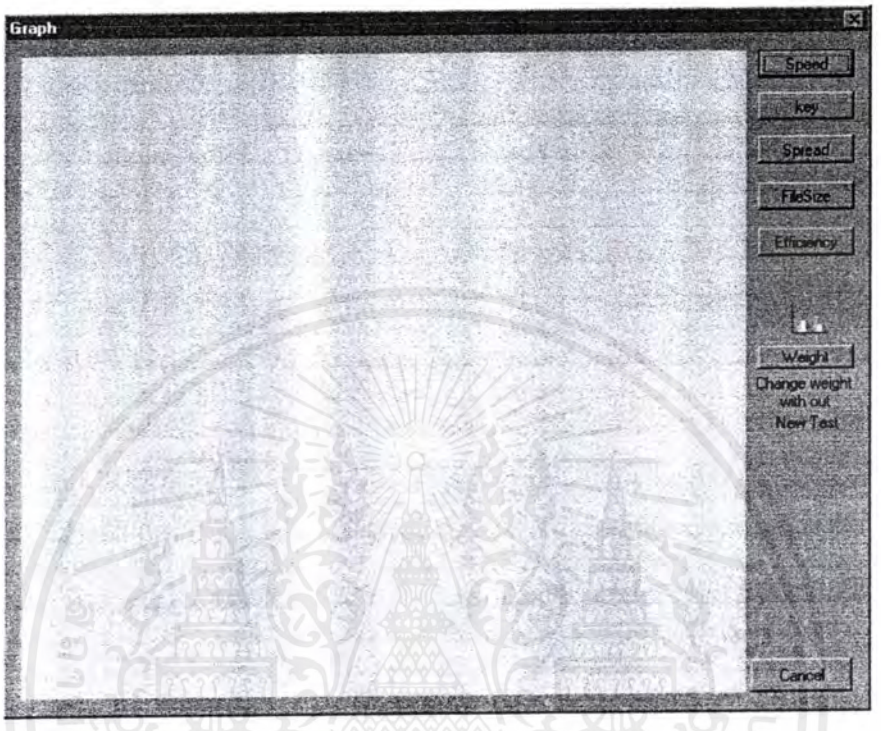
รูปที่ 4.11 แสดงส่วนหน้าจอขอการตัดสินใจเลือกวิธีการเข้ารหัสลับ

6 เมนูของส่วนที่กำหนดน้ำหนักของคุณสมบัติที่จะใช้ในการเปรียบเทียบแต่ละวิธี



รูปที่ 4.12 แสดงหน้าจอที่ใช้ในการกำหนดน้ำหนักของคุณสมบัติที่ใช้ในการเปรียบเทียบ

7 เมนู Efficiency จะเป็นส่วนของหน้าจอที่จะแสดงกราฟที่ได้จากค่าที่เกิดจากการเปรียบเทียบคุณสมบัติของการเข้ารหัสแบบต่างๆ ตามกลุ่มของการเลือกวิธีการเข้ารหัสที่ได้เลือกไว้



รูปที่ 4.13 แสดงหน้าจอของส่วนแสดงกราฟที่ได้จากการเปรียบเทียบ

บทที่ 5

ผลการทำงานของระบบ

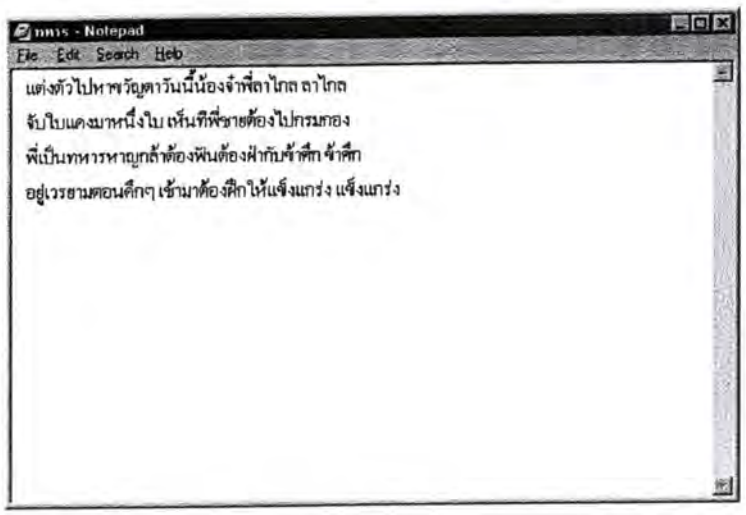
ข้อจำกัดในการใช้ระบบ

1. ข้อมูลที่จะใช้ในการเข้ารหัสแบบ Asymmetric Encryption ต้องเป็นไฟล์ชนิด ไฟล์ข้อความ (text file) เท่านั้น
2. ข้อมูลที่ใช้เข้ารหัสแบบ Symmetric Encryption จะเข้ารหัสลับเป็นได้ทุกไฟล์ข้อมูล แต่การเข้ารหัสแบบ Asymmetric Encryption จะต้องเป็นไฟล์ข้อความเท่านั้น จะเป็นภาษาอังกฤษหรือภาษาไทยก็ได้
3. ความต้องการอย่างต่ำของระบบ จะต้องมีลักษณะคือ Cpu รุ่น 486 ขึ้นไป หน่วยความจำ Ram 8 Mbyte มีเนื้อที่ว่างในฮาร์ดดิสก์ตั้งแต่ 5 Mbyte ขึ้นไป
4. ระบบที่แนะนำ Cpu Pentium 100 หน่วยความจำ Ram 16 Mbyte เนื้อที่ในฮาร์ดดิสก์ตั้งแต่ 10 Mbyte

ผลการทำงานของระบบ

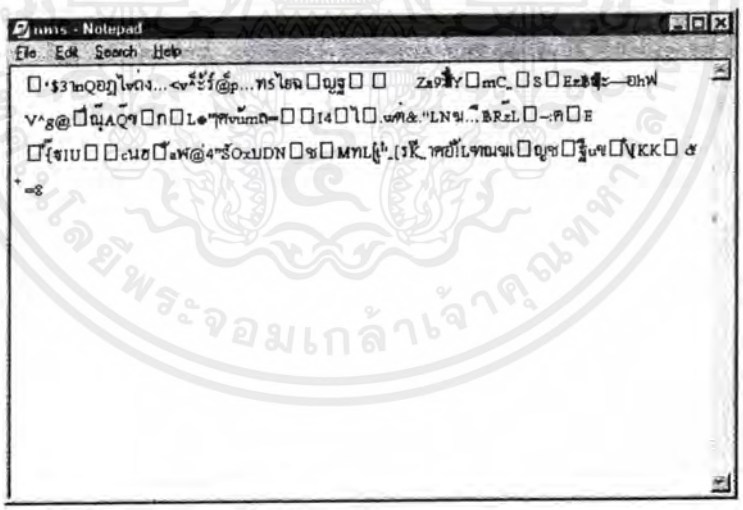
ซอฟต์แวร์ที่จัดทำขึ้นนี้ สามารถที่จะทำการเข้ารหัสลับและถอดรหัสลับได้ โดยใช้ Algorithm ของการเข้ารหัสลับแบบ Symmetric และ แบบ Asymmetric ได้ จุดประสงค์ของการพัฒนาซอฟต์แวร์นี้ คือ เพื่อที่จะสามารถใช้ในการเปรียบเทียบการเข้ารหัสลับ และถอดรหัสลับของ Algorithm แบบต่างๆได้ เพื่อที่จะเลือกวิธีการเข้ารหัสที่ดีที่สุดเพื่อให้เหมาะสมกับความต้องการที่จะใช้งานให้มากที่สุด

การทำงานของซอฟต์แวร์ เมื่อเลือกไฟล์ที่ต้องการจะทำการเข้ารหัส ซึ่งถ้าเป็นการเข้ารหัสลับแบบ Symmetric Encryption แล้ว จะสามารถทำการเลือกไฟล์ที่จะเข้ารหัสลับได้ทุกไฟล์ ไม่ว่าจะเป็นไฟล์ข้อความ รูปภาพ เสียง และอื่นๆ ซึ่งปรากฏดังรูปเป็นไฟล์ข้อความที่ต้องการเขจ้ารหัสลับ หรือเรียกว่า plaintext



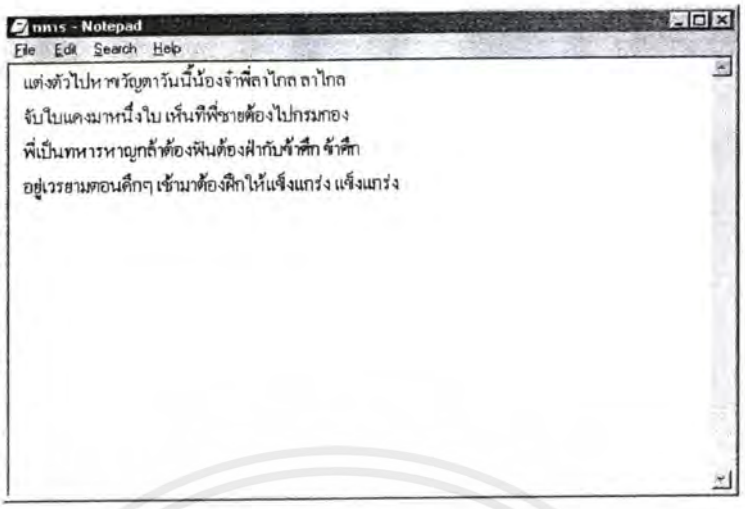
รูปที่ 5.1 แสดงข้อความในไฟล์ก่อนที่จะทำการเข้ารหัสลับ

ถ้าเป็นการเข้ารหัสลับไฟล์โดยใช้การเข้ารหัสลับแบบ Symmetric Encryption แบบ blowfish ที่ใช้รหัสลับในการเข้ารหัสขนาด 16 ไบต์ คือ 1234567890123456 แสดงดังรูป



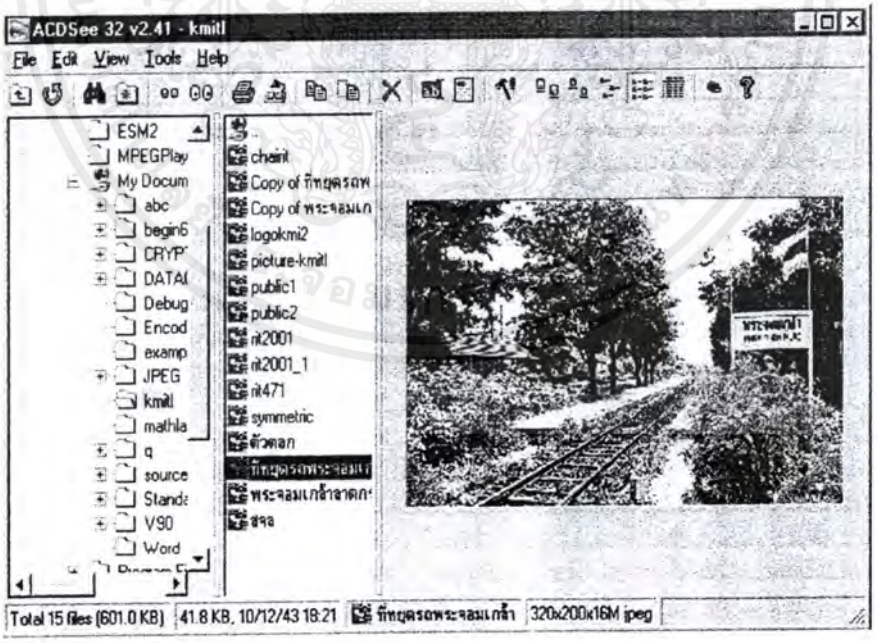
รูปที่ 5.2 แสดงข้อความในไฟล์หลังการเข้ารหัสลับโดยใช้ algorithm แบบ blowfish

และเมื่อเราทำการใช้รหัสลับเดิมที่ถูกต้องในการถอดรหัสข้อมูล เราจะได้ไฟล์ข้อความเดิมกลับมา ดังแสดงในรูป



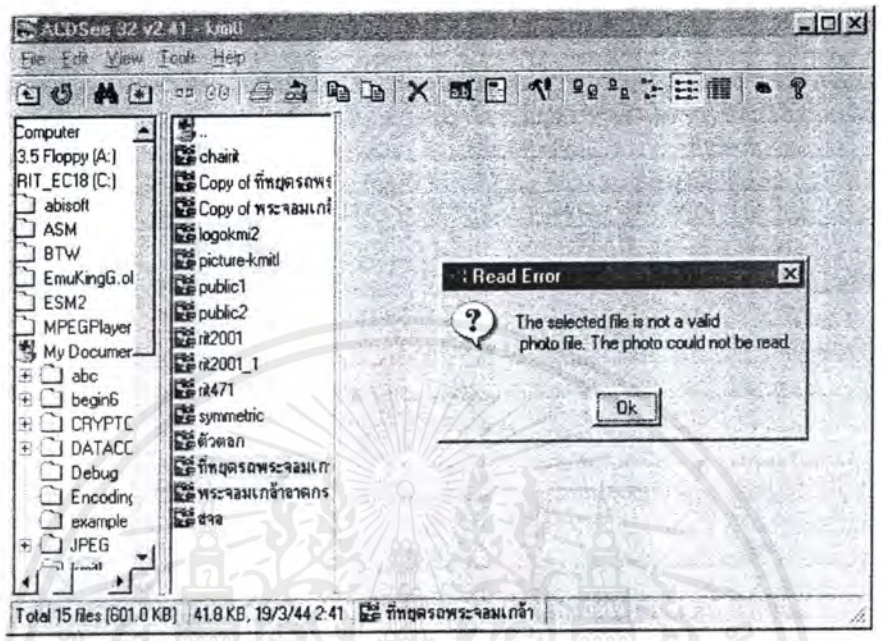
รูปที่ 5.3 แสดงข้อความในไฟล์หลังจากที่ทำการถอดรหัสแล้ว

และถ้าหากข้อมูลที่ต้องการเข้ารหัสลับ เป็นอย่างอื่นที่ไม่ใช่ ไฟล์ข้อความ หรือ plaintext ตัวอย่างเช่น ไฟล์รูปภาพ ในที่นี้จะใช้รูปภาพที่เป็น JPEG ในการเข้ารหัสลับ โดยใช้การเข้ารหัสแบบ RC2 ที่ใช้รหัสลับในการเข้ารหัสขนาด 16 ไบต์ คือ 1234567890123456 แสดงดังรูป



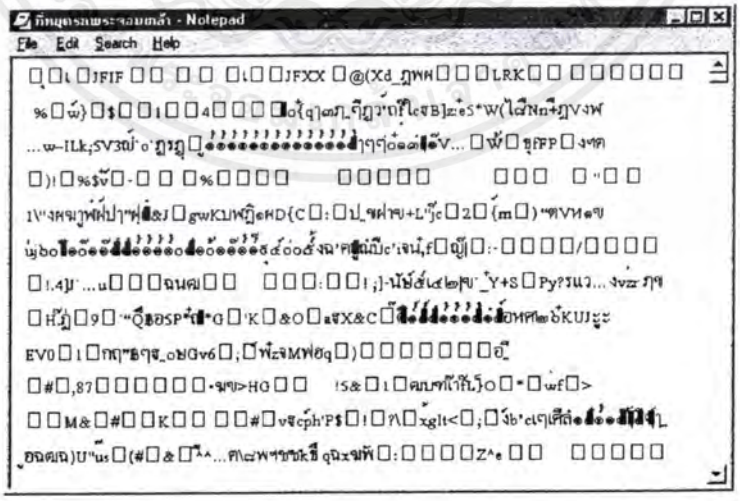
รูปที่ 5.4 แสดงหน้าจอของโปรแกรม ACDSee ที่แสดงรูปภาพ ที่หยุดรถพระจอมเกล้า. JPG ที่ต้องการจะทำการเข้ารหัสลับข้อมูล

และผลที่ได้จากการเข้ารหัสลับข้อมูลไฟล์รูปภาพ จะทำให้รูปแบบโครงสร้างของไฟล์ข้อมูลเปลี่ยนไป ทำให้ไม่สามารถที่จะอ่านข้อมูลได้

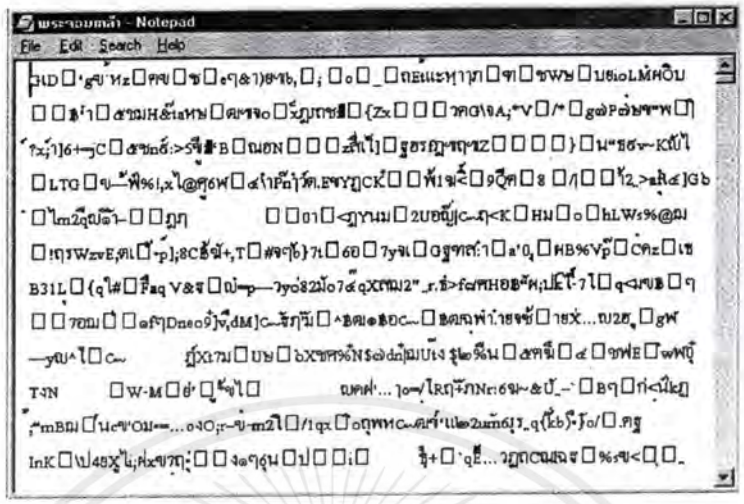


รูปที่ 5.5 แสดงหน้าจอของรูปภาพ ที่หยุดรถพระจอมเกล้า. JPG เมื่อทำการเข้ารหัสลับแล้ว

และถ้าเราอ่านข้อมูลรูปภาพนี้ด้วยโปรแกรม Notepad เพื่อแสดงให้เห็นถึงการเปลี่ยนแปลงของข้อมูลรูปภาพจาก ไฟล์ข้อมูลก่อนเข้ารหัสลับและหลังการเข้ารหัสลับ

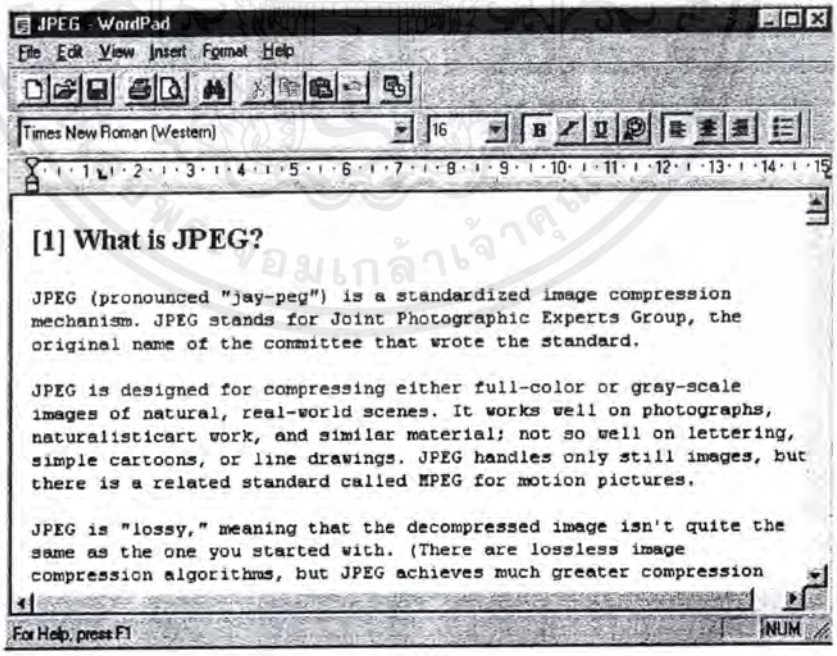


รูปที่ 5.6 แสดงหน้าจอของไฟล์ข้อความของรูป ที่หยุดรถพระจอมเกล้า. JPG ที่ต้องการจะทำการเข้ารหัสลับข้อมูล



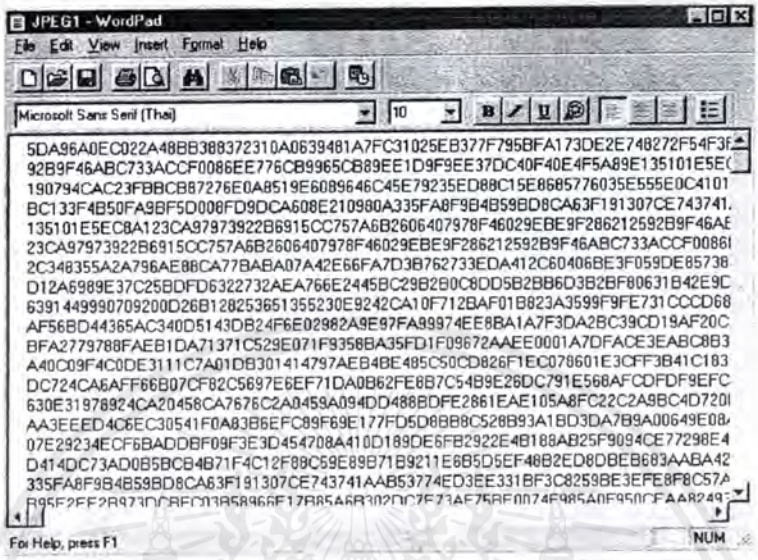
รูปที่ 5.7 แสดงหน้าจอของไฟล์ข้อความของรูป ที่หยุดรถพระจอมเกล้า. JPG ที่ทำการเข้ารหัสลับแล ข้อมูลแล้ว

และเมื่อต้องการเข้ารหัสลับแบบ Asymmetric Encryption ซึ่งสามารถเข้ารหัสลับได้แต่เฉพาะไฟล์ข้อความ หรือ plaintext เท่านั้น ซึ่งปรากฏดังรูป



รูปที่ 5.8 แสดงหน้าจอของไฟล์ข้อความ JPEG ก่อนทำการเข้ารหัสลับแบบ Asymmetric

ผลลัพธ์ที่ได้จะเป็นไฟล์ข้อความที่มีแต่ ตัวเลข และ ตัวอักษรภาษาอังกฤษ เนื่องมาจาก การคำนวณฟังก์ชันทางคณิตศาสตร์ของการเข้ารหัสลับแบบ Asymmetric

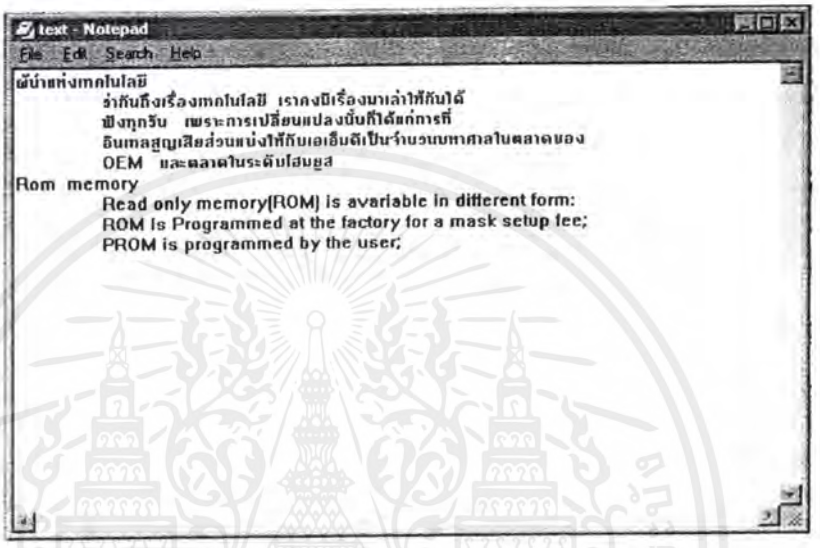


รูปที่ 5.9 แสดงหน้าจอไฟล์ข้อความ JPEG ที่ได้ทำการเข้ารหัสลับแบบ Asymmetric โดยใช้ รหัสลับคือ public key

และ เมื่อต้องการที่จะถอดรหัสลับไฟล์ข้อความกลับคืนมา เราจะต้องใช้ รหัสลับที่เรียกว่า private key ที่เป็นคู่กันกับ public key ที่ใช้ในการเข้ารหัสลับถอดกลับมา จึงจะได้ไฟล์ข้อความอ่านเดิม

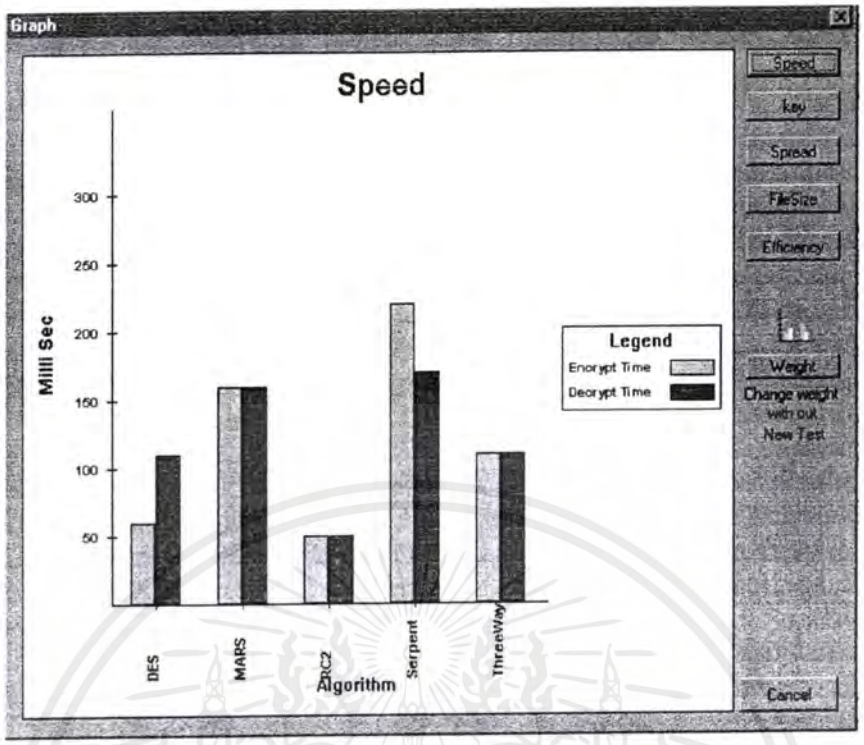
ผลการวัดประสิทธิภาพของอัลกอริทึม

วัดข้อมูลของไฟล์ข้อความเมื่อเรากำหนดน้ำหนักของปัจจัยในการตัดสินใจ ที่จะเลือกวิธีการเข้ารหัสข้อความ โดยกำหนดปัจจัยคุณสมบัติในการตัดสินใจ คือ ความเร็ว 30% การกระจายข้อมูล 30 % จำนวนรหัสลับ 20% และ ขนาดของไฟล์ 20%

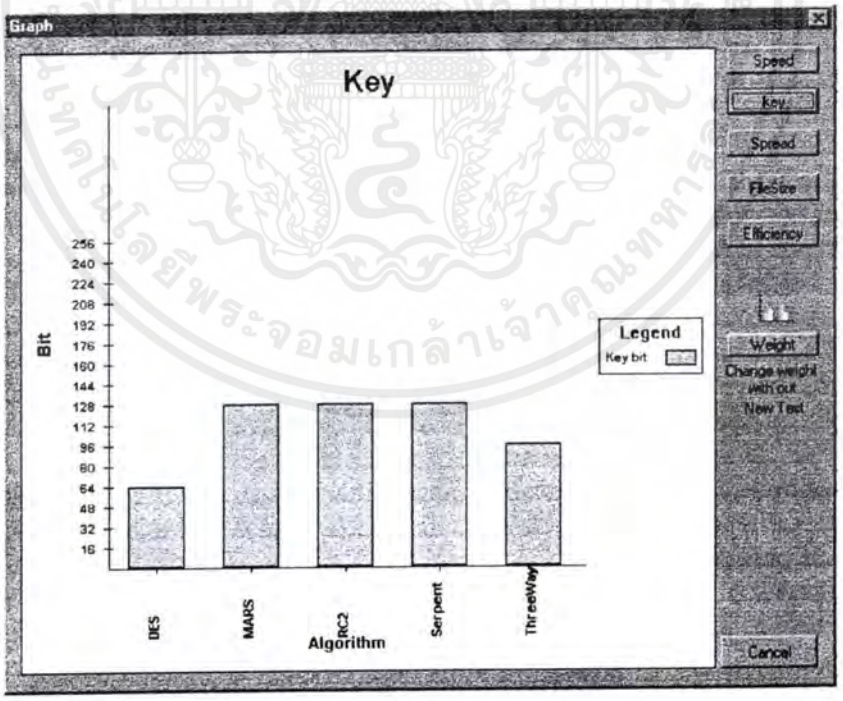


รูปที่ 5.10 แสดงไฟล์ข้อมูลที่จะทำการเปรียบเทียบวิธีการเข้ารหัสลับ

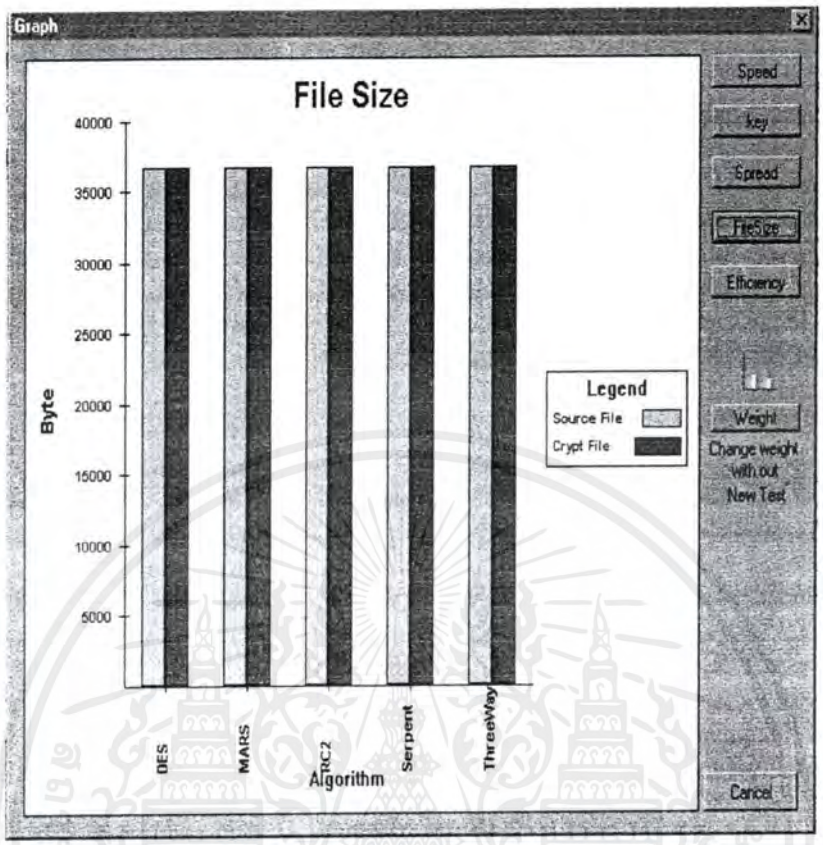
เมื่อทำการเปรียบเทียบการเข้ารหัสข้อมูล โดยเลือกวิธีการเข้ารหัสลับที่ต้องการ แล้วทำตามกระบวนการเข้ารหัสลับ แล้ว จะได้ผลดังนี้ คือ



รูปที่ 5.11 แสดงหน้าจอการเปรียบเทียบความเร็วของวิธีการเข้ารหัสลับ

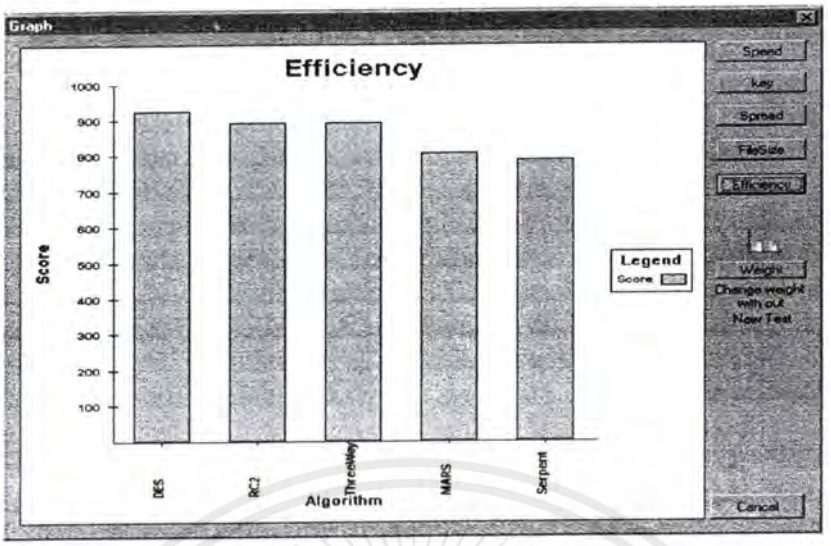


รูปที่ 5.12 แสดงหน้าจอการเปรียบเทียบจำนวนรหัสลับของวิธีการเข้ารหัสลับ



รูปที่ 5.13 แสดงหน้าจอการเปรียบเทียบขนาดของไฟล์หลังการเข้ารหัสลับแต่ละวิธี

เมื่อจบการประมวลผลการเลือกวิธีการเข้ารหัส เราสามารถทำการดูการแสดงผลกราฟของการเข้ารหัสลับแต่ละวิธีที่เราเลือกขึ้นมา เพื่อที่จะตัดสินใจว่าวิธีการเข้ารหัสลับใดเป็นวิธีการเข้ารหัสที่เหมาะสมกับความต้องการของเรามากที่สุด



รูปที่ 5.14 แสดงหน้าจอของ ผลของประสิทธิภาพรวมในการเปรียบเทียบของการตัดสินใจเลือกวิธีการเข้ารหัสลับวิธีต่างๆ

ผลการให้คะแนนปรากฏว่า วิธีการ DES ได้ 900 คะแนน RC2 ได้ 880 คะแนน Threeway ได้ 880 คะแนน MARS ได้คะแนน 800 คะแนน และ serpent ได้คะแนน 780 คะแนน ดังนั้น วิธีการเข้ารหัสที่น่าจะเหมาะสมกับความต้องการของผู้ใช้มากที่สุด คือ DES ที่ได้คะแนน 900 คะแนน ซึ่งเป็นวิธีที่มีคะแนนรวมสูงที่สุด

บทที่ 6
บทสรุปและข้อเสนอแนะ

สรุปผลการทำงาน

การทำปัญหาพิเศษนี้ มีจุดมุ่งหมายคือเพื่อจัดทำซอฟต์แวร์ที่สามารถใช้ทำการเข้ารหัสข้อมูลที่จัดเก็บในรูปแบบไฟล์ และทำการเปรียบเทียบประสิทธิภาพของการเข้ารหัส เพื่อนำข้อมูลที่ได้จากการเข้ารหัสนำไปช่วยการตัดสินใจ เลือกวิธีเข้ารหัสที่ตรงตามความต้องการของผู้ใช้มากที่สุด การเปรียบเทียบนั้น จะใช้ข้อมูลได้แก่ ระยะเวลาที่ใช้ในการเข้ารหัส ขนาดของ Key ที่ใช้ในการเข้ารหัส การกระจายของข้อมูลที่ได้ทำการเข้ารหัสแล้ว และจะนำไปประมวลผลกับ Weight ที่ผู้ใช้ได้กำหนดไว้ เพื่อหาวิธีที่ตรงกับความต้องการของผู้ใช้

ปัญหาและข้อเสนอแนะ

ในการจัดทำซอฟต์แวร์ได้พบปัญหาต่าง ๆ ดังนี้

1. ในการเข้ารหัสข้อมูลที่มีขนาดของข้อมูลไม่มากในกรณีที่ CPU มีความเร็วมาก ๆ การเข้ารหัสจะทำเสร็จได้ในเวลาสั้นมาก (เป็น millisecond) ทำให้ไม่สามารถจับเวลาในการเข้ารหัสได้อย่างแม่นยำเท่าที่ควร (ความละเอียดของเวลาไม่ต่ำกว่า 50 millisecond)

2. การเข้ารหัส – ถอดรหัส โดยใช้อัลกอริทึมแบบ Asymmetric จะสามารถกระทำกับข้อมูลที่เป็นข้อความได้เท่านั้น เพราะข้อมูลในรูปแบบอื่น (ภาพหรือเสียง) จะมีอักขระบางตัวที่ไม่สามารถนำมาทำการคำนวณเพื่อเข้ารหัสได้ปนอยู่ด้วย เมื่อทำการเข้ารหัสข้อมูลดังกล่าวจะทำให้ไม่สามารถถอดรหัสกลับมาเป็นข้อมูลเดิมได้

ถ้าต้องการให้ซอฟต์แวร์มีประสิทธิภาพมากขึ้น ควรแก้ไขดังนี้

ควรจะพัฒนาอัลกอริทึมที่ใช้ในการจับเวลาให้มีประสิทธิภาพมากขึ้น โดยให้สามารถจับเวลาได้ละเอียดมากขึ้น เพื่อแก้ปัญหาความผิดพลาดในการจับเวลาการเข้ารหัสข้อมูลที่มีขนาดเล็ก

อ้างอิง

นิรุธ อำนวยศิลป์ คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0 ฉบับเพื่อการใช้งานจริง. พิมพ์ครั้งที่ 3. กรุงเทพฯ: ซัคเซส มีเดีย, 2542

วนิดา ดิเรกสุนทร วีระวัฒน์ ฐานะประสิทธิ์ และ สมชาย ฉัฐพรรณรัฐ. "การเปรียบเทียบระบบบรหัสลับแบบ DES และ RSA " ปริญญานิพนธ์ปริญญาวิทยาศาสตร์บัณฑิต ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ,2540

สุวัฒน์ ปุณณชัยยะ, ต้น ตัณฑ์สุทธิวงศ์ และ สุพจน์ ปุณณชัยยะ. เปิดโลกของ TCP/IP และโปรโตคอลของอินเตอร์เน็ต. กรุงเทพฯ: โปรวิชั่น, 2543

Brain Beckett. Introduction to CRYPTOLOGY and PC SECURITY. London: McGraw-Hill, 1996

Randall K. Nichols. ICSA guide to CRYPTOGRAPHY. New York: McGraw-Hill,1999

Ronald L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 ™ Block Cipher. V1.1, August 20,Science,1998 Available at http://www.rsa.com/rsalabs/aes/.

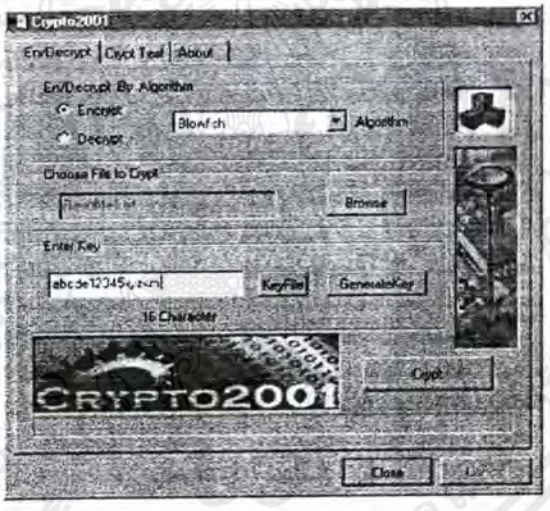
CRYPTO++ http://www.eskimo.com/~weidai/cryptlib.html

ภาคผนวก ก
คู่มือการใช้งานซอฟต์แวร์

ระบบซอฟต์แวร์เพื่อการตัดสินใจเลือกวิธีการเข้ารหัสลับ เป็นโปรแกรมที่จัดทำขึ้นเพื่อเป็นแนวทางให้ผู้ที่ต้องการจะเลือกใช้วิธีการเข้ารหัสลับ สามารถมีเหตุผลที่จะตัดสินใจเลือกวิธีการเข้ารหัสลับที่มีความเหมาะสมกับตัวผู้ใช้เองมากที่สุด

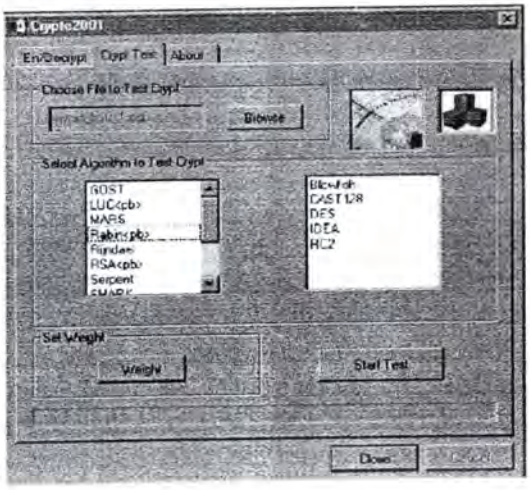
ส่วนของโปรแกรมจะประกอบไปด้วย

- ส่วนการเข้ารหัสและถอดรหัสลับ En/Decrypt เป็นหน้าจอที่ผู้ใช้เลือกใช้งานเมื่อต้องการที่จะทำการเข้ารหัสลับไฟล์ข้อมูล หรือ ทำการถอดรหัสลับไฟล์ข้อมูลที่ถูกเข้ารหัสลับแล้ว



รูปที่ ก.1 แสดงหน้าจอส่วนการเข้ารหัสและถอดรหัสลับ

- ส่วนการเปรียบเทียบประสิทธิภาพของการเข้ารหัสลับ Crypt Test เป็นหน้าจอที่ผู้ใช้เลือกใช้งานเมื่อต้องการที่จะทำการเปรียบเทียบประสิทธิภาพของการเข้ารหัสลับทุกๆวิธีที่มีความสนใจที่จะต้องการใช้งาน



รูปที่ ก.2 แสดงหน้าจอส่วนการเปรียบเทียบประสิทธิภาพการเข้ารหัสลับ

- ส่วน About เป็นหน้าจอแสดงเกี่ยวกับรายชื่อผู้พัฒนาโปรแกรม

□ <u>ส่วนการเข้ารหัสและถอดรหัส</u>

เมื่อผู้ใช้ต้องการที่จะทำการเข้ารหัสลับหรือถอดรหัสลับไฟล์ข้อมูลใดๆ ให้ทำการเปิดหน้าจอของส่วนการเข้ารหัสลับและถอดรหัสลับ

1 โดยผู้ใช้จะเลือกตัดสินใจว่าจะทำการเข้ารหัสลับข้อมูลหรือถอดรหัสลับข้อมูลโดย ทำการเลือกที่ Radio Button โดยผู้ใช้ต้องการ

รูปที่ ก.3 แสดงส่วนของการเลือกว่าจะทำการเข้ารหัสหรือถอดรหัสลับ

- ◆ Encrypt คือ การเลือกตัดสินใจว่าจะทำการเข้ารหัสลับข้อมูล
- ◆ Decrypt คือ การตัดสินใจเลือกว่าจะทำการถอดรหัสลับข้อมูล

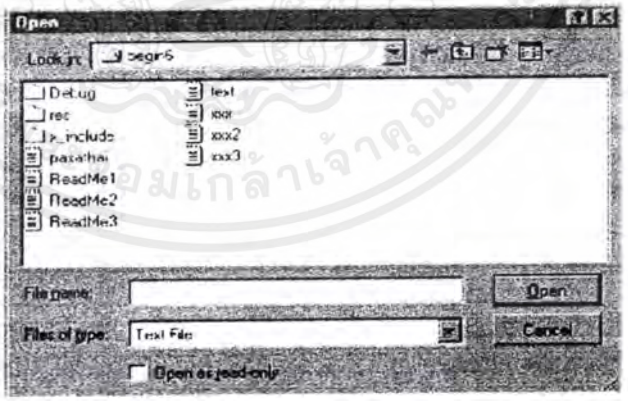
2 ทำการเลือก Algorithm ของการเข้ารหัสลับข้อมูลที่เราต้องการจะเข้ารหัสลับ หรือจะถอดรหัสลับข้อมูล

รูปที่ ก.4 แสดงส่วนการเลือกวิธีการเข้ารหัสลับ

3 จากนั้นผู้ใช้กดปุ่ม Browse เพื่อทำการเลือกไฟล์ข้อมูลที่จะเข้ารหัสลับหรือถอดรหัสลับ

รูปที่ ก. 5 แสดงปุ่มสำหรับการเลือกไฟล์ข้อมูล

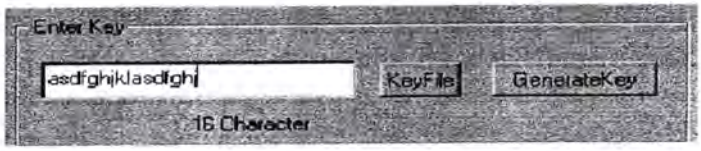
เลือกไฟล์ข้อมูล



รูปที่ ก. 6 แสดงหน้าจอสำหรับการเลือกไฟล์ข้อมูล

**<u>หมายเหตุ</u> ถ้าเป็นการเข้ารหัสลับหรือถอดรหัสลับแบบ Asymmetric สามารถที่จะเลือกไฟล์ที่จะทำการเข้ารหัสลับได้เฉพาะไฟล์ข้อความเท่านั้น ส่วนการเข้ารหัสลับแบบ Symmetric สามารถที่จะเลือกไฟล์ข้อมูลรูปแบบใดก็ได้

4 ผู้ใช้จะต้องกำหนด Key ที่ใช้ในการเข้ารหัสลับ โดย จะมี บาร์แสดงจำนวนของ Key ที่จะใช้ในการเข้ารหัสลับของการเข้ารหัสแต่ละวิธี



รูปที่ ก. 7 แสดงหน้าจอส่วนรับ Key

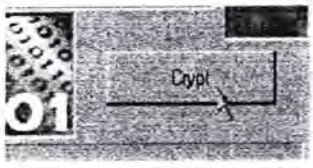
5 แต่ถ้าเป็นการเข้ารหัสลับแบบ Asymmetric ผู้ใช้จะไม่สามารถกำหนด Key เองได้ แต่จะต้องทำการเปิดไฟล์ โดยทำการคลิก Key File เพื่อสำหรับเป็น Key ในการเข้ารหัสแบบนี้

รูปที่ ก. 8 แสดงหน้าจอส่วนการสร้าง Key

ในการสร้าง Key ให้ทำการคลิกปุ่ม GenerateKey เพื่อสร้าง Key ขึ้นมาเพื่อใช้ในการเข้ารหัส จะได้ไฟล์Keyที่ใช้ในการเข้ารหัสออกมา 2 ไฟล์ คือ .pv และ .pb

**<u>หมายเหตุ</u> ถ้าเป็นการเข้ารหัสลับแบบ Asymmetric เราจะใช้ไฟล์ Key ที่เป็น Public key (file .pb) ในการเข้ารหัสลับ และใช้ไฟล์ Key ที่เป็น Private key (file .pv) ในการถอดรหัส

6 เมื่อทำการป้อนข้อมูลต่างๆสมบูรณ์แล้วก็ทำการคลิกปุ่ม Crypt เพื่อทำการประมวลผล



รูปที่ ก. 9 แสดงหน้าจอส่วนปุ่ม Crypt เพื่อทำการประมวลผล

โดยไฟล์ข้อมูลที่ทำการเข้ารหัสลับจะถูกเปลี่ยนข้อมูลไปจนไม่สามารถอ่านได้ ดังนั้นถ้าหากต้องการอ่านข้อมูลก็ให้ทำการถอดรหัสข้อมูลโดยให้ Key ในการถอดรหัส จะได้ข้อมูลเดิมกลับมา

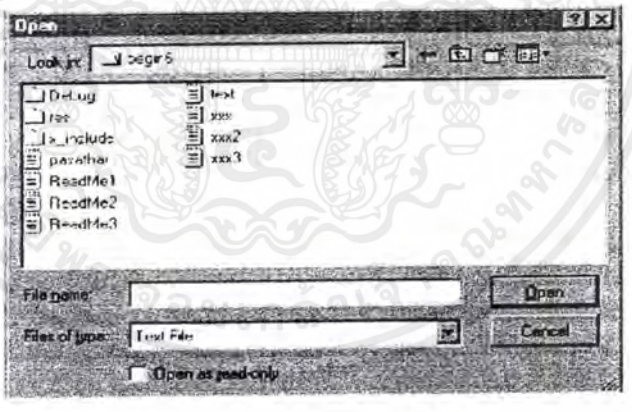
- **<u>ส่วนของการเปรียบเทียบประสิทธิภาพของวิธีการเข้ารหัสข้อมูล</u>**

เมื่อผู้ใช้ต้องการที่จะเปรียบเทียบว่าวิธีการเข้ารหัสลับวิธีในจะมีคุณสมบัติเหมาะสมกับความต้องการใช้งานของเรา ก็ทำการเลือกที่หน้าจอของส่วนการเปรียบเทียบสัมประสิทธิ์

1 อย่างแรกผู้ใช้ต้องทำการเลือกไฟล์ข้อมูลตัวอย่างที่จะใช้ในการเปรียบเทียบวิธีการเข้ารหัสต่างๆ โดยคลิกที่ปุ่ม Browse ที่กรอบของการเลือกไฟล์

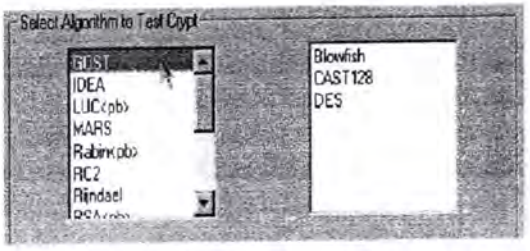
รูปที่ ก.10 แสดงหน้าจอส่วนการเลือกไฟล์ข้อมูล

แล้วทำการเลือกไฟล์ข้อมูลที่ต้องการ



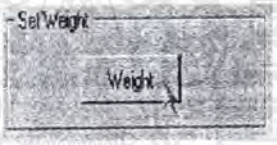
รูปที่ ก. 11 แสดงหน้าจอสำหรับการเลือกไฟล์ข้อมูล

2 จากนั้นทำการเลือกวิธีการเข้ารหัสลับที่มีอยู่จาก List box ทางซ้ายมือมาใส่ทางขวา ตามแต่ความต้องการของผู้ใช้ แต่ต้องไม่เกิน 9 วิธี



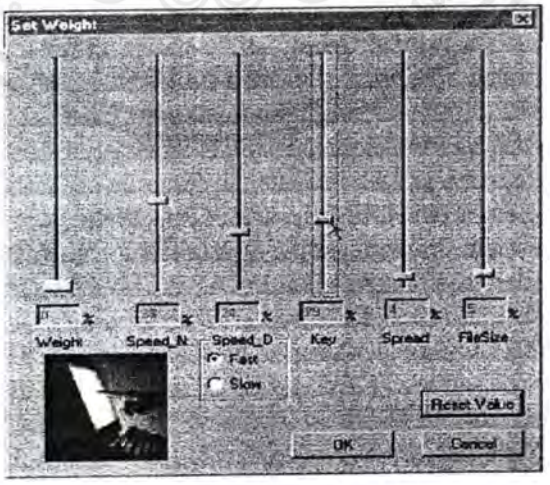
รูปที่ ก.12 แสดงหน้าจอส่วนของวิธีการเลือกการเข้ารหัสลับ

3 คลิกปุ่ม Set Weight เพื่อทำการเลือกให้น้ำหนักว่าต้องการที่จะกำหนดน้ำหนักของปัจจัยในเปรียบเทียบการเข้ารหัสแต่ละอย่างเป็นเท่าไรแล้วแต่ความต้องการของผู้ใช้เอง



รูปที่ ก.13 แสดงปุ่มการกำหนดน้ำหนักปัจจัยในการเข้ารหัส

ทำการเลือกให้น้ำหนัก

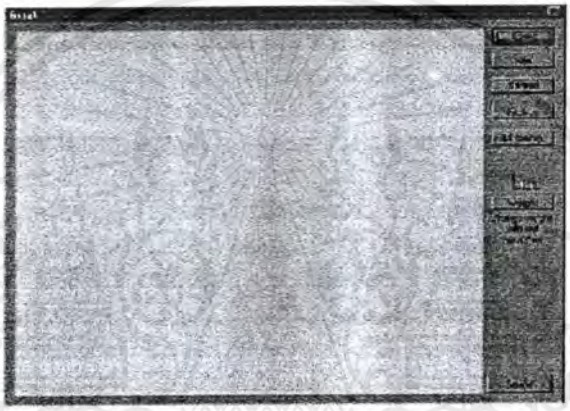


รูปที่ ก.14 แสดงหน้าจอการกำหนดน้ำหนักให้ปัจจัยต่างๆ

4 คลิกปุ่ม Test เพื่อทำการทำสอบการเข้ารหัสลับแต่ละวิธีว่ามีคุณสมบัติอย่างไร

รูปที่ ก. 15 แสดงปุ่ม Test

แล้วทำการ คลิก เพื่อดูกราฟของคุณสมบัติของแต่ละวิธีได้



รูปที่ ก. 16 แสดงหน้าจอส่วนแสดงผลของกราฟ

- โดยสามารถเลือกคลิก Speed เพื่อทำการดูผลการแสดงหน้าจอในการเปรียบเทียบความเร็วในการเข้ารหัสลับ
- เลือกคลิก Key เพื่อดูผลการแสดงหน้าจอในการเปรียบเทียบจำนวนรหัสลับที่ใช้ในการเข้ารหัสลับ
- เลือกคลิก Spread เพื่อดูผลการแสดงหน้าจอในการเปรียบเทียบเปอร์เซ็นต์การกระจายข้อมูลที่ผิดพลาดของการเข้ารหัสลับแต่ละวิธี
- เลือกคลิก File Size เพื่อดูผลการแสดงหน้าจอในการเปรียบเทียบขนาดของไฟล์ข้อมูลหลังการเข้ารหัสลับ
- เลือกคลิก Efficiency เพื่อดูผลการแสดงหน้าจอของประสิทธิภาพรวมของการเข้ารหัสลับแต่ ละวิธี

```
// begin6.h : main header file for the BEGIN6 application
//

#if !defined(AFX_BEGIN6_H__C76B8123_65C2_4404_A7ED_2B3A5FF4C0EE__INCLUDED_)
#define AFX_BEGIN6_H__C76B8123_65C2_4404_A7ED_2B3A5FF4C0EE__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CBegin6App:
// See begin6.cpp for the implementation of this class
//

class CBegin6App : public CWinApp
{
public:
	CBegin6App();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CBegin6App)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CBegin6App)
		// NOTE - the ClassWizard will add and remove member functions
here.
		//    DO NOT EDIT what you see in these blocks of generated code
!
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
);


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_BEGIN6_H__C76B8123_65C2_4404_A7ED_2B3A5FF4C0EE__INCLUDED_)
```

```
// begin6.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "begin6.h"
#include "begin6Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBegin6App


BEGIN_MESSAGE_MAP(CBegin6App, CWinApp)
	//{{AFX_MSG_MAP(CBegin6App)
		// NOTE - the ClassWizard will add and remove mapping macros
here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
//	ON_COMMAND(ID_HELP, CWinApp::OnHelp) //Disable Help Button
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBegin6App construction

CBegin6App::CBegin6App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CBegin6App object

CBegin6App theApp;

/////////////////////////////////////////////////////////////////////////////
// CBegin6App initialization

BOOL CBegin6App::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a
shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CAllControlsSheet   allcontrolssheet(_T("Crypto2001"));
	m_pMainWnd = &allcontrolssheet;
	allcontrolssheet.DoModal();

	return FALSE;
}
```

```
#if !defined(AFX_GENKEYDLG_H__0E3FAFE7_5D60_4C38_BDA1_3A959C9F2B57__INCLUDED_)
#define AFX_GENKEYDLG_H__0E3FAFE7_5D60_4C38_BDA1_3A959C9F2B57__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// GenkeyDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// GenkeyDlg dialog

class GenkeyDlg : public CDialog
{
// Construction
public:
	GenkeyDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(GenkeyDlg)
	enum { IDD = IDD_DIALOG_GEN };
	int		m_algor;
	CString	m_seed;
	CString	m_filename;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(GenkeyDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(GenkeyDlg)
	virtual void OnOK();
	afx_msg void GenerateRSAKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed);
	afx_msg void GenerateLUCKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed);
	afx_msg void GenerateRabinKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_GENKEYDLG_H__0E3FAFE7_5D60_4C38_BDA1_3A959C9F2B57__INCLUDED_)
```

```
// GenkeyDlg.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "GenkeyDlg.h"

///////////////spacial include////////////////////
#include "x_include\files.h"

#include "x_include\hex.h"

#include "x_include\rsa.h"
#include "x_include\luc.h"
#include "x_include\rabin.h"
#include "x_include\randpool.h"

#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <memory>
#include <exception>
#include <list>

/*
#if (_MSC_VER >= 1000)
#include <crtdbg.h>		// for the debug heap
#endif

#if defined(__MWERKS__) && defined(macintosh)
#include <console.h>
#endif
*/

//#include "Afxdisp.h"

USING_NAMESPACE(CryptoPP)
USING_NAMESPACE(std)
///////////////spacial include////////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// GenkeyDlg dialog


GenkeyDlg::GenkeyDlg(CWnd* pParent /*=NULL*/)
	: CDialog(GenkeyDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(GenkeyDlg)
//	m_algor = -1;
	m_seed = _T("");
	m_filename = _T("");
	//}}AFX_DATA_INIT
}


void GenkeyDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(GenkeyDlg)
```

```
        DDX_CBIndex(pDX, IDC_COMBO_ALGOR, m_algor);
        DDX_Text(pDX, IDC_EDIT_SEED, m_seed);
        DDV_MaxChars(pDX, m_seed, 8);
        DDX_Text(pDX, IDC_EDIT_FILENAME, m_filename);
        DDV_MaxChars(pDX, m_filename, 5);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(GenkeyDlg, CDialog)
        //{{AFX_MSG_MAP(GenkeyDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// GenkeyDlg message handlers

void GenkeyDlg::OnOK()
{
        CString         pv_name;
        CString         pb_name;

         UpdateData(TRUE);

 if((m_filename.IsEmpty()!=TRUE) && (m_seed.IsEmpty()!=TRUE))
 {
        pv_name = m_filename;
        pb_name = m_filename;

        switch(m_algor)
        {
        case 0 :{
                                pv_name.Insert(555, "_pv.rsa");
                                pb_name.Insert(555, "_pb.rsa");
                                GenerateRSAKey(512, pv_name, pb_name, m_seed);
                        } break;


        case 1 :{
                                pv_name.Insert(555, "_pv.luc");
                                pb_name.Insert(555, "_pb.luc");
                                GenerateLUCKey(512, pv_name, pb_name, m_seed);
                        } break;

        case 2 :{
                                pv_name.Insert(555, "_pv.rbn");
                                pb_name.Insert(555, "_pb.rbn");
                                GenerateRabinKey(512, pv_name, pb_name, m_seed);
                        } break;
        }

        CDialog::OnOK();
 }

}

//////////////Generate Key Algorithm/////////////////

void GenkeyDlg::GenerateRSAKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed)
{
        RandomPool randPool;
        randPool.Put((byte *)seed, strlen(seed));
```

```
        RSAES_OAEP_SHA_Decryptor priv(randPool, keyLength);
        HexEncoder privFile(new FileSink(privFilename));
        priv.DEREncode(privFile);
        privFile.Close();

        RSAES_OAEP_SHA_Encryptor pub(priv);
        HexEncoder pubFile(new FileSink(pubFilename));
        pub.DEREncode(pubFile);
        pubFile.Close();
}

void GenkeyDlg::GenerateLUCKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed)
{
        RandomPool randPool;
        randPool.Put((byte *)seed, strlen(seed));

        LUCES_OAEP_SHA_Decryptor priv(randPool, keyLength);
        HexEncoder privFile(new FileSink(privFilename));
        priv.DEREncode(privFile);
        privFile.Close();

        LUCES_OAEP_SHA_Encryptor pub(priv);
        HexEncoder pubFile(new FileSink(pubFilename));
        pub.DEREncode(pubFile);
        pubFile.Close();
}

void GenkeyDlg::GenerateRabinKey(unsigned int keyLength, const char
*privFilename, const char *pubFilename, const char *seed)
{
        RandomPool randPool;
        randPool.Put((byte *)seed, strlen(seed));

        RabinDecryptor priv(randPool, keyLength);
        HexEncoder privFile(new FileSink(privFilename));
        priv.DEREncode(privFile);
        privFile.Close();

        RabinEncryptor pub(priv);
        HexEncoder pubFile(new FileSink(pubFilename));
        pub.DEREncode(pubFile);
        pubFile.Close();
}

///////////////Generate Key Algorithm/////////////////
```

```
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1998 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

/////////////////////////////////////////////////////////////////////////////
// Auxiliary System/Screen metrics

struct GLOBAL_DATA
{

	////////DATA////////
	int    n_count;
	CString n_buffer[9];

	double n_diff_sec_en[9];   //speed
	double n_diff_sec_de[9];

	int n_key[9];                    //key

	int n_spread_0[9];               //spread
	int n_spread_1[9];

	long n_filesize_source;          //file size
	long n_filesize_destination[9];
	////////DATA////////

	////////WEIGHT////////
	int w_weight;
	int w_speed_n;
	int w_speed_d;
	int w_key;
	int w_spread;
	int w_filesize;

	int c_fast;
	////////WEIGHT////////

	////////PASS TO REPORT////////
	int isort[9];
	int effi[9];
	////////PASS TO REPORT////////

// Implementation
	GLOBAL_DATA();
};

extern GLOBAL_DATA globalData;

/////////////////////////////////////////////////////////////////////////////
```

```
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1998 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#include "stdafx.h"
#include "globals.h"

/////////////////////////////////////////////////////////////////////////////
// Cached system metrics, etc

GLOBAL_DATA globalData;

// Initialization code
GLOBAL_DATA::GLOBAL_DATA()
{
	n_count = 0;

	w_weight   = 100;
	w_speed_n = 0;
	w_speed_d = 0;
	w_key     = 0;
	w_spread   = 0;
	w_filesize= 0;

	c_fast = 1;


}

/////////////////////////////////////////////////////////////////////////////
```

```
#if
!defined(AFX_GRAPHTESTDLG_H__9E0F3D47_C1F1_4D5A_8EB4_DD1EEC1C197E__INCLUDED_)
#define AFX_GRAPHTESTDLG_H__9E0F3D47_C1F1_4D5A_8EB4_DD1EEC1C197E__INCLUDED_


#include "Graph.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// GraphtestDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// GraphtestDlg dialog

class GraphtestDlg : public CDialog
{
// Construction
public:
	GraphtestDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(GraphtestDlg)
	enum { IDD = IDD_DIALOG_GRAPH };
		// NOTE: the ClassWizard will add data members here

	//}}AFX_DATA

     CGraph* testGraph;
     BOOL graphComplete;

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(GraphtestDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(GraphtestDlg)
	afx_msg void OnButtonSpeed();
	afx_msg void OnButtonKey();
	afx_msg void OnButtonSpread();
	afx_msg void OnButtonFilesize();
	afx_msg void OnButtonEfficiency();
	afx_msg void OnButtonWeight();
	afx_msg void OnButtonReport();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_GRAPHTESTDLG_H__9E0F3D47_C1F1_4D5A_8EB4_DD1EEC1C197E__INCLUDED_)
```

```
// GraphtestDlg.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "GraphtestDlg.h"

#include "Graph.h"
#include "GraphSeries.h"

#include "globals.h"

#include "WeightDlg.h"                    //include for New Dialog
#include "ReportDlg.h"                    //include for New Dialog

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// GraphtestDlg dialog


GraphtestDlg::GraphtestDlg(CWnd* pParent /*=NULL*/)
	: CDialog(GraphtestDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(GraphtestDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void GraphtestDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(GraphtestDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(GraphtestDlg, CDialog)
	//{{AFX_MSG_MAP(GraphtestDlg)
	ON_BN_CLICKED(IDC_BUTTON_SPEED, OnButtonSpeed)
	ON_BN_CLICKED(IDC_BUTTON_KEY, OnButtonKey)
	ON_BN_CLICKED(IDC_BUTTON_SPREAD, OnButtonSpread)
	ON_BN_CLICKED(IDC_BUTTON_FILESIZE, OnButtonFilesize)
	ON_BN_CLICKED(IDC_BUTTON_EFFICIENCY, OnButtonEfficiency)
	ON_BN_CLICKED(IDC_BUTTON_WEIGHT, OnButtonWeight)
	ON_BN_CLICKED(IDC_BUTTON_REPORT, OnButtonReport)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// GraphtestDlg message handlers

void GraphtestDlg::OnButtonSpeed()
{
	// TODO: Add your control notification handler code here
	int n;
	double max_value;                                    //*
	int set_space;
	int set_range;
```

```
        CGraphSeries* series[9];
        testGraph = new CGraph();
        testGraph->SetGraphTitle("Speed");              //*

        //cal range
        max_value = 0;                                                  //*
        for(n=0; n<globalData.n_count; n++)
        {
              if(max_value < globalData.n_diff_sec_en[n])               //*
              {  max_value = globalData.n_diff_sec_en[n];}   //*
              if(max_value < globalData.n_diff_sec_de[n])               //*
              {  max_value = globalData.n_diff_sec_de[n];}   //*
        }
              max_value = max_value * 1000;
        /////////Cal Scale////////
        long range, scale, expo;
              range = 10;
              scale = 1;
              expo  = 1;

              while(range < max_value)
              {             scale = 1;
                            expo  = expo * 10;
                     while((range < max_value) && (scale<=10))
                     {
                            range = scale * expo;
                            scale++;
                     }
              }
                            scale--;

                            while(scale < 6){ scale = scale *2; }
                            set_space = range/scale;
                            set_range = range;
        /////////Cal Scale////////

        //cal range
        testGraph->SetTickSpace(set_space);                             //*
        testGraph->SetTickRange(set_range);                             //*

        testGraph->SetXAxisAlignment(90);
        testGraph->SetXAxisLabel("Algorithm");
        testGraph->SetYAxisLabel("Milli Sec");          //*
        testGraph->SetLegend(0, "Encrypt Time");        //*
        testGraph->SetLegend(1, "Decrypt Time");        //*

        //set up some series
        for(n=0; n<globalData.n_count; n++)
        {
          series[n] = new CGraphSeries();
          series[n]->SetLabel(globalData.n_buffer[n]);
          series[n]->SetData(0, (int)(globalData.n_diff_sec_en[n] * 1000));
        //*
          series[n]->SetData(1, (int)(globalData.n_diff_sec_de[n] * 1000));
        //*
        testGraph->AddSeries(series[n]);
        }


        CWnd* graphFrame = (CWnd*)GetDlgItem(IDC_GRAPH_FRAME);
        CDC* pDC = graphFrame->GetDC();
        testGraph->DrawGraph(pDC);
        ReleaseDC(pDC);
}
```

```
void GraphtestDlg::OnButtonKey()
{
	// TODO: Add your control notification handler code here
	int n;
	int max_value;                                          //*
	int set_space;
	int set_range;
	CGraphSeries* series[9];
	testGraph = new CGraph();
	testGraph->SetGraphTitle("Key");          //*

	//cal range
	max_value = 0;                                                  //*
	for(n=0; n<globalData.n_count; n++)
	{
		if(max_value < globalData.n_key[n])             //*
		{  max_value = globalData.n_key[n];}     //*
	}

	set_space= 256/16;  //*
	set_range= 256;     //*

	if(max_value >= 360)
	{             set_space= 650/10;  //*
	              set_range= 650;     //*
	}

	//cal range
	testGraph->SetTickSpace(set_space);                     //*
	testGraph->SetTickRange(set_range);                     //*

	testGraph->SetXAxisAlignment(90);
	testGraph->SetXAxisLabel("Algorithm");
	testGraph->SetYAxisLabel("Bit");          //*
	testGraph->SetLegend(0, "Key bit");       //*
	if(max_value >= 360)
	{testGraph->SetYAxisLabel("Byte");
	 testGraph->SetLegend(0, "Key Byte");
	}



	//set up some series
	for(n=0; n<globalData.n_count; n++)
	{
	  series[n] = new CGraphSeries();
	  series[n]->SetLabel(globalData.n_buffer[n]);
	  series[n]->SetData(0, globalData.n_key[n]*8);         //*
	testGraph->AddSeries(series[n]);
	}


	CWnd* graphFrame = (CWnd*)GetDlgItem(IDC_GRAPH_FRAME);
	CDC* pDC = graphFrame->GetDC();
	testGraph->DrawGraph(pDC);
	ReleaseDC(pDC);
}

void GraphtestDlg::OnButtonSpread()
{
	// TODO: Add your control notification handler code here
	int n;
	int max_value;                                          //*
	int set_space;
	int set_range;
```

```
        CGraphSeries* series[9];
        testGraph = new CGraph();
        testGraph->SetGraphTitle("Spread");            //*

        //cal range
        max_value = 0;                                                  //*
        for(n=0; n<globalData.n_count; n++)
        {
              if(max_value < globalData.n_spread_0[n])          //*
              {  max_value = globalData.n_spread_0[n];}         //*
              if(max_value < globalData.n_spread_1[n])          //*
              {  max_value = globalData.n_spread_1[n];}         //*
        }


        /////////Cal Scale////////
        long range, scale, expo;
              range = 10;
              scale = 1;
              expo  = 1;

              while(range < max_value)
              {             scale = 1;
                            expo  = expo * 10;
                    while((range < max_value) && (scale<=10))
                    {
                            range = scale * expo;
                            scale++;
                    }
              }
                            scale--;

                            while(scale < 6){ scale = scale *2; }
                            set_space = range/scale;
                            set_range = range;
        /////////Cal Scale////////

        //cal range
        testGraph->SetTickSpace(set_space);                             //*
        testGraph->SetTickRange(set_range);                             //*

        testGraph->SetXAxisAlignment(90);
        testGraph->SetXAxisLabel("Algorithm");
        testGraph->SetYAxisLabel("Spread Ratio(x0.01%)");       //*
        testGraph->SetLegend(0, "Spread of Bit 0");             //*
        testGraph->SetLegend(1, "Spread of Bit 1");             //*

        //set up some series
        for(n=0; n<globalData.n_count; n++)
        {
          series[n] = new CGraphSeries();
          series[n]->SetLabel(globalData.n_buffer[n]);
          series[n]->SetData(0, globalData.n_spread_0[n]);      //*
          series[n]->SetData(1, globalData.n_spread_1[n]);      //*
      testGraph->AddSeries(series[n]);
        }


        CWnd* graphFrame = (CWnd*)GetDlgItem(IDC_GRAPH_FRAME);
        CDC* pDC = graphFrame->GetDC();
        testGraph->DrawGraph(pDC);
        ReleaseDC(pDC);
}

void GraphtestDlg::OnButtonFilesize()
```

```
{
	// TODO: Add your control notification handler code here
	int n;
	long max_value;                                          //*
	int set_space;
	int set_range;
	CGraphSeries* series[9];
	testGraph = new CGraph();
	testGraph->SetGraphTitle("File Size");          //*

	//cal range
	max_value = 0;                                                  //*
	for(n=0; n<globalData.n_count; n++)
	{
		if(max_value < globalData.n_filesize_source)            //*
		{  max_value = globalData.n_filesize_source;}  //*
		if(max_value < globalData.n_filesize_destination[n])          //*
		{  max_value = globalData.n_filesize_destination[n];} //*
	}

	/////////Cal Scale///////
	long range, scale, expo;
		range = 10;
		scale = 1;
		expo  = 1;

		while(range < max_value)
		{               scale = 1;
				expo  = expo * 10;
			while((range < max_value) && (scale<=10))
			{
				range = scale * expo;
				scale++;
			}
		}
				scale--;

				while(scale < 6){ scale = scale *2; }
				set_space = range/scale;
				set_range = range;
	/////////Cal Scale///////

	//cal range
	testGraph->SetTickSpace(set_space);                     //*
	testGraph->SetTickRange(set_range);                     //*

	testGraph->SetXAxisAlignment(90);
	testGraph->SetXAxisLabel("Algorithm");
	testGraph->SetYAxisLabel("Byte");       //*
	testGraph->SetLegend(0, "Source File"); //*
	testGraph->SetLegend(1, "Crypt File");  //*

	//set up some series
	for(n=0; n<globalData.n_count; n++)
	{
	  series[n] = new CGraphSeries();
	  series[n]->SetLabel(globalData.n_buffer[n]);
	  series[n]->SetData(0, globalData.n_filesize_source);      //*
	  series[n]->SetData(1, globalData.n_filesize_destination[n]);    //*
	testGraph->AddSeries(series[n]);
	}


	CWnd* graphFrame = (CWnd*)GetDlgItem(IDC_GRAPH_FRAME);
	CDC* pDC = graphFrame->GetDC();
```

```
        testGraph->DrawGraph(pDC);
        ReleaseDC(pDC);
}

void GraphtestDlg::OnButtonEfficiency()
{
        int n, m;
        int score;
        int index[9];

        double cp_speed_n[9];
        double cp_speed_d[9];
        int    cp_key[9];
        int    cp_spread_0[9];
        int    cp_spread_1[9];
        long   cp_filesize[9];

        double buf_double;
        int    buf_int;
        long   buf_long;
        int    buf_index;

        int sc_speed_n[9];
        int sc_speed_d[9];
        int sc_key[9];
        int sc_spread_0[9];
        int sc_spread_1[9];
        int sc_filesize[9];
        int sc_total[9];

        for(n=0; n<=9; n++)          //init
        {
      index[n]        = 0;

         cp_speed_n[n]   = 0;
         cp_speed_d[n]   = 0;
         cp_key[n]       = 0;
         cp_spread_0[n]  = 0;
         cp_spread_1[n]  = 0;
         cp_filesize[n]  = 0;
         sc_speed_n[n]   = 0;
         sc_speed_d[n]   = 0;
         sc_key[n]       = 0;
         sc_spread_0[n]  = 0;
         sc_spread_1[n]  = 0;
         sc_filesize[n]  = 0;
         sc_total[n]     = 0;
        }

        ////////////Give Score (Speed_En)///////////
        score = 10;
        for(n=0; n<globalData.n_count; n++)              //copy data
        {       index[n] = n;
                cp_speed_n[n] = globalData.n_diff_sec_en[n];   //*
        }

        for(n=0; n<(globalData.n_count-1); n++)          //sort data
        {
       for(m=0; m<(globalData.n_count-1); m++)
          {
                if(cp_speed_n[m] > cp_speed_n[m+1])    //* min->max
                {
                        buf_double      = cp_speed_n[m];              //*
                        cp_speed_n[m]   = cp_speed_n[m+1];    //*
                        cp_speed_n[m+1] = buf_double;                 //*
```

```
                    buf_index  = index[m];
                    index[m]   = index[m+1];
                    index[m+1] = buf_index;
                }
            }
        }
        for(n=0; n<globalData.n_count; n++)            //give score
        {
            sc_speed_n[index[n]] = score;                              //*
            if(cp_speed_n[n] != cp_speed_n[n+1])       //*
            {
                score--;
            }
        }
        ///////////Give Score (Speed_En)///////////

        ///////////Give Score (Speed_De)///////////
        score = 10;
        for(n=0; n<globalData.n_count; n++)            //copy data
        {   index[n] = n;
            cp_speed_d[n] = globalData.n_diff_sec_de[n];   //*
        }

    if(globalData.c_fast==1)
    {
        for(n=0; n<(globalData.n_count-1); n++)        //sort data
        {
        for(m=0; m<(globalData.n_count-1); m++)
            {
                if(cp_speed_d[m] > cp_speed_d[m+1])    //* min->max
                {
                    buf_double     = cp_speed_d[m];            //*
                    cp_speed_d[m]  = cp_speed_d[m+1];    //*
                    cp_speed_d[m+1] = buf_double;               //*

                    buf_index  = index[m];
                    index[m]   = index[m+1];
                    index[m+1] = buf_index;
                }
            }
        }
    }
      if(globalData.c_fast==1)      //want fast
    {
        for(n=0; n<(globalData.n_count-1); n++)        //sort data
        {
        for(m=0; m<(globalData.n_count-1); m++)
            {
                if(cp_speed_d[m] > cp_speed_d[m+1])    //* min->max
                {
                    buf_double     = cp_speed_d[m];            //*
                    cp_speed_d[m]  = cp_speed_d[m+1];    //*
                    cp_speed_d[m+1] = buf_double;               //*

                    buf_index  = index[m];
                    index[m]   = index[m+1];
                    index[m+1] = buf_index;
                }
            }
        }
    }
      if(globalData.c_fast==0)      //want slow
    {
        for(n=0; n<(globalData.n_count-1); n++)        //sort data
```

```
    {
    for(m=0; m<(globalData.n_count-1); m++)
      {
            if(cp_speed_d[m] < cp_speed_d[m+1])   //* max->min
            {
                  buf_double      = cp_speed_d[m];              //*
                  cp_speed_d[m]   = cp_speed_d[m+1];    //*
                  cp_speed_d[m+1] = buf_double;                 //*

                  buf_index  = index[m];
                  index[m]   = index[m+1];
                  index[m+1] = buf_index;
            }
      }
    }
}

    for(n=0; n<globalData.n_count; n++)           //give score
    {
          sc_speed_d[index[n]] = score;                         //*
          if(cp_speed_d[n] != cp_speed_d[n+1])        //*
          {
                score--;
          }
    }
    ////////////Give Score (Speed_De)////////////

    ////////////Give Score (Key)////////////
    score = 10;
    for(n=0; n<globalData.n_count; n++)           //copy data
    {     index[n] = n;
          cp_key[n] = globalData.n_key[n]; //*
    }

    for(n=0; n<(globalData.n_count-1); n++)       //sort data
    {
    for(m=0; m<(globalData.n_count-1); m++)
      {
            if(cp_key[m] < cp_key[m+1])     //* max->min /
            {
                  buf_int      = cp_key[m];          //*
                  cp_key[m]    = cp_key[m+1];        //*
                  cp_key[m+1] = buf_int;             //*

                  buf_index  = index[m];
                  index[m]   = index[m+1];
                  index[m+1] = buf_index;
            }
      }
    }
    for(n=0; n<globalData.n_count; n++)           //give score
    {
          sc_key[index[n]] = score;                     //*
          if(cp_key[n] != cp_key[n+1])          //*
          {
                score--;
          }
    }
    ////////////Give Score (Key)////////////

    ////////////Give Score (Spread_0)////////////
    score = 10;
    for(n=0; n<globalData.n_count; n++)           //copy data
    {     index[n] = n;
          cp_spread_0[n] = globalData.n_spread_0[n];    //*
```

```
}

for(n=0; n<(globalData.n_count-1); n++)        //sort data
{
for(m=0; m<(globalData.n_count-1); m++)
  {
        if(cp_spread_0[m] < cp_spread_0[m+1]) //* max->min
        {
              buf_int          = cp_spread_0[m];    //*
              cp_spread_0[m]   = cp_spread_0[m+1];  //*
              cp_spread_0[m+1] = buf_int;                    //*

              buf_index  = index[m];
              index[m]   = index[m+1];
              index[m+1] = buf_index;
        }
  }
}
for(n=0; n<globalData.n_count; n++)            //give score
{
      sc_spread_0[index[n]] = score;                          //*
      if(cp_spread_0[n] != cp_spread_0[n+1])      //*
      {
            score--;
      }
}
///////////Give Score (Spread_0)///////////

///////////Give Score (Spread_1)///////////
score = 10;
for(n=0; n<globalData.n_count; n++)            //copy data
{     index[n] = n;
      cp_spread_1[n] = globalData.n_spread_1[n];    //*
}

for(n=0; n<(globalData.n_count-1); n++)        //sort data
{
for(m=0; m<(globalData.n_count-1); m++)
  {
        if(cp_spread_1[m] < cp_spread_1[m+1]) //* max->min/
        {
              buf_int          = cp_spread_1[m];    //*
              cp_spread_1[m]   = cp_spread_1[m+1];  //*
              cp_spread_1[m+1] = buf_int;                    //*

              buf_index  = index[m];
              index[m]   = index[m+1];
              index[m+1] = buf_index;
        }
  }
}
for(n=0; n<globalData.n_count; n++)            //give score
{
      sc_spread_0[index[n]] = score;                          //*
      if(cp_spread_1[n] != cp_spread_1[n+1])      //*
      {
            score--;
      }
}
///////////Give Score (Spread_1)///////////

///////////Give Score (File Size)///////////
score = 10;
for(n=0; n<globalData.n_count; n++)            //copy data
{     index[n] = n;
```

```
        cp_filesize[n] = globalData.n_filesize_destination[n];      //*
}

for(n=0; n<(globalData.n_count-1); n++)          //sort data
{
for(m=0; m<(globalData.n_count-1); m++)
  {
        if(cp_filesize[m] < cp_filesize[m+1]) //* max->min
        {
              buf_long         = cp_filesize[m];    //*
              cp_filesize[m]   = cp_filesize[m+1];  //*
              cp_filesize[m+1] = buf_long;                  //*

              buf_index  = index[m];
              index[m]   = index[m+1];
              index[m+1] = buf_index;
        }
  }
}
for(n=0; n<globalData.n_count; n++)              //give score
{
      sc_filesize[index[n]] = score;                        //*
      if(cp_filesize[n] != cp_filesize[n+1])      //*
      {
            score--;
      }
}
///////////Give Score (File Size)///////////


/////////Calculate total score and Sort
for(n=0; n<globalData.n_count; n++)              //cal
{
      sc_total[n] = (sc_speed_n[n] * globalData.w_speed_n)+
                    (sc_speed_d[n] * globalData.w_speed_d)+
                  (sc_key[n]     * globalData.w_key)       +
                    (sc_spread_0[n]* globalData.w_spread) +
                    (sc_spread_1[n]* globalData.w_spread) +
                    (sc_filesize[n]* globalData.w_filesize);


}

for(n=0; n<globalData.n_count; n++)              //set index
{     index[n] = n;
}


for(n=0; n<(globalData.n_count-1); n++)          //sort data
{
for(m=0; m<(globalData.n_count-1); m++)
  {
        if(sc_total[m] < sc_total[m+1])//* max->min /
        {
              buf_int        = sc_total[m];          //*
              sc_total[m]    = sc_total[m+1]; //*
              sc_total[m+1] = buf_int;               //*

              buf_index  = index[m];
              index[m]   = index[m+1];
              index[m+1] = buf_index;
        }
  }
}
/////////Calculate total score and Sort
```

```
        ////////////////////graph///////////////
//      int n;
        int max_value;                                                  //*
        int set_space;
        int set_range;
        CGraphSeries* series[9];
        testGraph = new CGraph();
        testGraph->SetGraphTitle("Efficiency");         //*

        //cal range
        max_value = 0;                                                  //*
        for(n=0; n<globalData.n_count; n++)
        {
              if(max_value < sc_total[n])                               //*
              {  max_value = sc_total[n];}              //*
        }


        /////////Cal Scale////////
        long range, scale, expo;
              range = 10;
              scale = 1;
              expo  = 1;

              while(range < max_value)
              {             scale = 1;
                            expo  = expo * 10;
                    while((range < max_value) && (scale<=10))
                    {
                            range = scale * expo;
                            scale++;
                    }
              }
                            scale--;

                            while(scale < 6){ scale = scale *2; }
                            set_space = range/scale;
                            set_range = range;
        /////////Cal Scale////////

        //cal range
        testGraph->SetTickSpace(set_space);                             //*
        testGraph->SetTickRange(set_range);                             //*

        testGraph->SetXAxisAlignment(90);
        testGraph->SetXAxisLabel("Algorithm");
        testGraph->SetYAxisLabel("Score");                              //*
        testGraph->SetLegend(0, "Score");               //*


        //set up some series
        for(n=0; n<globalData.n_count; n++)
        {
          series[n] = new CGraphSeries();
          series[n]->SetLabel(globalData.n_buffer[index[n]]);
          series[n]->SetData(0, sc_total[n]);           //*

       testGraph->AddSeries(series[n]);
        }

        for(n=0; n<=9; n++) //parameter pass to report
        {
        globalData.isort[n] = index[n];
        globalData.effi[n]  = sc_total[n];
```

```
#if !defined(AFX_TAB1_H__745714F8_F404_406E_8818_3544DE3D0B05__INCLUDED_)
#define AFX_TAB1_H__745714F8_F404_406E_8818_3544DE3D0B05__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// tab1.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// tab1 dialog

class tab1 : public CPropertyPage
{
	DECLARE_DYNCREATE(tab1)

// Construction
public:
	tab1();
	~tab1();

// Dialog Data
	//{{AFX_DATA(tab1)
	enum { IDD = IDD_DIALOG_CRYPT };
	CEdit  m_max;
	CProgressCtrl m_progress_crypt;
	CAnimateCtrl  m_animate;
	int           m_algor;
	CString       m_password;
	CString       m_key;
	CString       m_source;
	CString       m_destination;
	//}}AFX_DATA


// Overrides
	// ClassWizard generate virtual function overrides
	//{{AFX_VIRTUAL(tab1)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual BOOL OnInitDialog(); //*

protected:
	HICON m_hIcon;      //*
	// Generated message map functions
	//{{AFX_MSG(tab1)
	afx_msg void OnButtonGenkey();
	afx_msg void OnButtonKeyfile();
	afx_msg void OnButtonSource();
	afx_msg void OnSelchangeComboAlgor();
	afx_msg void OnButtonCrypt();
	afx_msg void OnMaxtextEditPassword();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

	//Symmetic
	void des_N_crypt();
	void des_D_crypt();
	void idea_N_crypt();
	void idea_D_crypt();
	void blowfish_N_crypt();
```

```
        void blowfish_D_crypt();
        void cast128_N_crypt();
        void cast128_D_crypt();
        void gost_N_crypt();
        void gost_D_crypt();
        void mars_N_crypt();
        void mars_D_crypt();
        void rc2_N_crypt();
        void rc2_D_crypt();
        void rijndael_N_crypt();
        void rijndael_D_crypt();
        void serpent_N_crypt();
        void serpent_D_crypt();
        void shark_N_crypt();
        void shark_D_crypt();
        void square_N_crypt();
        void square_D_crypt();
        void tea_N_crypt();
        void tea_D_crypt();
        void threeway_N_crypt();
        void threeway_D_crypt();
        void twofish_N_crypt();
        void twofish_D_crypt();

        //Public
        void rsa_N_crypt();
        void rsa_D_crypt();
        void luc_N_crypt();
        void luc_D_crypt();
        void rabin_N_crypt();
        void rabin_D_crypt();



        //Code Crypt String [For Pubilc]
        char* RSAEncryptString(const char *pubFilename, const char *seed, const
char *message);
        char* RSADecryptString(const char *privFilename, const char
*ciphertext);
        char* LUCEncryptString(const char *pubFilename, const char *seed, const
char *message);
        char* LUCDecryptString(const char *privFilename, const char
*ciphertext);
        char* RabinEncryptString(const char *pubFilename, const char *seed,
const char *message);
        char* RabinDecryptString(const char *privFilename, const char
*ciphertext);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_TAB1_H__745714F8_F404_406E_8818_3544DE3D0B05__INCLUDED_)
```

```
// tab1.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "tab1.h"

#include "GenkeyDlg.h"                    //include for New Dialog

//////////////spacial include//////////////////
#include "x_include\des.h"
#include "x_include\idea.h"
#include "x_include\blowfish.h"
#include "x_include\cast.h"
#include "x_include\gost.h"
#include "x_include\mars.h"
#include "x_include\rc2.h"
#include "x_include\rijndael.h"
#include "x_include\serpent.h"
#include "x_include\shark.h"
#include "x_include\square.h"
#include "x_include\tea.h"
#include "x_include\3way.h"
#include "x_include\twofish.h"

#include "x_include\rsa.h"
#include "x_include\luc.h"
#include "x_include\rabin.h"
#include "x_include\randpool.h"
#include "x_include\files.h"
#include "x_include\hex.h"

//#include "Afxdisp.h"

USING_NAMESPACE(CryptoPP)
USING_NAMESPACE(std)
//////////////spacial include//////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// tab1 property page

IMPLEMENT_DYNCREATE(tab1, CPropertyPage)

tab1::tab1() : CPropertyPage(tab1::IDD)
{
```

```
	//{{AFX_DATA_INIT(tab1)
	m_password = _T("");
	m_key = _T("");
	m_source = _T("");
	//}}AFX_DATA_INIT
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

tab1::~tab1()
{
}

void tab1::DoDataExchange(CDataExchange* pDX)
{
	CPropertyPage::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(tab1)
	DDX_Control(pDX, IDC_EDIT_PASSWORD, m_max);
	DDX_Control(pDX, IDC_PROGRESS_CRYPT, m_progress_crypt);
	DDX_Control(pDX, IDC_ANIMATE, m_animate);
	DDX_CBIndex(pDX, IDC_COMBO_ALGOR, m_algor);
	DDX_Text(pDX, IDC_EDIT_PASSWORD, m_password);
	DDX_Text(pDX, IDC_STATIC_KEY, m_key);
	DDX_Text(pDX, IDC_EDIT_SOURCE, m_source);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(tab1, CPropertyPage)
	//{{AFX_MSG_MAP(tab1)
	ON_BN_CLICKED(IDC_BUTTON_GENKEY, OnButtonGenkey)
	ON_BN_CLICKED(IDC_BUTTON_KEYFILE, OnButtonKeyfile)
	ON_BN_CLICKED(IDC_BUTTON_SOURCE, OnButtonSource)
	ON_CBN_SELCHANGE(IDC_COMBO_ALGOR, OnSelchangeComboAlgor)
	ON_BN_CLICKED(IDC_BUTTON_CRYPT, OnButtonCrypt)
	ON_EN_MAXTEXT(IDC_EDIT_PASSWORD, OnMaxtextEditPassword)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// tab1 message handlers
BOOL tab1::OnInitDialog()
{
	CancelToClose();

//	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	m_source="Select Source FilE->";
	m_key="Password";
//	m_destination="Select Dest->";
    UpdateData(FALSE);

	m_progress_crypt.SetRange(0, 100);
	m_progress_crypt.SetStep(5);
	m_progress_crypt.SetPos(0);

	m_animate.Open(IDR_AVI);

	return CPropertyPage::OnInitDialog();
```

```
}

void tab1::OnButtonSource()
{
       CFileDialog fd(TRUE, NULL, NULL, NULL,
   "Text File|*.txt|Picture File|*.jpg; *.gif; *.bmp|Wave File|*.wav;
*.mp3|All File|*.*||");
       if(fd.DoModal()==IDOK)
       {
              m_source = fd.GetFileName();
              UpdateData(FALSE);
       }

}
void tab1::OnButtonKeyfile()
{
       CFileDialog fd(TRUE, NULL, NULL, NULL,
                              "RSA|*.rsa|LUC|*.luc|Rabin|*.rbn|All
File|*.*||");
       if(fd.DoModal()==IDOK)
       {
              m_password = fd.GetFileName();
              UpdateData(FALSE);
       }
}


void tab1::OnButtonGenkey()
{

       GenkeyDlg dialog;
       dialog.DoModal();

}

////////////Function Crypt Algorithm//////////
void tab1::des_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==8)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<8; m++)              //8 key
       {
```

```
  key[m] = m_password.GetAt(m);     //Copy Key
}

DESEncryption algorE(key);

step = (filesize / fileread) / 21;  //calculate step progress

do
{
nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


countstep++;                        //calculate about progress
 if((countstep > step) && (step > 0))
 {
      m_progress_crypt.StepIt();
      countstep = 0;
 }


 if(nBytesRead!=fileread)
 {
      for(int n=nBytesRead; n<fileread; n++)
      {
           m_byte[n]=' ';
      }
      switch(nBytesRead)                //Mem Last Byte Block
      {
           case 1 : lastbyte = '1'; break;
           case 2 : lastbyte = '2'; break;
           case 3 : lastbyte = '3'; break;
           case 4 : lastbyte = '4'; break;
           case 5 : lastbyte = '5'; break;
           case 6 : lastbyte = '6'; break;
           case 7 : lastbyte = '7'; break;
           case 8 : lastbyte = '8'; break;
           case 9 : lastbyte = '9'; break;
           case 10: lastbyte = 'A'; break;
           case 11: lastbyte = 'B'; break;
           case 12: lastbyte = 'C'; break;
           case 13: lastbyte = 'D'; break;
           case 14: lastbyte = 'E'; break;
           case 15: lastbyte = 'F'; break;
           case 16: lastbyte = 'G'; break;
      }
 }

      algorE.ProcessBlock(m_byte);


if(nBytesRead != 0)
      {
           fileSink.Write(m_byte, filewrite);  //Write
      }

}while(nBytesRead==fileread);

           fileSink.Write(&lastbyte, 1);  //Write

//Encryption;

fileSource.Close();
fileSink.Close();

  m_password.Empty();       //Delete password
```

```
        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }
}

void tab1::des_D_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==8)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Decryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
      byte m_byte[8];
      byte m_byte2[8];
    UINT nBytesRead;
      UINT nBytesRead2;
      int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
      for(int m=0; m<8; m++)
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      DESDecryption algorD(key);

      step = (filesize / fileread) / 21;  //calculate step progress


      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

      do
      {

      countstep++;

       if((countstep > step) && (step > 0))                    //calculate
about progress
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);
```

```
        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
            }
        }

        if(nBytesRead2==1)                  //last byte block
        {
              fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
              fileSink.Write(m_byte, filewrite);  //Write
        }

            nBytesRead = nBytesRead2;          //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                m_byte[n] = m_byte2[n];
            }

      }while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();                 //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);

  }

}

////////////

void tab1::idea_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16) //key
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
```

```
    CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

    //Encryption;

    int filesize = fileSource.GetLength(); //about progress
    int fileread  = 8;  //block
    int filewrite = 8;  //block
    int countstep = 0;
    int step = 0;
  UINT nBytesRead;
    byte m_byte[8];     //block
    char lastbyte = '8';       //block
  byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,   //key
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

    for(int m=0; m<16; m++)           //key
    {
      key[m] = m_password.GetAt(m);     //Copy Key
    }

    IDEAEncryption algorE(key);

    step = (filesize / fileread) / 21;  //calculate step progress

    do
    {
    nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


    countstep++;                      //calculate about progress
     if((countstep > step) && (step > 0))
     {
          m_progress_crypt.StepIt();
          countstep = 0;
     }


     if(nBytesRead!=fileread)
     {
          for(int n=nBytesRead; n<fileread; n++)
          {
               m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
               case 1 : lastbyte = '1'; break;
               case 2 : lastbyte = '2'; break;
               case 3 : lastbyte = '3'; break;
               case 4 : lastbyte = '4'; break;
               case 5 : lastbyte = '5'; break;
               case 6 : lastbyte = '6'; break;
               case 7 : lastbyte = '7'; break;
               case 8 : lastbyte = '8'; break;
               case 9 : lastbyte = '9'; break;
               case 10: lastbyte = 'A'; break;
               case 11: lastbyte = 'B'; break;
               case 12: lastbyte = 'C'; break;
               case 13: lastbyte = 'D'; break;
               case 14: lastbyte = 'E'; break;
               case 15: lastbyte = 'F'; break;
               case 16: lastbyte = 'G'; break;
          }
```

```
        )
              algorE.ProcessBlock(m_byte);

        if(nBytesRead != 0)
              {
                    fileSink.Write(m_byte, filewrite);  //Write
              }

        }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
   }

}

void tab1::idea_D_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
       byte m_byte[8];
       byte m_byte2[8];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
       for(int m=0; m<16; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       IDEADecryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress
```

```
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


          algorD.ProcessBlock(m_byte);


          nBytesRead2 = fileSource.Read(m_byte2, fileread);

          if(nBytesRead2==1)
          {
               switch(m_byte2[0])                    //Mem Last Byte Block
               {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
               }
          }

          if(nBytesRead2==1)                  //last byte block
          {
                  fileSink.Write(m_byte, lastbyte);  //Write
          }
          else
          {
                  fileSink.Write(m_byte, filewrite);  //Write
          }

               nBytesRead = nBytesRead2;          //Transfer Byte
               for(int n=0; n<fileread; n++)
               {
                    m_byte[n] = m_byte2[n];
               }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();
```

```
        m_password.Empty();             //Delete password
        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

////////////

void tab1::blowfish_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[8];
      char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

      for(int m=0; m<16; m++)           //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      BlowfishEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


      countstep++;                      //calculate about progress
       if((countstep > step) && (step > 0))
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       if(nBytesRead!=fileread)
       {
            for(int n=nBytesRead; n<fileread; n++)
```

```
                {
                        m_byte[n]=' ';
                }
                switch(nBytesRead)                  //Mem Last Byte Block
                {
                        case 1 : lastbyte = '1'; break;
                        case 2 : lastbyte = '2'; break;
                        case 3 : lastbyte = '3'; break;
                        case 4 : lastbyte = '4'; break;
                        case 5 : lastbyte = '5'; break;
                        case 6 : lastbyte = '6'; break;
                        case 7 : lastbyte = '7'; break;
                        case 8 : lastbyte = '8'; break;
                        case 9 : lastbyte = '9'; break;
                        case 10: lastbyte = 'A'; break;
                        case 11: lastbyte = 'B'; break;
                        case 12: lastbyte = 'C'; break;
                        case 13: lastbyte = 'D'; break;
                        case 14: lastbyte = 'E'; break;
                        case 15: lastbyte = 'F'; break;
                        case 16: lastbyte = 'G'; break;
                }
        }
                algorE.ProcessBlock(m_byte);

        if(nBytesRead != 0)
                {
                fileSink.Write(m_byte, filewrite);  //Write
                }

        }while(nBytesRead==fileread);

                        fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
    }

}

void tab1::blowfish_D_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
```

```
        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        BlowfishDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


          algorD.ProcessBlock(m_byte);


          nBytesRead2 = fileSource.Read(m_byte2, fileread);

          if(nBytesRead2==1)
          {
               switch(m_byte2[0])                    //Mem Last Byte Block
               {
                     case '1' : lastbyte = 1; break;
                     case '2' : lastbyte = 2; break;
                     case '3' : lastbyte = 3; break;
                     case '4' : lastbyte = 4; break;
                     case '5' : lastbyte = 5; break;
                     case '6' : lastbyte = 6; break;
                     case '7' : lastbyte = 7; break;
                     case '8' : lastbyte = 8; break;
                     case '9' : lastbyte = 9; break;
                     case 'A' : lastbyte = 10; break;
                     case 'B' : lastbyte = 11; break;
                     case 'C' : lastbyte = 12; break;
                     case 'D' : lastbyte = 13; break;
                     case 'E' : lastbyte = 14; break;
                     case 'F' : lastbyte = 15; break;
                     case 'G' : lastbyte = 16; break;
               }
```

```
        }
        if(nBytesRead2==1)              //last byte block
        {
                fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
                fileSink.Write(m_byte, filewrite);  //Write
        }
             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                   m_byte[n] = m_byte2[n];
             }
       }while(nBytesRead==fileread);
        //Decryption;
       fileSource.Close();
       fileSink.Close();
         m_password.Empty();              //Delete password
         UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }
}
////////////
void tab1::cast128_N_crypt()
{
        UpdateData(TRUE);
  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
      //Encryption;
      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[8];
      char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
      for(int m=0; m<16; m++)           //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
```

```
}

CAST128Encryption algorE(key);

step = (filesize / fileread) / 21;  //calculate step progress

do
{
nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


countstep++;                        //calculate about progress
 if((countstep > step) && (step > 0))
 {
      m_progress_crypt.StepIt();
      countstep = 0;
 }


 if(nBytesRead!=fileread)
 {
      for(int n=nBytesRead; n<fileread; n++)
      {
           m_byte[n]=' ';
      }
      switch(nBytesRead)                  //Mem Last Byte Block
      {
           case 1 : lastbyte = '1'; break;
           case 2 : lastbyte = '2'; break;
           case 3 : lastbyte = '3'; break;
           case 4 : lastbyte = '4'; break;
           case 5 : lastbyte = '5'; break;
           case 6 : lastbyte = '6'; break;
           case 7 : lastbyte = '7'; break;
           case 8 : lastbyte = '8'; break;
           case 9 : lastbyte = '9'; break;
           case 10: lastbyte = 'A'; break;
           case 11: lastbyte = 'B'; break;
           case 12: lastbyte = 'C'; break;
           case 13: lastbyte = 'D'; break;
           case 14: lastbyte = 'E'; break;
           case 15: lastbyte = 'F'; break;
           case 16: lastbyte = 'G'; break;
      }
 }

      algorE.ProcessBlock(m_byte);


if(nBytesRead != 0)
      {
           fileSink.Write(m_byte, filewrite);  //Write
      }

}while(nBytesRead==fileread);

           fileSink.Write(&lastbyte, 1);  //Write

//Encryption;

fileSource.Close();
fileSink.Close();

  m_password.Empty();       //Delete password
  UpdateData(FALSE);
```

```
        m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
    }

}

void tab1::cast128_D_crypt()
{
         UpdateData(TRUE);

   if(m_password.GetLength()==16)
   {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                           0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        CAST128Decryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


          algorD.ProcessBlock(m_byte);
```

```
        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
                  case '2' : lastbyte = 2; break;
                  case '3' : lastbyte = 3; break;
                  case '4' : lastbyte = 4; break;
                  case '5' : lastbyte = 5; break;
                  case '6' : lastbyte = 6; break;
                  case '7' : lastbyte = 7; break;
                  case '8' : lastbyte = 8; break;
                  case '9' : lastbyte = 9; break;
                  case 'A' : lastbyte = 10; break;
                  case 'B' : lastbyte = 11; break;
                  case 'C' : lastbyte = 12; break;
                  case 'D' : lastbyte = 13; break;
                  case 'E' : lastbyte = 14; break;
                  case 'F' : lastbyte = 15; break;
                  case 'G' : lastbyte = 16; break;
            }
        }

        if(nBytesRead2==1)                  //last byte block
        {
              fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
              fileSink.Write(m_byte, filewrite);  //Write
        }

            nBytesRead = nBytesRead2;          //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                m_byte[n] = m_byte2[n];
            }

      }while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();                 //Delete password
        UpdateData(FALSE);
    m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

////////////

void tab1::gost_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==32)
  {
```

```
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
      UINT nBytesRead;
        byte m_byte[8];
        char lastbyte = '8';
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

        for(int m=0; m<32; m++)           //key
        {
          key[m] = m_password.GetAt(m);    //Copy Key
        }

        GOSTEncryption algorE(key);

        step = (filesize / fileread) / 21;  //calculate step progress

        do
        {
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


        countstep++;                      //calculate about progress
         if((countstep > step) && (step > 0))
         {
               m_progress_crypt.StepIt();
               countstep = 0;
         }


         if(nBytesRead!=fileread)
         {
               for(int n=nBytesRead; n<fileread; n++)
               {
                     m_byte[n]=' ';
               }
               switch(nBytesRead)              //Mem Last Byte Block
               {
                     case 1 : lastbyte = '1'; break;
                     case 2 : lastbyte = '2'; break;
                     case 3 : lastbyte = '3'; break;
                     case 4 : lastbyte = '4'; break;
                     case 5 : lastbyte = '5'; break;
                     case 6 : lastbyte = '6'; break;
                     case 7 : lastbyte = '7'; break;
                     case 8 : lastbyte = '8'; break;
                     case 9 : lastbyte = '9'; break;
                     case 10: lastbyte = 'A'; break;
                     case 11: lastbyte = 'B'; break;
                     case 12: lastbyte = 'C'; break;
                     case 13: lastbyte = 'D'; break;
```

```
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

              m_password.Empty();       //Delete password
              UpdateData(FALSE);
            m_progress_crypt.SetPos(0); //set step progress at 0
        }
        else
        {
                MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
        }

}

void tab1::gost_D_crypt()
{
            UpdateData(TRUE);

        if(m_password.GetLength()==32)
        {
            CStdioFile fileSource(m_source,
                                CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 8;
            int filewrite = 8;
            int countstep = 0;
            int step = 0;
            byte m_byte[8];
            byte m_byte2[8];
          UINT nBytesRead;
            UINT nBytesRead2;
            int lastbyte;
          byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
            for(int m=0; m<32; m++)
            {
```

```
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      GOSTDecryption algorD(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

      do
      {

      countstep++;

       if((countstep > step) && (step > 0))                    //calculate
about progress
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }

       algorD.ProcessBlock(m_byte);

       nBytesRead2 = fileSource.Read(m_byte2, fileread);

       if(nBytesRead2==1)
       {
            switch(m_byte2[0])              //Mem Last Byte Block
            {
                 case '1' : lastbyte = 1; break;
                 case '2' : lastbyte = 2; break;
                 case '3' : lastbyte = 3; break;
                 case '4' : lastbyte = 4; break;
                 case '5' : lastbyte = 5; break;
                 case '6' : lastbyte = 6; break;
                 case '7' : lastbyte = 7; break;
                 case '8' : lastbyte = 8; break;
                 case '9' : lastbyte = 9; break;
                 case 'A' : lastbyte = 10; break;
                 case 'B' : lastbyte = 11; break;
                 case 'C' : lastbyte = 12; break;
                 case 'D' : lastbyte = 13; break;
                 case 'E' : lastbyte = 14; break;
                 case 'F' : lastbyte = 15; break;
                 case 'G' : lastbyte = 16; break;
            }
       }

       if(nBytesRead2==1)              //last byte block
       {
              fileSink.Write(m_byte, lastbyte);  //Write
       }
       else
       {
              fileSink.Write(m_byte, filewrite);  //Write
       }

            nBytesRead = nBytesRead2;         //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                 m_byte[n] = m_byte2[n];
            }
```

```
        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();             //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                  MB_OK/MB_ICONINFORMATION);
   }

}

////////////

void tab1::mars_N_crypt()
{
         UpdateData(TRUE);

   if(m_password.GetLength()==16)
   {
        CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                          CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
     UINT nBytesRead;
        byte m_byte[16];
        char lastbyte = 'G';
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

        for(int m=0; m<16; m++)           //key
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        MARSEncryption algorE(key);

        step = (filesize / fileread) / 21;  //calculate step progress

        do
        {
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


        countstep++;                        //calculate about progress
         if((countstep > step) && (step > 0))
         {
              m_progress_crypt.StepIt();
              countstep = 0;
```

```
        }

        if(nBytesRead!=fileread)
        {
              for(int n=nBytesRead; n<fileread; n++)
              {
                    m_byte[n]=' ';
              }
              switch(nBytesRead)                    //Mem Last Byte Block
              {
                    case 1 : lastbyte = '1'; break;
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
              }
        }
              algorE.ProcessBlock(m_byte);

       if(nBytesRead != 0)
              {
                    fileSink.Write(m_byte, filewrite);  //Write
              }

       }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

       //Encryption;

       fileSource.Close();
       fileSink.Close();

         m_password.Empty();        //Delete password
         UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

void tab1::mars_D_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
```

```
                                CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        byte m_byte[16];
        byte m_byte2[16];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        MARSDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                  //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
```

```
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                        m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();               //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {
            MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

////////////

void tab1::rc2_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[8];
      char lastbyte = '8';
```

```
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   RC2Encryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                        //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
                m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);


   if(nBytesRead != 0)
          {
                fileSink.Write(m_byte, filewrite);  //Write
          }

   }while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write

   //Encryption;
```

```
        fileSource.Close();
        fileSink.Close();

          m_password.Empty();        //Delete password
          UpdateData(FALSE);
        m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
    }

}

void tab1::rc2_D_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        RC2Decryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }
```

```
            algorD.ProcessBlock(m_byte);


            nBytesRead2 = fileSource.Read(m_byte2, fileread);

            if(nBytesRead2==1)
            {
                 switch(m_byte2[0])                  //Mem Last Byte Block
                 {
                        case '1' : lastbyte = 1; break;
                        case '2' : lastbyte = 2; break;
                        case '3' : lastbyte = 3; break;
                        case '4' : lastbyte = 4; break;
                        case '5' : lastbyte = 5; break;
                        case '6' : lastbyte = 6; break;
                        case '7' : lastbyte = 7; break;
                        case '8' : lastbyte = 8; break;
                        case '9' : lastbyte = 9; break;
                        case 'A' : lastbyte = 10; break;
                        case 'B' : lastbyte = 11; break;
                        case 'C' : lastbyte = 12; break;
                        case 'D' : lastbyte = 13; break;
                        case 'E' : lastbyte = 14; break;
                        case 'F' : lastbyte = 15; break;
                        case 'G' : lastbyte = 16; break;
                 }
            }

            if(nBytesRead2==1)                  //last byte block
            {
                  fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                  fileSink.Write(m_byte, filewrite);  //Write
            }

                 nBytesRead = nBytesRead2;          //Transfer Byte
                 for(int n=0; n<fileread; n++)
                 {
                       m_byte[n] = m_byte2[n];
                 }

          }while(nBytesRead==fileread);
           //Decryption;

          fileSource.Close();
          fileSink.Close();

            m_password.Empty();              //Delete password
            UpdateData(FALSE);
         m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {          MessageBox("Password not Support", "Password",
                                 MB_OK/MB_ICONINFORMATION);
      }

}

//////////////

void tab1::rijndael_N_crypt()
```

```
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                            CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                          CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 16;
      int filewrite = 16;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[16];
      char lastbyte = 'G';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

      for(int m=0; m<16; m++)          //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      RijndaelEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


      countstep++;                        //calculate about progress
       if((countstep > step) && (step > 0))
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       if(nBytesRead!=fileread)
       {
            for(int n=nBytesRead; n<fileread; n++)
            {
                  m_byte[n]=' ';
            }
            switch(nBytesRead)                //Mem Last Byte Block
            {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
                  case 6 : lastbyte = '6'; break;
                  case 7 : lastbyte = '7'; break;
                  case 8 : lastbyte = '8'; break;
                  case 9 : lastbyte = '9'; break;
                  case 10: lastbyte = 'A'; break;
```

```
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

              m_password.Empty();       //Delete password
              UpdateData(FALSE);
            m_progress_crypt.SetPos(0); //set step progress at 0
        }
        else
        {
                MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
        }

}

void tab1::rijndael_D_crypt()
{
            UpdateData(TRUE);

        if(m_password.GetLength()==16)
        {
            CStdioFile fileSource(m_source,
                                CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 16;
            int filewrite = 16;
            int countstep = 0;
            int step = 0;
            byte m_byte[16];
            byte m_byte2[16];
          UINT nBytesRead;
            UINT nBytesRead2;
            int lastbyte;
          byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
            for(int m=0; m<16; m++)
```

```
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        RijndaelDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
               m_progress_crypt.StepIt();
               countstep = 0;
         }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
               switch(m_byte2[0])                    //Mem Last Byte Block
               {
                     case '1' : lastbyte = 1; break;
                     case '2' : lastbyte = 2; break;
                     case '3' : lastbyte = 3; break;
                     case '4' : lastbyte = 4; break;
                     case '5' : lastbyte = 5; break;
                     case '6' : lastbyte = 6; break;
                     case '7' : lastbyte = 7; break;
                     case '8' : lastbyte = 8; break;
                     case '9' : lastbyte = 9; break;
                     case 'A' : lastbyte = 10; break;
                     case 'B' : lastbyte = 11; break;
                     case 'C' : lastbyte = 12; break;
                     case 'D' : lastbyte = 13; break;
                     case 'E' : lastbyte = 14; break;
                     case 'F' : lastbyte = 15; break;
                     case 'G' : lastbyte = 16; break;
               }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(m_byte, filewrite);  //Write
         }

               nBytesRead = nBytesRead2;          //Transfer Byte
               for(int n=0; n<fileread; n++)
               {
                     m_byte[n] = m_byte2[n];
```

```
            }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();             //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

   }

}

////////////

void tab1::serpent_N_crypt()
{
        UpdateData(TRUE);

   if(m_password.GetLength()==16)
   {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 16;
       int countstep = 0;
       int step = 0;
     UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)           //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       SerpentEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                        //calculate about progress
        if((countstep > step) && (step > 0))
        {
            m_progress_crypt.StepIt();
```

```
            countstep = 0;
        }

        if(nBytesRead!=fileread)
        {
            for(int n=nBytesRead; n<fileread; n++)
            {
                m_byte[n]=' ';
            }
            switch(nBytesRead)                  //Mem Last Byte Block
            {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
            }
        }
            algorE.ProcessBlock(m_byte);

      if(nBytesRead != 0)
            {
                fileSink.Write(m_byte, filewrite);  //Write
            }

      }while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write

      //Encryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();         //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);

  }

}

void tab1::serpent_D_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
```

```
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        byte m_byte[16];
        byte m_byte2[16];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        SerpentDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


          algorD.ProcessBlock(m_byte);


          nBytesRead2 = fileSource.Read(m_byte2, fileread);

          if(nBytesRead2==1)
          {
               switch(m_byte2[0])                  //Mem Last Byte Block
               {
                     case '1' : lastbyte = 1; break;
                     case '2' : lastbyte = 2; break;
                     case '3' : lastbyte = 3; break;
                     case '4' : lastbyte = 4; break;
                     case '5' : lastbyte = 5; break;
                     case '6' : lastbyte = 6; break;
                     case '7' : lastbyte = 7; break;
                     case '8' : lastbyte = 8; break;
                     case '9' : lastbyte = 9; break;
                     case 'A' : lastbyte = 10; break;
```

```
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();               //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {
            MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

////////////

void tab1::shark_N_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                            CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
      UINT nBytesRead;
        byte m_byte[8];
```

```
   char lastbyte = '8';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   SHARKEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                          //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
               m_byte[n]=' ';
          }
          switch(nBytesRead)                  //Mem Last Byte Block
          {
                    case 1 : lastbyte = '1'; break;
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);


   if(nBytesRead != 0)
          {
                fileSink.Write(m_byte, filewrite);  //Write
          }

   }while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write
```

```
        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
        m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

void tab1::shark_D_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        SHARKDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
          {
                m_progress_crypt.StepIt();
                countstep = 0;
```

```
        }

        algorD.ProcessBlock(m_byte);

        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
                  case '2' : lastbyte = 2; break;
                  case '3' : lastbyte = 3; break;
                  case '4' : lastbyte = 4; break;
                  case '5' : lastbyte = 5; break;
                  case '6' : lastbyte = 6; break;
                  case '7' : lastbyte = 7; break;
                  case '8' : lastbyte = 8; break;
                  case '9' : lastbyte = 9; break;
                  case 'A' : lastbyte = 10; break;
                  case 'B' : lastbyte = 11; break;
                  case 'C' : lastbyte = 12; break;
                  case 'D' : lastbyte = 13; break;
                  case 'E' : lastbyte = 14; break;
                  case 'F' : lastbyte = 15; break;
                  case 'G' : lastbyte = 16; break;
            }
        }

        if(nBytesRead2==1)                  //last byte block
        {
              fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
              fileSink.Write(m_byte, filewrite);  //Write
        }

            nBytesRead = nBytesRead2;          //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                  m_byte[n] = m_byte2[n];
            }

      )while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();                 //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

//////////////
```

```
void tab1::square_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                      CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 16;
      int filewrite = 16;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[16];
      char lastbyte = 'G';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

      for(int m=0; m<16; m++)           //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      SquareEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


      countstep++;                      //calculate about progress
       if((countstep > step) && (step > 0))
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       if(nBytesRead!=fileread)
       {
            for(int n=nBytesRead; n<fileread; n++)
            {
                  m_byte[n]=' ';
            }
            switch(nBytesRead)                //Mem Last Byte Block
            {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
                  case 6 : lastbyte = '6'; break;
                  case 7 : lastbyte = '7'; break;
                  case 8 : lastbyte = '8'; break;
                  case 9 : lastbyte = '9'; break;
```

```
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

              m_password.Empty();       //Delete password
              UpdateData(FALSE);
            m_progress_crypt.SetPos(0); //set step progress at 0
        }
        else
        {
                MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
        }

}

void tab1::square_D_crypt()
{
            UpdateData(TRUE);

        if(m_password.GetLength()==16)
        {
            CStdioFile fileSource(m_source,
                                  CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                              CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 16;
            int filewrite = 16;
            int countstep = 0;
            int step = 0;
            byte m_byte[16];
            byte m_byte2[16];
          UINT nBytesRead;
            UINT nBytesRead2;
            int lastbyte;
          byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
```

```
for(int m=0; m<16; m++)
{
  key[m] = m_password.GetAt(m);     //Copy Key
}

SquareDecryption algorD(key);

step = (filesize / fileread) / 21;  //calculate step progress


nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

do
{

countstep++;

 if((countstep > step) && (step > 0))                              //calculate
about progress
 {
      m_progress_crypt.StepIt();
      countstep = 0;
 }


  algorD.ProcessBlock(m_byte);


  nBytesRead2 = fileSource.Read(m_byte2, fileread);

  if(nBytesRead2==1)
  {
      switch(m_byte2[0])                  //Mem Last Byte Block
      {
            case '1' : lastbyte = 1; break;
            case '2' : lastbyte = 2; break;
            case '3' : lastbyte = 3; break;
            case '4' : lastbyte = 4; break;
            case '5' : lastbyte = 5; break;
            case '6' : lastbyte = 6; break;
            case '7' : lastbyte = 7; break;
            case '8' : lastbyte = 8; break;
            case '9' : lastbyte = 9; break;
            case 'A' : lastbyte = 10; break;
            case 'B' : lastbyte = 11; break;
            case 'C' : lastbyte = 12; break;
            case 'D' : lastbyte = 13; break;
            case 'E' : lastbyte = 14; break;
            case 'F' : lastbyte = 15; break;
            case 'G' : lastbyte = 16; break;
      }
  }

  if(nBytesRead2==1)                  //last byte block
  {
        fileSink.Write(m_byte, lastbyte);  //Write
  }
  else
  {
        fileSink.Write(m_byte, filewrite);  //Write
  }

      nBytesRead = nBytesRead2;          //Transfer Byte
      for(int n=0; n<fileread; n++)
      {
```

```
                    m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();              //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
    }

}

////////////

void tab1::tea_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)           //8 key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       TEAEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                          //calculate about progress
        if((countstep > step) && (step > 0))
        {
```

```
            m_progress_crypt.StepIt();
            countstep = 0;
        }

        if(nBytesRead!=fileread)
        {
            for(int n=nBytesRead; n<fileread; n++)
            {
                m_byte[n]=' ';
            }
            switch(nBytesRead)                  //Mem Last Byte Block
            {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
            }
        }
            algorE.ProcessBlock(m_byte);

      if(nBytesRead != 0)
            {
                fileSink.Write(m_byte, filewrite);  //Write
            }

      )while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write

      //Encryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();       //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                            MB_OK/MB_ICONINFORMATION);
  }

}

void tab1::tea_D_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
```

```
{
      CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                      CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Decryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
      byte m_byte[8];
      byte m_byte2[8];
   UINT nBytesRead;
      UINT nBytesRead2;
      int lastbyte;
   byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
      for(int m=0; m<16; m++)
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      TEADecryption algorD(key);

      step = (filesize / fileread) / 21;  //calculate step progress


      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

      do
      {

      countstep++;

       if((countstep > step) && (step > 0))                    //calculate
about progress
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
                  case '2' : lastbyte = 2; break;
                  case '3' : lastbyte = 3; break;
                  case '4' : lastbyte = 4; break;
                  case '5' : lastbyte = 5; break;
                  case '6' : lastbyte = 6; break;
                  case '7' : lastbyte = 7; break;
                  case '8' : lastbyte = 8; break;
                  case '9' : lastbyte = 9; break;
```

```
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;         //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();               //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
    }

}

////////////

void tab1::threeway_N_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==12)
    {
        CStdioFile fileSource(m_source,
                            CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                          CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 12;
        int filewrite = 12;
        int countstep = 0;
        int step = 0;
      UINT nBytesRead;
```

```
   byte m_byte[12];
   char lastbyte = 'C';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00};

   for(int m=0; m<12; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   ThreeWayEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                          //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
                m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);


   if(nBytesRead != 0)
          {
                fileSink.Write(m_byte, filewrite);  //Write
          }

   }while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write
```

```
        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
        m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
    }

}

void tab1::threeway_D_crypt()
{
        UpdateData(TRUE);

    if(m_password.GetLength()==12)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 12;
        int filewrite = 12;
        int countstep = 0;
        int step = 0;
        byte m_byte[12];
        byte m_byte2[12];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00};
        for(int m=0; m<12; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        ThreeWayDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
         {
                m_progress_crypt.StepIt();
```

```
            countstep = 0;
        }

        algorD.ProcessBlock(m_byte);

        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                case '1' : lastbyte = 1; break;
                case '2' : lastbyte = 2; break;
                case '3' : lastbyte = 3; break;
                case '4' : lastbyte = 4; break;
                case '5' : lastbyte = 5; break;
                case '6' : lastbyte = 6; break;
                case '7' : lastbyte = 7; break;
                case '8' : lastbyte = 8; break;
                case '9' : lastbyte = 9; break;
                case 'A' : lastbyte = 10; break;
                case 'B' : lastbyte = 11; break;
                case 'C' : lastbyte = 12; break;
                case 'D' : lastbyte = 13; break;
                case 'E' : lastbyte = 14; break;
                case 'F' : lastbyte = 15; break;
                case 'G' : lastbyte = 16; break;
            }
        }

        if(nBytesRead2==1)                  //last byte block
        {
            fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
            fileSink.Write(m_byte, filewrite);  //Write
        }

            nBytesRead = nBytesRead2;       //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                m_byte[n] = m_byte2[n];
            }

      }while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();             //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                            MB_OK/MB_ICONINFORMATION);
  }

}

//////////////
```

```
void tab1::twofish_N_crypt()
{
        UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                            CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                          CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 16;
      int filewrite = 16;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[16];
      char lastbyte = 'G';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

      for(int m=0; m<16; m++)          //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      TwofishEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


      countstep++;                        //calculate about progress
       if((countstep > step) && (step > 0))
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       if(nBytesRead!=fileread)
       {
            for(int n=nBytesRead; n<fileread; n++)
            {
                  m_byte[n]=' ';
            }
            switch(nBytesRead)                //Mem Last Byte Block
            {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
                  case 6 : lastbyte = '6'; break;
                  case 7 : lastbyte = '7'; break;
                  case 8 : lastbyte = '8'; break;
```

```
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

              m_password.Empty();       //Delete password
              UpdateData(FALSE);
            m_progress_crypt.SetPos(0); //set step progress at 0
        }
        else
        {
              MessageBox("Password not Support", "Password",
                                  MB_OK/MB_ICONINFORMATION);
        }

}

void tab1::twofish_D_crypt()
{
        UpdateData(TRUE);

        if(m_password.GetLength()==16)
        {
            CStdioFile fileSource(m_source,
                                  CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                                CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 16;
            int filewrite = 16;
            int countstep = 0;
            int step = 0;
            byte m_byte[16];
            byte m_byte2[16];
          UINT nBytesRead;
            UINT nBytesRead2;
            int lastbyte;
          byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00);
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        TwofishDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                              //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                   case '1' : lastbyte = 1; break;
                   case '2' : lastbyte = 2; break;
                   case '3' : lastbyte = 3; break;
                   case '4' : lastbyte = 4; break;
                   case '5' : lastbyte = 5; break;
                   case '6' : lastbyte = 6; break;
                   case '7' : lastbyte = 7; break;
                   case '8' : lastbyte = 8; break;
                   case '9' : lastbyte = 9; break;
                   case 'A' : lastbyte = 10; break;
                   case 'B' : lastbyte = 11; break;
                   case 'C' : lastbyte = 12; break;
                   case 'D' : lastbyte = 13; break;
                   case 'E' : lastbyte = 14; break;
                   case 'F' : lastbyte = 15; break;
                   case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(m_byte, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
```

```
            {
                    m_byte[n] = m_byte2[n];
            }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();              //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

   }

}



/////Public Key///////

void tab1::rsa_N_crypt()
{
        UpdateData(TRUE);

   if( m_password.Find("_pb.rsa") != -1 )
   {
        CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                      CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 128;
        int countstep = 0;
        int step = 0;
        char *ciphertext;
     UINT nBytesRead;
        byte m_byte[16];
        char lastbyte = 'G';

        step = (filesize / fileread) / 21;  //calculate step progress

        do
        {
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


        countstep++;                          //calculate about progress
         if((countstep > step) && (step > 0))
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }
```

```
        if(nBytesRead!=fileread)
        {
              for(int n=nBytesRead; n<fileread; n++)
              {
                    m_byte[n]=' ';
              }
              switch(nBytesRead)                    //Mem Last Byte Block
              {
                    case 1 : lastbyte = '1'; break;
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
              }
        }

              ciphertext = RSAEncryptString(m_password, "seed", (char*)m_byte);

        if(nBytesRead != 0)
              {
                    fileSink.Write(ciphertext, filewrite);  //Write
              }

        }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

void tab1::rsa_D_crypt()
{
        UpdateData(TRUE);

  if( m_password.Find("_pv.rsa") != -1 )
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
```

```
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 128;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        char* decrypted;
        byte m_byte[128];
        byte m_byte2[128];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }

        decrypted = RSADecryptString(m_password, (char*)m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                  //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                  fileSink.Write(decrypted, lastbyte);  //Write
```

```
        }
        else
        {
                fileSink.Write(decrypted, filewrite);  //Write
        }

             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                  m_byte[n] = m_byte2[n];
             }

       }while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

         m_password.Empty();              //Delete password
         UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

void tab1::luc_N_crypt()
{
       UpdateData(TRUE);

  if( m_password.Find("_pb.luc") != -1 )
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 128;
       int countstep = 0;
       int step = 0;
       char *ciphertext;
    UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                          //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
```

```
            countstep = 0;
        }

        if(nBytesRead!=fileread)
        {
            for(int n=nBytesRead; n<fileread; n++)
            {
                m_byte[n]=' ';
            }
            switch(nBytesRead)                  //Mem Last Byte Block
            {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
            }
        }
            ciphertext = LUCEncryptString(m_password, "seed", (char*)m_byte);

        if(nBytesRead != 0)
            {
                fileSink.Write(ciphertext, filewrite);  //Write
            }

        }while(nBytesRead==fileread);

                fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();       //Delete password
          UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {       MessageBox("Password not Support", "Password",
                            MB_OK/MB_ICONINFORMATION);
    }

}

void tab1::luc_D_crypt()
{
        UpdateData(TRUE);

    if( m_password.Find("_pv.luc") != -1 )
    {
        CStdioFile fileSource(m_source,
```

```
                        CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 128;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        char* decrypted;
        byte m_byte[128];
        byte m_byte2[128];
    UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }

        decrypted = LUCDecryptString(m_password, (char*)m_byte);


          nBytesRead2 = fileSource.Read(m_byte2, fileread);

          if(nBytesRead2==1)
          {
               switch(m_byte2[0])                    //Mem Last Byte Block
               {
                     case '1' : lastbyte = 1; break;
                     case '2' : lastbyte = 2; break;
                     case '3' : lastbyte = 3; break;
                     case '4' : lastbyte = 4; break;
                     case '5' : lastbyte = 5; break;
                     case '6' : lastbyte = 6; break;
                     case '7' : lastbyte = 7; break;
                     case '8' : lastbyte = 8; break;
                     case '9' : lastbyte = 9; break;
                     case 'A' : lastbyte = 10; break;
                     case 'B' : lastbyte = 11; break;
                     case 'C' : lastbyte = 12; break;
                     case 'D' : lastbyte = 13; break;
                     case 'E' : lastbyte = 14; break;
                     case 'F' : lastbyte = 15; break;
                     case 'G' : lastbyte = 16; break;
               }
          }
```

```
        if(nBytesRead2==1)              //last byte block
        {
                fileSink.Write(decrypted, lastbyte);  //Write
        }
        else
        {
                fileSink.Write(decrypted, filewrite);  //Write
        }

             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                   m_byte[n] = m_byte2[n];
             }

       )while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

         m_password.Empty();                  //Delete password
         UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);

  }

}

void tab1::rabin_N_crypt()
{
        UpdateData(TRUE);

  if( m_password.Find("_pb.rbn") != -1 )
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 128;
       int countstep = 0;
       int step = 0;
       char *ciphertext;
    UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                         //calculate about progress
```

```
        if((countstep > step) && (step > 0))
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }

        if(nBytesRead!=fileread)
        {
              for(int n=nBytesRead; n<fileread; n++)
              {
                    m_byte[n]=' ';
              }
              switch(nBytesRead)                    //Mem Last Byte Block
              {
                    case 1 : lastbyte = '1'; break;
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
              }
        }
              ciphertext = RabinEncryptString(m_password, "seed", (char*)
m_byte);

      if(nBytesRead != 0)
              {
                    fileSink.Write(ciphertext, filewrite);  //Write
              }

      )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

      //Encryption;

      fileSource.Close();
      fileSink.Close();

        m_password.Empty();        //Delete password
        UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

void tab1::rabin_D_crypt()
{
        UpdateData(TRUE);
```

```
  if( m_password.Find("_pv.rbn") != -1 )
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 128;
       int filewrite = 16;
       int countstep = 0;
       int step = 0;
       char* decrypted;
       byte m_byte[128];
       byte m_byte2[128];
     UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }

       decrypted = RabinDecryptString(m_password, (char*)m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                  //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
```

```
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(decrypted, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(decrypted, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                      m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

          m_password.Empty();                //Delete password
          UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {
          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);

    }

}

/////////////Function Crypt Algorithm/////////////


/////////Code for Crypt String [Public Key]//////

char* tab1::RSAEncryptString(const char *pubFilename, const char *seed, const
char *message)
{
      FileSource pubFile(pubFilename, true, new HexDecoder);
      RSAES_OAEP_SHA_Encryptor pub(pubFile);

      if (strlen(message) > pub.MaxPlainTextLength())
      {
            cerr << "message too long for this key\n";
            abort();
      }

      RandomPool randPool;
      randPool.Put((byte *)seed, strlen(seed));

      char *outstr = new char[2*pub.CipherTextLength()+1];
      pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

      HexEncoder hexEncoder;
      hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
      hexEncoder.Close();
      hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

      outstr[2*pub.CipherTextLength()] = 0;
```

```
	return outstr;
}

char* tab1::RSADecryptString(const char *privFilename, const char *ciphertext)
{
	FileSource privFile(privFilename, true, new HexDecoder);
	RSAES_OAEP_SHA_Decryptor priv(privFile);

	HexDecoder hexDecoder;
	hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
	hexDecoder.Close();
	SecByteBlock buf(priv.CipherTextLength());
	hexDecoder.Get(buf, priv.CipherTextLength());

	char *outstr = new char[priv.MaxPlainTextLength()+1];
	unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
	outstr[messageLength] = 0;
	return outstr;
}

char* tab1::LUCEncryptString(const char *pubFilename, const char *seed, const
char *message)
{
	FileSource pubFile(pubFilename, true, new HexDecoder);
	LUCES_OAEP_SHA_Encryptor pub(pubFile);

	if (strlen(message) > pub.MaxPlainTextLength())
	{
		cerr << "message too long for this key\n";
		abort();
	}

	RandomPool randPool;
	randPool.Put((byte *)seed, strlen(seed));

	char *outstr = new char[2*pub.CipherTextLength()+1];
	pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

	HexEncoder hexEncoder;
	hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
	hexEncoder.Close();
	hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

	outstr[2*pub.CipherTextLength()] = 0;
	return outstr;
}

char* tab1::LUCDecryptString(const char *privFilename, const char *ciphertext)
{
	FileSource privFile(privFilename, true, new HexDecoder);
	LUCES_OAEP_SHA_Decryptor priv(privFile);

	HexDecoder hexDecoder;
	hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
	hexDecoder.Close();
	SecByteBlock buf(priv.CipherTextLength());
	hexDecoder.Get(buf, priv.CipherTextLength());

	char *outstr = new char[priv.MaxPlainTextLength()+1];
	unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
	outstr[messageLength] = 0;
	return outstr;
}
```

```
char* tab1::RabinEncryptString(const char *pubFilename, const char *seed,
const char *message)
{
	FileSource pubFile(pubFilename, true, new HexDecoder);
	RabinEncryptor pub(pubFile);

	if (strlen(message) > pub.MaxPlainTextLength())
	{
		cerr << "message too long for this key\n";
		abort();
	}

	RandomPool randPool;
	randPool.Put((byte *)seed, strlen(seed));

	char *outstr = new char[2*pub.CipherTextLength()+1];
	pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

	HexEncoder hexEncoder;
	hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
	hexEncoder.Close();
	hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

	outstr[2*pub.CipherTextLength()] = 0;
	return outstr;
}

char* tab1::RabinDecryptString(const char *privFilename, const char
*ciphertext)
{
	FileSource privFile(privFilename, true, new HexDecoder);
	RabinDecryptor priv(privFile);

	HexDecoder hexDecoder;
	hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
	hexDecoder.Close();
	SecByteBlock buf(priv.CipherTextLength());
	hexDecoder.Get(buf, priv.CipherTextLength());

	char *outstr = new char[priv.MaxPlainTextLength()+1];
	unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
	outstr[messageLength] = 0;
	return outstr;
}

/////////Code for Crypt String [Public Key]//////



void tab1::OnSelchangeComboAlgor()
{
	// TODO: Add your control notification handler code here
	 UpdateData(TRUE);
		m_key.Empty();
		m_password.Empty();
	switch(m_algor)
	{
	case 0 :{m_key += " 8 Character";
			m_max.SetLimitText(8);
			}break;
	case 1 :{m_key += "16 Character";
			m_max.SetLimitText(16);
			}break;
	case 2 :{m_key += "16 Character";
```

```
                    m_max.SetLimitText(16);
                    }break;
        case 3 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 4 :{m_key += "32 Character";
                    m_max.SetLimitText(32);
                    }break;
        case 5 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 6 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 7 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 8 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 9 :{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 10:{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 11:{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;
        case 12:{m_key += "12 Character";
                    m_max.SetLimitText(12);
                    }break;
        case 13:{m_key += "16 Character";
                    m_max.SetLimitText(16);
                    }break;

        case 14:{m_key += "Click Button to Select Key File";
                    m_max.SetLimitText(32);
                    }break;
        case 15:{m_key += "Click Button to Select Key File";
                    m_max.SetLimitText(32);
                    }break;
        case 16:{m_key += "Click Button to Select Key File";
                    m_max.SetLimitText(32);
                    }break;
        }

         UpdateData(FALSE);

}
void tab1::OnButtonCrypt()
{
        // TODO: Add your control notification handler code here
        CButton *Encryptbutton, *Decryptbutton;
        Encryptbutton = (CButton*)GetDlgItem(IDC_RADIO_ENCRYPT);
        Decryptbutton = (CButton*)GetDlgItem(IDC_RADIO_DECRYPT);

if((Encryptbutton->GetCheck())==1)
        {
        m_animate.Play(0, -1, -1);
        m_destination = "buffer.dat";           //About File

         UpdateData(TRUE);
```

```
        switch(m_algor)
        {
              case 0 :des_N_crypt();                    break;
              case 1 :idea_N_crypt();                   break;
              case 2 :blowfish_N_crypt();               break;
              case 3 :cast128_N_crypt();          break;
              case 4 :gost_N_crypt();                   break;
              case 5 :mars_N_crypt();                   break;
              case 6 :rc2_N_crypt();                    break;
              case 7 :rijndael_N_crypt();               break;
              case 8 :serpent_N_crypt();          break;
              case 9 :shark_N_crypt();            break;
              case 10:square_N_crypt();           break;
              case 11:tea_N_crypt();                    break;
              case 12:threeway_N_crypt();               break;
              case 13:twofish_N_crypt();          break;

              case 14:rsa_N_crypt();                    break;
              case 15:luc_N_crypt();                    break;
              case 16:rabin_N_crypt();            break;
        }

        CFileStatus status;
        CFile::GetStatus( m_destination, status);

        if(status.m_size > 0)
        {
        CFile::Remove( m_source );
        CFile::Rename( m_destination, m_source);
        MessageBox("Encrypt Complete", "Finish", MB_OK);
        }
        m_animate.Stop();
        }

if((Decryptbutton->GetCheck())==1)
        {
        m_animate.Play(0, -1, -1);
        m_destination = "buffer.dat";             //About File

         UpdateData(TRUE);

        switch(m_algor)
        {
              case 0 :des_D_crypt();                    break;
              case 1 :idea_D_crypt();                   break;
              case 2 :blowfish_D_crypt();               break;
              case 3 :cast128_D_crypt();          break;
              case 4 :gost_D_crypt();                   break;
              case 5 :mars_D_crypt();                   break;
              case 6 :rc2_D_crypt();                    break;
              case 7 :rijndael_D_crypt();               break;
              case 8 :serpent_D_crypt();          break;
              case 9 :shark_D_crypt();            break;
              case 10:square_D_crypt();           break;
              case 11:tea_D_crypt();                    break;
              case 12:threeway_D_crypt();               break;
              case 13:twofish_D_crypt();          break;

              case 14:rsa_D_crypt();                    break;
              case 15:luc_D_crypt();                    break;
              case 16:rabin_D_crypt();            break;
        }

        CFileStatus status;
        CFile::GetStatus( m_destination, status);
```

```
        if(status.m_size > 0)
        {
        CFile::Remove( m_source );
        CFile::Rename( m_destination, m_source);
        MessageBox("Decrypt Complete", "Finish", MB_OK);
        }
        m_animate.Stop();
        }

}

void tab1::OnMaxtextEditPassword()
{
        // TODO: Add your control notification handler code here
        MessageBox("Can't enter over Max", "Max", MB_OK);
}
```

```
#if !defined(AFX_TAB2_H__A10589F7_FA30_47DB_A33D_A5313ECC4101__INCLUDED_)
#define AFX_TAB2_H__A10589F7_FA30_47DB_A33D_A5313ECC4101__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// tab2.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// tab2 dialog

class tab2 : public CPropertyPage
{
	DECLARE_DYNCREATE(tab2)

// Construction
public:
	tab2();
	~tab2();

// Dialog Data
	//{{AFX_DATA(tab2)
	enum { IDD = IDD_DIALOG_TEST };
	CListBox	m_list_algorsel;
	CListBox	m_list_algor;
	CProgressCtrl	m_progress_crypt;
	CAnimateCtrl	m_animate;
	CString	m_source;
	CString	m_destination;
	CString	m_key;
	CString	m_password;
	//}}AFX_DATA



// Overrides
	// ClassWizard generate virtual function overrides
	//{{AFX_VIRTUAL(tab2)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual BOOL OnInitDialog(); //*

protected:
	HICON m_hIcon;		//*
	// Generated message map functions
	//{{AFX_MSG(tab2)
	afx_msg void OnButtonSource();
	afx_msg void OnDblclkListAlgor();
	afx_msg void OnDblclkListAlgorsel();
	afx_msg void OnButtonWeight();
	afx_msg void OnButtonTest();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

	//Symmetic
	void des_N_crypt();
	void des_D_crypt();
	void idea_N_crypt();
	void idea_D_crypt();
	void blowfish_N_crypt();
```

```
        void blowfish_D_crypt();
        void cast128_N_crypt();
        void cast128_D_crypt();
        void gost_N_crypt();
        void gost_D_crypt();
        void mars_N_crypt();
        void mars_D_crypt();
        void rc2_N_crypt();
        void rc2_D_crypt();
        void rijndael_N_crypt();
        void rijndael_D_crypt();
        void serpent_N_crypt();
        void serpent_D_crypt();
        void shark_N_crypt();
        void shark_D_crypt();
        void square_N_crypt();
        void square_D_crypt();
        void tea_N_crypt();
        void tea_D_crypt();
        void threeway_N_crypt();
        void threeway_D_crypt();
        void twofish_N_crypt();
        void twofish_D_crypt();

        //Public
        void rsa_N_crypt();
        void rsa_D_crypt();
        void luc_N_crypt();
        void luc_D_crypt();
        void rabin_N_crypt();
        void rabin_D_crypt();



        //Code Crypt String [For Pubilc]
        char* RSAEncryptString(const char *pubFilename, const char *seed, const
char *message);
        char* RSADecryptString(const char *privFilename, const char
*ciphertext);
        char* LUCEncryptString(const char *pubFilename, const char *seed, const
char *message);
        char* LUCDecryptString(const char *privFilename, const char
*ciphertext);
        char* RabinEncryptString(const char *pubFilename, const char *seed,
const char *message);
        char* RabinDecryptString(const char *privFilename, const char
*ciphertext);

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_TAB2_H__A10589F7_FA30_47DB_A33D_A5313ECC4101__INCLUDED_)
```

```
// tab2.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "tab2.h"

//#include "GenkeyDlg.h"                    //include for New Dialog
#include "WeightDlg.h"                      //include for New Dialog
#include "GraphtestDlg.h"

#include "globals.h"

//////////////spacial include////////////////////
#include "x_include\des.h"
#include "x_include\idea.h"
#include "x_include\blowfish.h"
#include "x_include\cast.h"
#include "x_include\gost.h"
#include "x_include\mars.h"
#include "x_include\rc2.h"
#include "x_include\rijndael.h"
#include "x_include\serpent.h"
#include "x_include\shark.h"
#include "x_include\square.h"
#include "x_include\tea.h"
#include "x_include\3way.h"
#include "x_include\twofish.h"

#include "x_include\rsa.h"
#include "x_include\luc.h"
#include "x_include\rabin.h"
#include "x_include\randpool.h"
#include "x_include\files.h"
#include "x_include\hex.h"

//////////////////////
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <string.h>
#include <math.h>
//////////////////////


//#include "Afxdisp.h"

USING_NAMESPACE(CryptoPP)
USING_NAMESPACE(std)
```

```
//////////////spacial include//////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// tab2 property page

IMPLEMENT_DYNCREATE(tab2, CPropertyPage)

tab2::tab2() : CPropertyPage(tab2::IDD)
{
	//{{AFX_DATA_INIT(tab2)
	m_source = _T("");
	m_key = _T("");
	//}}AFX_DATA_INIT
}

tab2::~tab2()
{
}

void tab2::DoDataExchange(CDataExchange* pDX)
{
	CPropertyPage::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(tab2)
	DDX_Control(pDX, IDC_LIST_ALGORSEL, m_list_algorsel);
	DDX_Control(pDX, IDC_LIST_ALGOR, m_list_algor);
	DDX_Control(pDX, IDC_PROGRESS_CRYPT, m_progress_crypt);
	DDX_Control(pDX, IDC_ANIMATE, m_animate);
	DDX_Text(pDX, IDC_EDIT_SOURCE, m_source);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(tab2, CPropertyPage)
	//{{AFX_MSG_MAP(tab2)
	ON_BN_CLICKED(IDC_BUTTON_SOURCE, OnButtonSource)
	ON_LBN_DBLCLK(IDC_LIST_ALGOR, OnDblclkListAlgor)
	ON_LBN_DBLCLK(IDC_LIST_ALGORSEL, OnDblclkListAlgorsel)
	ON_BN_CLICKED(IDC_BUTTON_WEIGHT, OnButtonWeight)
	ON_BN_CLICKED(IDC_BUTTON_TEST, OnButtonTest)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// tab2 message handlers
BOOL tab2::OnInitDialog()
{
	CancelToClose();

//	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	m_source="Select Source FilE->";
	m_key="Password";
```

```
//     m_destination="Select Dest->";
    UpdateData(FALSE);

       m_progress_crypt.SetRange(0, 100);
       m_progress_crypt.SetStep(5);
       m_progress_crypt.SetPos(0);

       m_animate.Open(IDR_AVI);

////////add string
       m_list_algor.AddString("DES");
       m_list_algor.AddString("IDEA");
       m_list_algor.AddString("Blowfish");
       m_list_algor.AddString("CAST128");
       m_list_algor.AddString("GOST");
       m_list_algor.AddString("MARS");
       m_list_algor.AddString("RC2");
       m_list_algor.AddString("Rijndael");
       m_list_algor.AddString("Serpent");
       m_list_algor.AddString("SHARK");
       m_list_algor.AddString("Square");
       m_list_algor.AddString("TEA");
       m_list_algor.AddString("ThreeWay");
       m_list_algor.AddString("Twofish");
       m_list_algor.AddString("RSA<pb>");
       m_list_algor.AddString("LUC<pb>");
       m_list_algor.AddString("Rabin<pb>");

////////add string

       return CPropertyPage::OnInitDialog();
}

////////////Function Crypt Algorithm////////////
void tab2::des_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==8)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<8; m++)              //8 key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       DESEncryption algorE(key);
```

```
        step = (filesize / fileread) / 21;  //calculate step progress

        do
        {
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


        countstep++;                          //calculate about progress
         if((countstep > step) && (step > 0))
         {
               m_progress_crypt.StepIt();
               countstep = 0;
         }


         if(nBytesRead!=fileread)
         {
               for(int n=nBytesRead; n<fileread; n++)
               {
                     m_byte[n]=' ';
               }
               switch(nBytesRead)                    //Mem Last Byte Block
               {
                     case 1 : lastbyte = '1'; break;
                     case 2 : lastbyte = '2'; break;
                     case 3 : lastbyte = '3'; break;
                     case 4 : lastbyte = '4'; break;
                     case 5 : lastbyte = '5'; break;
                     case 6 : lastbyte = '6'; break;
                     case 7 : lastbyte = '7'; break;
                     case 8 : lastbyte = '8'; break;
                     case 9 : lastbyte = '9'; break;
                     case 10: lastbyte = 'A'; break;
                     case 11: lastbyte = 'B'; break;
                     case 12: lastbyte = 'C'; break;
                     case 13: lastbyte = 'D'; break;
                     case 14: lastbyte = 'E'; break;
                     case 15: lastbyte = 'F'; break;
                     case 16: lastbyte = 'G'; break;
               }
         }

               algorE.ProcessBlock(m_byte);


        if(nBytesRead != 0)
               {
                     fileSink.Write(m_byte, filewrite);  //Write
               }

        }while(nBytesRead==fileread);

                     fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//       m_password.Empty();        //Delete password
//       UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
```

```
    {           MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
    }
}

void tab2::des_D_crypt()
{
  if(m_password.GetLength()==8)
  {
       CStdioFile fileSource(m_source,
                            CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                          CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
       byte m_byte[8];
       byte m_byte2[8];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
       for(int m=0; m<8; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       DESDecryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
```

```
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

           }while(nBytesRead==fileread);
            //Decryption;

           fileSource.Close();
           fileSink.Close();

//         m_password.Empty();              //Delete password
//         UpdateData(FALSE);
         m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

////////////

void tab2::idea_N_crypt()
{
//       UpdateData(TRUE);

    if(m_password.GetLength()==16) //key
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
```

```
   //Encryption;

   int filesize = fileSource.GetLength(); //about progress
   int fileread  = 8;  //block
   int filewrite = 8;  //block
   int countstep = 0;
   int step = 0;
UINT nBytesRead;
   byte m_byte[8];      //block
   char lastbyte = '8';       //block
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  //key
              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   IDEAEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                      //calculate about progress
    if((countstep > step) && (step > 0))
    {
         m_progress_crypt.StepIt();
         countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
         for(int n=nBytesRead; n<fileread; n++)
         {
              m_byte[n]=' ';
         }
         switch(nBytesRead)                //Mem Last Byte Block
         {
              case 1 : lastbyte = '1'; break;
              case 2 : lastbyte = '2'; break;
              case 3 : lastbyte = '3'; break;
              case 4 : lastbyte = '4'; break;
              case 5 : lastbyte = '5'; break;
              case 6 : lastbyte = '6'; break;
              case 7 : lastbyte = '7'; break;
              case 8 : lastbyte = '8'; break;
              case 9 : lastbyte = '9'; break;
              case 10: lastbyte = 'A'; break;
              case 11: lastbyte = 'B'; break;
              case 12: lastbyte = 'C'; break;
              case 13: lastbyte = 'D'; break;
              case 14: lastbyte = 'E'; break;
              case 15: lastbyte = 'F'; break;
              case 16: lastbyte = 'G'; break;
         }
    }

         algorE.ProcessBlock(m_byte);
```

```
        if(nBytesRead != 0)
                {
                        fileSink.Write(m_byte, filewrite);  //Write
                }

        )while(nBytesRead==fileread);

                        fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

   }

}

void tab2::idea_D_crypt()
{

   if(m_password.GetLength()==16)
   {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        IDEADecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {
```

```
        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                   case '1' : lastbyte = 1; break;
                   case '2' : lastbyte = 2; break;
                   case '3' : lastbyte = 3; break;
                   case '4' : lastbyte = 4; break;
                   case '5' : lastbyte = 5; break;
                   case '6' : lastbyte = 6; break;
                   case '7' : lastbyte = 7; break;
                   case '8' : lastbyte = 8; break;
                   case '9' : lastbyte = 9; break;
                   case 'A' : lastbyte = 10; break;
                   case 'B' : lastbyte = 11; break;
                   case 'C' : lastbyte = 12; break;
                   case 'D' : lastbyte = 13; break;
                   case 'E' : lastbyte = 14; break;
                   case 'F' : lastbyte = 15; break;
                   case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                    //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(m_byte, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                   m_byte[n] = m_byte2[n];
              }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();                    //Delete password
//        UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
   else
```

```
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
  }

}

////////////

void tab2::blowfish_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 8;
      int filewrite = 8;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[8];
      char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

      for(int m=0; m<16; m++)              //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      BlowfishEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


      countstep++;                        //calculate about progress
       if((countstep > step) && (step > 0))
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       if(nBytesRead!=fileread)
       {
            for(int n=nBytesRead; n<fileread; n++)
            {
                  m_byte[n]=' ';
            }
            switch(nBytesRead)                //Mem Last Byte Block
            {
                  case 1 : lastbyte = '1'; break;
```

```
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

//            m_password.Empty();         //Delete password
//            UpdateData(FALSE);
          m_progress_crypt.SetPos(0); //set step progress at 0
        }
        else
        {           MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

        }

}

void tab2::blowfish_D_crypt()
{

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
```

```
        byte m_byte[8];
        byte m_byte2[8];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
      byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        BlowfishDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }

         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)              //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
```

```
        {
                fileSink.Write(m_byte, filewrite);  //Write
        }

             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                  m_byte[n] = m_byte2[n];
             }

       }while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

  }

}

////////////

void tab2::cast128_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)          //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       CAST128Encryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
```

```
        {
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


        countstep++;                          //calculate about progress
         if((countstep > step) && (step > 0))
         {
               m_progress_crypt.StepIt();
               countstep = 0;
         }


         if(nBytesRead!=fileread)
         {
               for(int n=nBytesRead; n<fileread; n++)
               {
                     m_byte[n]=' ';
               }
               switch(nBytesRead)                    //Mem Last Byte Block
               {
                     case 1 : lastbyte = '1'; break;
                     case 2 : lastbyte = '2'; break;
                     case 3 : lastbyte = '3'; break;
                     case 4 : lastbyte = '4'; break;
                     case 5 : lastbyte = '5'; break;
                     case 6 : lastbyte = '6'; break;
                     case 7 : lastbyte = '7'; break;
                     case 8 : lastbyte = '8'; break;
                     case 9 : lastbyte = '9'; break;
                     case 10: lastbyte = 'A'; break;
                     case 11: lastbyte = 'B'; break;
                     case 12: lastbyte = 'C'; break;
                     case 13: lastbyte = 'D'; break;
                     case 14: lastbyte = 'E'; break;
                     case 15: lastbyte = 'F'; break;
                     case 16: lastbyte = 'G'; break;
               }
         }

               algorE.ProcessBlock(m_byte);


        if(nBytesRead != 0)
               {
                     fileSink.Write(m_byte, filewrite);  //Write
               }

        }while(nBytesRead==fileread);

                     fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();         //Delete password
//        UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }
```

```
}

void tab2::cast128_D_crypt()
{

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
       byte m_byte[8];
       byte m_byte2[8];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
       for(int m=0; m<16; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       CAST128Decryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
             switch(m_byte2[0])                  //Mem Last Byte Block
             {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
```

```
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)                  //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

           )while(nBytesRead==fileread);
            //Decryption;

           fileSource.Close();
           fileSink.Close();

//          m_password.Empty();                   //Delete password
//          UpdateData(FALSE);
         m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

      }

}

////////////

void tab2::gost_N_crypt()
{
//        UpdateData(TRUE);

  if(m_password.GetLength()==32)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;
```

```
   int filesize = fileSource.GetLength(); //about progress
   int fileread  = 8;
   int filewrite = 8;
   int countstep = 0;
   int step = 0;
UINT nBytesRead;
   byte m_byte[8];
   char lastbyte = '8';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<32; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   GOSTEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                        //calculate about progress
    if((countstep > step) && (step > 0))
    {
         m_progress_crypt.StepIt();
         countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
         for(int n=nBytesRead; n<fileread; n++)
         {
              m_byte[n]=' ';
         }
         switch(nBytesRead)                //Mem Last Byte Block
         {
               case 1 : lastbyte = '1'; break;
               case 2 : lastbyte = '2'; break;
               case 3 : lastbyte = '3'; break;
               case 4 : lastbyte = '4'; break;
               case 5 : lastbyte = '5'; break;
               case 6 : lastbyte = '6'; break;
               case 7 : lastbyte = '7'; break;
               case 8 : lastbyte = '8'; break;
               case 9 : lastbyte = '9'; break;
               case 10: lastbyte = 'A'; break;
               case 11: lastbyte = 'B'; break;
               case 12: lastbyte = 'C'; break;
               case 13: lastbyte = 'D'; break;
               case 14: lastbyte = 'E'; break;
               case 15: lastbyte = 'F'; break;
               case 16: lastbyte = 'G'; break;
         }
    }

         algorE.ProcessBlock(m_byte);
```

```
        if(nBytesRead != 0)
                {
                        fileSink.Write(m_byte, filewrite);  //Write
                }

        )while(nBytesRead==fileread);

                        fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

  }

}

void tab2::gost_D_crypt()
{

  if(m_password.GetLength()==32)
  {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 8;
        int filewrite = 8;
        int countstep = 0;
        int step = 0;
        byte m_byte[8];
        byte m_byte2[8];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<32; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        GOSTDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file
```

```
        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                   case '1' : lastbyte = 1; break;
                   case '2' : lastbyte = 2; break;
                   case '3' : lastbyte = 3; break;
                   case '4' : lastbyte = 4; break;
                   case '5' : lastbyte = 5; break;
                   case '6' : lastbyte = 6; break;
                   case '7' : lastbyte = 7; break;
                   case '8' : lastbyte = 8; break;
                   case '9' : lastbyte = 9; break;
                   case 'A' : lastbyte = 10; break;
                   case 'B' : lastbyte = 11; break;
                   case 'C' : lastbyte = 12; break;
                   case 'D' : lastbyte = 13; break;
                   case 'E' : lastbyte = 14; break;
                   case 'F' : lastbyte = 15; break;
                   case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)              //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(m_byte, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                    m_byte[n] = m_byte2[n];
              }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
```

```
  }
  else
  {         MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
  }

}

////////////

void tab2::mars_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                         CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 16;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)          //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       MARSEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                        //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                  //Mem Last Byte Block
```

```
            {
                    case 1 : lastbyte = '1'; break;
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
            }
        }

            algorE.ProcessBlock(m_byte);


        if(nBytesRead != 0)
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

        )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();       //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {           MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

void tab2::mars_D_crypt()
{

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
```

```
        int countstep = 0;
        int step = 0;
        byte m_byte[16];
        byte m_byte2[16];
    UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        MARSDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
            m_progress_crypt.StepIt();
            countstep = 0;
        }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
             switch(m_byte2[0])                  //Mem Last Byte Block
             {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
             }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
```

```
        }
        else
        {
                fileSink.Write(m_byte, filewrite);  //Write
        }

             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                   m_byte[n] = m_byte2[n];
             }

       }while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

  }

}

////////////

void tab2::rc2_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                       CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)           //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       RC2Encryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress
```

```
do
{
nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


countstep++;                          //calculate about progress
 if((countstep > step) && (step > 0))
 {
       m_progress_crypt.StepIt();
       countstep = 0;
 }


 if(nBytesRead!=fileread)
 {
       for(int n=nBytesRead; n<fileread; n++)
       {
             m_byte[n]=' ';
       }
       switch(nBytesRead)                    //Mem Last Byte Block
       {
             case 1 : lastbyte = '1'; break;
             case 2 : lastbyte = '2'; break;
             case 3 : lastbyte = '3'; break;
             case 4 : lastbyte = '4'; break;
             case 5 : lastbyte = '5'; break;
             case 6 : lastbyte = '6'; break;
             case 7 : lastbyte = '7'; break;
             case 8 : lastbyte = '8'; break;
             case 9 : lastbyte = '9'; break;
             case 10: lastbyte = 'A'; break;
             case 11: lastbyte = 'B'; break;
             case 12: lastbyte = 'C'; break;
             case 13: lastbyte = 'D'; break;
             case 14: lastbyte = 'E'; break;
             case 15: lastbyte = 'F'; break;
             case 16: lastbyte = 'G'; break;
       }
 }
       algorE.ProcessBlock(m_byte);


if(nBytesRead != 0)
       {
             fileSink.Write(m_byte, filewrite);  //Write
       }

}while(nBytesRead==fileread);

             fileSink.Write(&lastbyte, 1);  //Write

//Encryption;

fileSource.Close();
fileSink.Close();

//     m_password.Empty();       //Delete password
//     UpdateData(FALSE);
    m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {       MessageBox("Password not Support", "Password",
                          MB_OK/MB_ICONINFORMATION);
```

```
    }

}

void tab2::rc2_D_crypt()
{

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
       byte m_byte[8];
       byte m_byte2[8];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
       for(int m=0; m<16; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       RC2Decryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
             switch(m_byte2[0])                  //Mem Last Byte Block
             {
                  case '1' : lastbyte = 1; break;
```

```
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)                  //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();                  //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

    }

}

////////////

void tab2::rijndael_N_crypt()
{
//        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
```

```
   //Encryption;

   int filesize = fileSource.GetLength(); //about progress
   int fileread  = 16;
   int filewrite = 16;
   int countstep = 0;
   int step = 0;
UINT nBytesRead;
   byte m_byte[16];
   char lastbyte = 'G';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   RijndaelEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                        //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
                m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);
```

```
        if(nBytesRead != 0)
                {
                        fileSink.Write(m_byte, filewrite);  //Write
                }

        )while(nBytesRead==fileread);

                        fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

  }

}

void tab2::rijndael_D_crypt()
{

  if(m_password.GetLength()==16)
  {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        byte m_byte[16];
        byte m_byte2[16];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        RijndaelDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {
```

```
        countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
        }

        if(nBytesRead2==1)                  //last byte block
        {
                fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
                fileSink.Write(m_byte, filewrite);  //Write
        }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                    m_byte[n] = m_byte2[n];
              }

       )while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

//       m_password.Empty();                //Delete password
//       UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
```

```
    {           MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
    }

}

////////////

void tab2::serpent_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 16;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)              //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       SerpentEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                         //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                 //Mem Last Byte Block
             {
                  case 1 : lastbyte = '1'; break;
```

```
                    case 2 : lastbyte = '2'; break;
                    case 3 : lastbyte = '3'; break;
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

//          m_password.Empty();         //Delete password
//          UpdateData(FALSE);
          m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {         MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

      }

}

void tab2::serpent_D_crypt()
{

      if(m_password.GetLength()==16)
      {
            CStdioFile fileSource(m_source,
                                  CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                                CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 16;
            int filewrite = 16;
            int countstep = 0;
            int step = 0;
```

```
        byte m_byte[16];
        byte m_byte2[16];
    UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        SerpentDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }


         algorD.ProcessBlock(m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                  //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
```

```
        {
                fileSink.Write(m_byte, filewrite);  //Write
        }

             nBytesRead = nBytesRead2;          //Transfer Byte
             for(int n=0; n<fileread; n++)
             {
                   m_byte[n] = m_byte2[n];
             }

       }while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);

  }

}

////////////

void tab2::shark_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)          //key
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       SHARKEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
```

```
{
nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


countstep++;                          //calculate about progress
 if((countstep > step) && (step > 0))
 {
      m_progress_crypt.StepIt();
      countstep = 0;
 }


 if(nBytesRead!=fileread)
 {
      for(int n=nBytesRead; n<fileread; n++)
      {
           m_byte[n]=' ';
      }
      switch(nBytesRead)                    //Mem Last Byte Block
      {
           case 1 : lastbyte = '1'; break;
           case 2 : lastbyte = '2'; break;
           case 3 : lastbyte = '3'; break;
           case 4 : lastbyte = '4'; break;
           case 5 : lastbyte = '5'; break;
           case 6 : lastbyte = '6'; break;
           case 7 : lastbyte = '7'; break;
           case 8 : lastbyte = '8'; break;
           case 9 : lastbyte = '9'; break;
           case 10: lastbyte = 'A'; break;
           case 11: lastbyte = 'B'; break;
           case 12: lastbyte = 'C'; break;
           case 13: lastbyte = 'D'; break;
           case 14: lastbyte = 'E'; break;
           case 15: lastbyte = 'F'; break;
           case 16: lastbyte = 'G'; break;
      }
 }
      algorE.ProcessBlock(m_byte);


if(nBytesRead != 0)
      {
           fileSink.Write(m_byte, filewrite);  //Write
      }

}while(nBytesRead==fileread);

           fileSink.Write(&lastbyte, 1);  //Write

//Encryption;

fileSource.Close();
fileSink.Close();

//      m_password.Empty();       //Delete password
//      UpdateData(FALSE);
    m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {       MessageBox("Password not Support", "Password",
                          MB_OK/MB_ICONINFORMATION);

  }
```

```
}
void tab2::shark_D_crypt()
{
  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
       byte m_byte[8];
       byte m_byte2[8];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
       for(int m=0; m<16; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       SHARKDecryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
             switch(m_byte2[0])                  //Mem Last Byte Block
             {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
```

```
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;       //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();               //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

////////////

void tab2::square_N_crypt()
{
//        UpdateData(TRUE);

    if(m_password.GetLength()==16)
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Encryption;
```

```
   int filesize = fileSource.GetLength(); //about progress
   int fileread  = 16;
   int filewrite = 16;
   int countstep = 0;
   int step = 0;
UINT nBytesRead;
   byte m_byte[16];
   char lastbyte = 'G';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   SquareEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                        //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
                m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);


    if(nBytesRead != 0)
          {
```

```
                    fileSink.Write(m_byte, filewrite);  //Write
                }

        )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

   }

}

void tab2::square_D_crypt()
{

   if(m_password.GetLength()==16)
   {
        CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 16;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        byte m_byte[16];
        byte m_byte2[16];
     UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
     byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
        for(int m=0; m<16; m++)
        {
          key[m] = m_password.GetAt(m);     //Copy Key
        }

        SquareDecryption algorD(key);

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;
```

```
        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }

         algorD.ProcessBlock(m_byte);

         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                  //last byte block
         {
                fileSink.Write(m_byte, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(m_byte, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                    m_byte[n] = m_byte2[n];
              }

       }while(nBytesRead==fileread);
        //Decryption;

       fileSource.Close();
       fileSink.Close();

//       m_password.Empty();                  //Delete password
//       UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                  MB_OK/MB_ICONINFORMATION);
```

```
    }

}

////////////

void tab2::tea_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 8;
       int filewrite = 8;
       int countstep = 0;
       int step = 0;
    UINT nBytesRead;
       byte m_byte[8];
       char lastbyte = '8';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

       for(int m=0; m<16; m++)          //8 key
       {
         key[m] = m_password.GetAt(m);      //Copy Key
       }

       TEAEncryption algorE(key);

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                        //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                 //Mem Last Byte Block
             {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
```

```
                    case 4 : lastbyte = '4'; break;
                    case 5 : lastbyte = '5'; break;
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                algorE.ProcessBlock(m_byte);


            if(nBytesRead != 0)
                {
                    fileSink.Write(m_byte, filewrite);  //Write
                }

            )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

//          m_password.Empty();         //Delete password
//          UpdateData(FALSE);
          m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {
                MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);

      }

}

void tab2::tea_D_crypt()
{

      if(m_password.GetLength()==16)
      {
            CStdioFile fileSource(m_source,
                                CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                              CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 8;
            int filewrite = 8;
            int countstep = 0;
            int step = 0;
            byte m_byte[8];
            byte m_byte2[8];
```

```
    UINT nBytesRead;
      UINT nBytesRead2;
      int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
      for(int m=0; m<16; m++)
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      TEADecryption algorD(key);

      step = (filesize / fileread) / 21;  //calculate step progress


      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

      do
      {

      countstep++;

       if((countstep > step) && (step > 0))                    //calculate
about progress
       {
            m_progress_crypt.StepIt();
            countstep = 0;
       }


       algorD.ProcessBlock(m_byte);


       nBytesRead2 = fileSource.Read(m_byte2, fileread);

       if(nBytesRead2==1)
       {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
                  case '2' : lastbyte = 2; break;
                  case '3' : lastbyte = 3; break;
                  case '4' : lastbyte = 4; break;
                  case '5' : lastbyte = 5; break;
                  case '6' : lastbyte = 6; break;
                  case '7' : lastbyte = 7; break;
                  case '8' : lastbyte = 8; break;
                  case '9' : lastbyte = 9; break;
                  case 'A' : lastbyte = 10; break;
                  case 'B' : lastbyte = 11; break;
                  case 'C' : lastbyte = 12; break;
                  case 'D' : lastbyte = 13; break;
                  case 'E' : lastbyte = 14; break;
                  case 'F' : lastbyte = 15; break;
                  case 'G' : lastbyte = 16; break;
            }
       }

       if(nBytesRead2==1)                 //last byte block
       {
              fileSink.Write(m_byte, lastbyte);  //Write
       }
       else
       {
              fileSink.Write(m_byte, filewrite);  //Write
```

```
        }

            nBytesRead = nBytesRead2;          //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                  m_byte[n] = m_byte2[n];
            }

      }while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
  }

}

////////////

void tab2::threeway_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==12)
  {
      CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                      CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Encryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 12;
      int filewrite = 12;
      int countstep = 0;
      int step = 0;
    UINT nBytesRead;
      byte m_byte[12];
      char lastbyte = 'C';
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00};

      for(int m=0; m<12; m++)           //key
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      ThreeWayEncryption algorE(key);

      step = (filesize / fileread) / 21;  //calculate step progress

      do
      {
```

```
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        countstep++;                          //calculate about progress
         if((countstep > step) && (step > 0))
         {
               m_progress_crypt.StepIt();
               countstep = 0;
         }

         if(nBytesRead!=fileread)
         {
               for(int n=nBytesRead; n<fileread; n++)
               {
                     m_byte[n]=' ';
               }
               switch(nBytesRead)                    //Mem Last Byte Block
               {
                     case 1 : lastbyte = '1'; break;
                     case 2 : lastbyte = '2'; break;
                     case 3 : lastbyte = '3'; break;
                     case 4 : lastbyte = '4'; break;
                     case 5 : lastbyte = '5'; break;
                     case 6 : lastbyte = '6'; break;
                     case 7 : lastbyte = '7'; break;
                     case 8 : lastbyte = '8'; break;
                     case 9 : lastbyte = '9'; break;
                     case 10: lastbyte = 'A'; break;
                     case 11: lastbyte = 'B'; break;
                     case 12: lastbyte = 'C'; break;
                     case 13: lastbyte = 'D'; break;
                     case 14: lastbyte = 'E'; break;
                     case 15: lastbyte = 'F'; break;
                     case 16: lastbyte = 'G'; break;
               }
         }
               algorE.ProcessBlock(m_byte);

        if(nBytesRead != 0)
               {
                     fileSink.Write(m_byte, filewrite);  //Write
               }

        }while(nBytesRead==fileread);

                     fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();         //Delete password
//        UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
   }
   else
   {          MessageBox("Password not Support", "Password",
                                 MB_OK/MB_ICONINFORMATION);
   }
}
```

```
void tab2::threeway_D_crypt()
{

  if(m_password.GetLength()==12)
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Decryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 12;
       int filewrite = 12;
       int countstep = 0;
       int step = 0;
       byte m_byte[12];
       byte m_byte2[12];
    UINT nBytesRead;
       UINT nBytesRead2;
       int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                  0x00,0x00,0x00,0x00};
       for(int m=0; m<12; m++)
       {
         key[m] = m_password.GetAt(m);     //Copy Key
       }

       ThreeWayDecryption algorD(key);

       step = (filesize / fileread) / 21;  //calculate step progress


       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

       do
       {

       countstep++;

        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        algorD.ProcessBlock(m_byte);


        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
             switch(m_byte2[0])                  //Mem Last Byte Block
             {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
```

```
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
                }
            }

            if(nBytesRead2==1)              //last byte block
            {
                    fileSink.Write(m_byte, lastbyte);  //Write
            }
            else
            {
                    fileSink.Write(m_byte, filewrite);  //Write
            }

                nBytesRead = nBytesRead2;          //Transfer Byte
                for(int n=0; n<fileread; n++)
                {
                    m_byte[n] = m_byte2[n];
                }

           )while(nBytesRead==fileread);
            //Decryption;

           fileSource.Close();
           fileSink.Close();

//         m_password.Empty();               //Delete password
//         UpdateData(FALSE);
         m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
      }

}

////////////

void tab2::twofish_N_crypt()
{
//       UpdateData(TRUE);

  if(m_password.GetLength()==16)
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
```

```
   int fileread  = 16;
   int filewrite = 16;
   int countstep = 0;
   int step = 0;
UINT nBytesRead;
   byte m_byte[16];
   char lastbyte = 'G';
byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

   for(int m=0; m<16; m++)           //key
   {
     key[m] = m_password.GetAt(m);     //Copy Key
   }

   TwofishEncryption algorE(key);

   step = (filesize / fileread) / 21;  //calculate step progress

   do
   {
   nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


   countstep++;                        //calculate about progress
    if((countstep > step) && (step > 0))
    {
          m_progress_crypt.StepIt();
          countstep = 0;
    }


    if(nBytesRead!=fileread)
    {
          for(int n=nBytesRead; n<fileread; n++)
          {
                m_byte[n]=' ';
          }
          switch(nBytesRead)                //Mem Last Byte Block
          {
                case 1 : lastbyte = '1'; break;
                case 2 : lastbyte = '2'; break;
                case 3 : lastbyte = '3'; break;
                case 4 : lastbyte = '4'; break;
                case 5 : lastbyte = '5'; break;
                case 6 : lastbyte = '6'; break;
                case 7 : lastbyte = '7'; break;
                case 8 : lastbyte = '8'; break;
                case 9 : lastbyte = '9'; break;
                case 10: lastbyte = 'A'; break;
                case 11: lastbyte = 'B'; break;
                case 12: lastbyte = 'C'; break;
                case 13: lastbyte = 'D'; break;
                case 14: lastbyte = 'E'; break;
                case 15: lastbyte = 'F'; break;
                case 16: lastbyte = 'G'; break;
          }
    }

          algorE.ProcessBlock(m_byte);


   if(nBytesRead != 0)
          {
                fileSink.Write(m_byte, filewrite);  //Write
```

```
            }

        }while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

  }

}

void tab2::twofish_D_crypt()
{

  if(m_password.GetLength()==16)
  {
      CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

      CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

      //Decryption;

      int filesize = fileSource.GetLength(); //about progress
      int fileread  = 16;
      int filewrite = 16;
      int countstep = 0;
      int step = 0;
      byte m_byte[16];
      byte m_byte2[16];
    UINT nBytesRead;
      UINT nBytesRead2;
      int lastbyte;
    byte key[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
      for(int m=0; m<16; m++)
      {
        key[m] = m_password.GetAt(m);     //Copy Key
      }

      TwofishDecryption algorD(key);

      step = (filesize / fileread) / 21;  //calculate step progress


      nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

      do
      {

      countstep++;
```

```
        if((countstep > step) && (step > 0))                    //calculate
about progress
        {
              m_progress_crypt.StepIt();
              countstep = 0;
        }

        algorD.ProcessBlock(m_byte);

        nBytesRead2 = fileSource.Read(m_byte2, fileread);

        if(nBytesRead2==1)
        {
            switch(m_byte2[0])                  //Mem Last Byte Block
            {
                  case '1' : lastbyte = 1; break;
                  case '2' : lastbyte = 2; break;
                  case '3' : lastbyte = 3; break;
                  case '4' : lastbyte = 4; break;
                  case '5' : lastbyte = 5; break;
                  case '6' : lastbyte = 6; break;
                  case '7' : lastbyte = 7; break;
                  case '8' : lastbyte = 8; break;
                  case '9' : lastbyte = 9; break;
                  case 'A' : lastbyte = 10; break;
                  case 'B' : lastbyte = 11; break;
                  case 'C' : lastbyte = 12; break;
                  case 'D' : lastbyte = 13; break;
                  case 'E' : lastbyte = 14; break;
                  case 'F' : lastbyte = 15; break;
                  case 'G' : lastbyte = 16; break;
            }
        }

        if(nBytesRead2==1)              //last byte block
        {
              fileSink.Write(m_byte, lastbyte);  //Write
        }
        else
        {
              fileSink.Write(m_byte, filewrite);  //Write
        }

            nBytesRead = nBytesRead2;         //Transfer Byte
            for(int n=0; n<fileread; n++)
            {
                  m_byte[n] = m_byte2[n];
            }

      }while(nBytesRead==fileread);
       //Decryption;

      fileSource.Close();
      fileSink.Close();

//       m_password.Empty();              //Delete password
//       UpdateData(FALSE);
     m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }
```

```
}


/////Public Key////////

void tab2::rsa_N_crypt()
{
//       UpdateData(TRUE);

  if( m_password.Find("_pb.rsa") != -1 )
  {
       CStdioFile fileSource(m_source,
                             CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                           CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 128;
       int countstep = 0;
       int step = 0;
       char *ciphertext;
    UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                        //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                //Mem Last Byte Block
             {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
                  case 6 : lastbyte = '6'; break;
                  case 7 : lastbyte = '7'; break;
                  case 8 : lastbyte = '8'; break;
                  case 9 : lastbyte = '9'; break;
```

```
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
            }

                ciphertext = RSAEncryptString(m_password, "seed", (char*)m_byte);

            if(nBytesRead != 0)
                {
                    fileSink.Write(ciphertext, filewrite);  //Write
                }

            )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

            //Encryption;

            fileSource.Close();
            fileSink.Close();

//          m_password.Empty();         //Delete password
//          UpdateData(FALSE);
          m_progress_crypt.SetPos(0); //set step progress at 0
      }
      else
      {
                MessageBox("Password not Support", "Password",
                                    MB_OK/MB_ICONINFORMATION);
      }

}

void tab2::rsa_D_crypt()
{

      if( m_password.Find("_pv.rsa") != -1 )
      {
            CStdioFile fileSource(m_source,
                                  CFile::modeRead | CFile::typeBinary);

            CStdioFile fileSink(m_destination,
                                CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

            //Decryption;

            int filesize = fileSource.GetLength(); //about progress
            int fileread  = 128;
            int filewrite = 16;
            int countstep = 0;
            int step = 0;
            char* decrypted;
            byte m_byte[128];
            byte m_byte2[128];
        UINT nBytesRead;
            UINT nBytesRead2;
            int lastbyte;

            step = (filesize / fileread) / 21;  //calculate step progress
```

```
        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }

        decrypted = RSADecryptString(m_password, (char*)m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)                    //last byte block
         {
                fileSink.Write(decrypted, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(decrypted, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                    m_byte[n] = m_byte2[n];
              }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();                   //Delete password
```

```
//       UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                              MB_OK/MB_ICONINFORMATION);
  }

}

void tab2::luc_N_crypt()
{
//       UpdateData(TRUE);

  if( m_password.Find("_pb.luc") != -1 )
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                        CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 128;
       int countstep = 0;
       int step = 0;
       char *ciphertext;
     UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                          //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                  //Mem Last Byte Block
             {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
                  case 6 : lastbyte = '6'; break;
                  case 7 : lastbyte = '7'; break;
```

```
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
        }

                ciphertext = LUCEncryptString(m_password, "seed", (char*)m_byte);

        if(nBytesRead != 0)
                {
                    fileSink.Write(ciphertext, filewrite);  //Write
                }

        )while(nBytesRead==fileread);

                    fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();        //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

void tab2::luc_D_crypt()
{

    if( m_password.Find("_pv.luc") != -1 )
    {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 128;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        char* decrypted;
        byte m_byte[128];
        byte m_byte2[128];
      UINT nBytesRead;
        UINT nBytesRead2;
        int lastbyte;
```

```
        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }

        decrypted = LUCDecryptString(m_password, (char*)m_byte);


          nBytesRead2 = fileSource.Read(m_byte2, fileread);

          if(nBytesRead2==1)
          {
               switch(m_byte2[0])                    //Mem Last Byte Block
               {
                    case '1' : lastbyte = 1; break;
                    case '2' : lastbyte = 2; break;
                    case '3' : lastbyte = 3; break;
                    case '4' : lastbyte = 4; break;
                    case '5' : lastbyte = 5; break;
                    case '6' : lastbyte = 6; break;
                    case '7' : lastbyte = 7; break;
                    case '8' : lastbyte = 8; break;
                    case '9' : lastbyte = 9; break;
                    case 'A' : lastbyte = 10; break;
                    case 'B' : lastbyte = 11; break;
                    case 'C' : lastbyte = 12; break;
                    case 'D' : lastbyte = 13; break;
                    case 'E' : lastbyte = 14; break;
                    case 'F' : lastbyte = 15; break;
                    case 'G' : lastbyte = 16; break;
               }
          }

          if(nBytesRead2==1)                    //last byte block
          {
                  fileSink.Write(decrypted, lastbyte);  //Write
          }
          else
          {
                  fileSink.Write(decrypted, filewrite);  //Write
          }

               nBytesRead = nBytesRead2;          //Transfer Byte
               for(int n=0; n<fileread; n++)
               {
                    m_byte[n] = m_byte2[n];
               }

        }while(nBytesRead==fileread);
         //Decryption;

        fileSource.Close();
        fileSink.Close();
```

```
//        m_password.Empty();              //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);
  }

}

void tab2::rabin_N_crypt()
{
//        UpdateData(TRUE);

  if( m_password.Find("_pb.rbn") != -1 )
  {
       CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);

       CStdioFile fileSink(m_destination,
                       CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

       //Encryption;

       int filesize = fileSource.GetLength(); //about progress
       int fileread  = 16;
       int filewrite = 128;
       int countstep = 0;
       int step = 0;
       char *ciphertext;
     UINT nBytesRead;
       byte m_byte[16];
       char lastbyte = 'G';

       step = (filesize / fileread) / 21;  //calculate step progress

       do
       {
       nBytesRead = fileSource.Read(m_byte, fileread);  //Read file


       countstep++;                          //calculate about progress
        if((countstep > step) && (step > 0))
        {
             m_progress_crypt.StepIt();
             countstep = 0;
        }


        if(nBytesRead!=fileread)
        {
             for(int n=nBytesRead; n<fileread; n++)
             {
                  m_byte[n]=' ';
             }
             switch(nBytesRead)                  //Mem Last Byte Block
             {
                  case 1 : lastbyte = '1'; break;
                  case 2 : lastbyte = '2'; break;
                  case 3 : lastbyte = '3'; break;
                  case 4 : lastbyte = '4'; break;
                  case 5 : lastbyte = '5'; break;
```

```
                    case 6 : lastbyte = '6'; break;
                    case 7 : lastbyte = '7'; break;
                    case 8 : lastbyte = '8'; break;
                    case 9 : lastbyte = '9'; break;
                    case 10: lastbyte = 'A'; break;
                    case 11: lastbyte = 'B'; break;
                    case 12: lastbyte = 'C'; break;
                    case 13: lastbyte = 'D'; break;
                    case 14: lastbyte = 'E'; break;
                    case 15: lastbyte = 'F'; break;
                    case 16: lastbyte = 'G'; break;
                }
        }

                ciphertext = RabinEncryptString(m_password, "seed", (char*)
m_byte);

        if(nBytesRead != 0)
                {
                        fileSink.Write(ciphertext, filewrite);  //Write
                }

        }while(nBytesRead==fileread);

                        fileSink.Write(&lastbyte, 1);  //Write

        //Encryption;

        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();       //Delete password
//        UpdateData(FALSE);
      m_progress_crypt.SetPos(0); //set step progress at 0
  }
  else
  {         MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

  }

}

void tab2::rabin_D_crypt()
{

  if( m_password.Find("_pv.rbn") != -1 )
  {
        CStdioFile fileSource(m_source,
                              CFile::modeRead | CFile::typeBinary);

        CStdioFile fileSink(m_destination,
                            CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

        //Decryption;

        int filesize = fileSource.GetLength(); //about progress
        int fileread  = 128;
        int filewrite = 16;
        int countstep = 0;
        int step = 0;
        char* decrypted;
        byte m_byte[128];
        byte m_byte2[128];
    UINT nBytesRead;
```

```
        UINT nBytesRead2;
        int lastbyte;

        step = (filesize / fileread) / 21;  //calculate step progress


        nBytesRead = fileSource.Read(m_byte, fileread);  //Read file

        do
        {

        countstep++;

         if((countstep > step) && (step > 0))                          //calculate
about progress
         {
              m_progress_crypt.StepIt();
              countstep = 0;
         }

        decrypted = RabinDecryptString(m_password, (char*)m_byte);


         nBytesRead2 = fileSource.Read(m_byte2, fileread);

         if(nBytesRead2==1)
         {
              switch(m_byte2[0])                    //Mem Last Byte Block
              {
                   case '1' : lastbyte = 1; break;
                   case '2' : lastbyte = 2; break;
                   case '3' : lastbyte = 3; break;
                   case '4' : lastbyte = 4; break;
                   case '5' : lastbyte = 5; break;
                   case '6' : lastbyte = 6; break;
                   case '7' : lastbyte = 7; break;
                   case '8' : lastbyte = 8; break;
                   case '9' : lastbyte = 9; break;
                   case 'A' : lastbyte = 10; break;
                   case 'B' : lastbyte = 11; break;
                   case 'C' : lastbyte = 12; break;
                   case 'D' : lastbyte = 13; break;
                   case 'E' : lastbyte = 14; break;
                   case 'F' : lastbyte = 15; break;
                   case 'G' : lastbyte = 16; break;
              }
         }

         if(nBytesRead2==1)              //last byte block
         {
                fileSink.Write(decrypted, lastbyte);  //Write
         }
         else
         {
                fileSink.Write(decrypted, filewrite);  //Write
         }

              nBytesRead = nBytesRead2;          //Transfer Byte
              for(int n=0; n<fileread; n++)
              {
                   m_byte[n] = m_byte2[n];
              }

        }while(nBytesRead==fileread);
         //Decryption;
```

```
        fileSource.Close();
        fileSink.Close();

//        m_password.Empty();              //Delete password
//        UpdateData(FALSE);
       m_progress_crypt.SetPos(0); //set step progress at 0
    }
    else
    {          MessageBox("Password not Support", "Password",
                                MB_OK/MB_ICONINFORMATION);

    }

}

//////////////Function Crypt Algorithm//////////////


/////////Code for Crypt String [Public Key]//////

char* tab2::RSAEncryptString(const char *pubFilename, const char *seed, const
char *message)
{
        FileSource pubFile(pubFilename, true, new HexDecoder);
        RSAES_OAEP_SHA_Encryptor pub(pubFile);

        if (strlen(message) > pub.MaxPlainTextLength())
        {
                cerr << "message too long for this key\n";
                abort();
        }

        RandomPool randPool;
        randPool.Put((byte *)seed, strlen(seed));

        char *outstr = new char[2*pub.CipherTextLength()+1];
        pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

        HexEncoder hexEncoder;
        hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
        hexEncoder.Close();
        hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

        outstr[2*pub.CipherTextLength()] = 0;
        return outstr;
}

char* tab2::RSADecryptString(const char *privFilename, const char *ciphertext)
{
        FileSource privFile(privFilename, true, new HexDecoder);
        RSAES_OAEP_SHA_Decryptor priv(privFile);

        HexDecoder hexDecoder;
        hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
        hexDecoder.Close();
        SecByteBlock buf(priv.CipherTextLength());
        hexDecoder.Get(buf, priv.CipherTextLength());

        char *outstr = new char[priv.MaxPlainTextLength()+1];
        unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
        outstr[messageLength] = 0;
        return outstr;
}
```

```
char* tab2::LUCEncryptString(const char *pubFilename, const char *seed, const
char *message)
{
	FileSource pubFile(pubFilename, true, new HexDecoder);
	LUCES_OAEP_SHA_Encryptor pub(pubFile);

	if (strlen(message) > pub.MaxPlainTextLength())
	{
		cerr << "message too long for this key\n";
		abort();
	}

	RandomPool randPool;
	randPool.Put((byte *)seed, strlen(seed));

	char *outstr = new char[2*pub.CipherTextLength()+1];
	pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

	HexEncoder hexEncoder;
	hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
	hexEncoder.Close();
	hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

	outstr[2*pub.CipherTextLength()] = 0;
	return outstr;
}

char* tab2::LUCDecryptString(const char *privFilename, const char *ciphertext)
{
	FileSource privFile(privFilename, true, new HexDecoder);
	LUCES_OAEP_SHA_Decryptor priv(privFile);

	HexDecoder hexDecoder;
	hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
	hexDecoder.Close();
	SecByteBlock buf(priv.CipherTextLength());
	hexDecoder.Get(buf, priv.CipherTextLength());

	char *outstr = new char[priv.MaxPlainTextLength()+1];
	unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
	outstr[messageLength] = 0;
	return outstr;
}

char* tab2::RabinEncryptString(const char *pubFilename, const char *seed,
const char *message)
{
	FileSource pubFile(pubFilename, true, new HexDecoder);
	RabinEncryptor pub(pubFile);

	if (strlen(message) > pub.MaxPlainTextLength())
	{
		cerr << "message too long for this key\n";
		abort();
	}

	RandomPool randPool;
	randPool.Put((byte *)seed, strlen(seed));

	char *outstr = new char[2*pub.CipherTextLength()+1];
	pub.Encrypt(randPool, (byte *)message, strlen(message), (byte *)outstr);

	HexEncoder hexEncoder;
	hexEncoder.Put((byte *)outstr, pub.CipherTextLength());
	hexEncoder.Close();
```

```
	hexEncoder.Get((byte *)outstr, 2*pub.CipherTextLength());

	outstr[2*pub.CipherTextLength()] = 0;
	return outstr;
}

char* tab2::RabinDecryptString(const char *privFilename, const char
*ciphertext)
{
	FileSource privFile(privFilename, true, new HexDecoder);
	RabinDecryptor priv(privFile);

	HexDecoder hexDecoder;
	hexDecoder.Put((byte *)ciphertext, strlen(ciphertext));
	hexDecoder.Close();
	SecByteBlock buf(priv.CipherTextLength());
	hexDecoder.Get(buf, priv.CipherTextLength());

	char *outstr = new char[priv.MaxPlainTextLength()+1];
	unsigned messageLength = priv.Decrypt(buf, (byte *)outstr);
	outstr[messageLength] = 0;
	return outstr;
}

/////////Code for Crypt String [Public Key]//////

void tab2::OnButtonSource()
{
	// TODO: Add your control notification handler code here
	CFileDialog fd(TRUE, NULL, NULL, NULL,
   "Text File|*.txt|Picture File|*.jpg; *.gif; *.bmp|Wave File|*.wav;
*.mp3|All File|*.*||");
	if(fd.DoModal()==IDOK)
	{
		m_source = fd.GetFileName();
		UpdateData(FALSE);
	}
}

void tab2::OnDblclkListAlgor()
{
	// TODO: Add your control notification handler code here
	char buffer[15];
	m_list_algor.GetText(m_list_algor.GetCurSel(), buffer);
	m_list_algor.DeleteString(m_list_algor.GetCurSel());
	m_list_algorsel.AddString(buffer);
}

void tab2::OnDblclkListAlgorsel()
{
	// TODO: Add your control notification handler code here
	char buffer[15];
	m_list_algorsel.GetText(m_list_algorsel.GetCurSel(), buffer);
	m_list_algorsel.DeleteString(m_list_algorsel.GetCurSel());
	m_list_algor.AddString(buffer);
}

void tab2::OnButtonWeight()
{
	// TODO: Add your control notification handler code here
	WeightDlg dialog;
	dialog.DoModal();
}
```

```
void tab2::OnButtonTest()
{
      // TODO: Add your control notification handler code here
   m_animate.Play(0, -1, -1);
   UpdateData(TRUE);
   CString buffer;
   CString m_source_buf;
   int n=0;
   time_t start_s, finish_s;
   struct _timeb start_m, finish_m;
   double diff_sec, diff_mil;
   time_t start_s2, finish_s2;
   struct _timeb start_m2, finish_m2;
   double diff_sec2, diff_mil2;
   UINT nBytesRead;
   byte m_byte, m_byte2;
   int fileread = 1;
   int n2=0;
   long n_source_0, n_source_1;
   long n_destination_0, n_destination_1;

   m_source_buf = m_source;

   globalData.n_count = m_list_algorsel.GetCount();

//////////start test///////////////
      if( (globalData.n_count != 0) &&
          ((m_source.Compare("Select Source FilE->")) != 0) &&
          (globalData.w_weight == 0)
        )
      {

        for(n=0; n<globalData.n_count; n++)
        {

          m_list_algorsel.GetText(n, buffer);
          globalData.n_buffer[n] = buffer;

//************ DES
          if( buffer.Find("DES") == 0)      //*
          {
              m_source = m_source_buf;
//                m_animate.Play(0, -1, -1);
                m_destination.Empty();
                m_password.Empty();
                m_destination = "des_en.buf";      //*
                m_password = "aaaaaaaa";           //*

   CStdioFile fileSource(m_source,
                         CFile::modeRead | CFile::typeBinary);
   globalData.n_filesize_source = fileSource.GetLength();

   //find bit 0 1
   n_source_0 = 0;
   n_source_1 = 0;
      do
      {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
```

```
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
            }
        }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer//////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
              _ftime( &start_m );
  ////////work
                des_N_crypt();                                          //*
  ////////work
              time( &finish_s );
              _ftime( &finish_m );
  ////////////end timer//////////////
globalData.n_key[n] = 8;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
        nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "des_en.buf";   //*
                m_destination = "des_de.buf";     //*
                m_password = "aaaaaaaa";          //*

  ////////////start timer2//////////////
              diff_sec2 = 0;
                diff_mil2 = 0;

              time( &start_s2 );
              _ftime( &start_m2 );
  ////////work
                des_D_crypt();                                          //*
  ////////work
```

```
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ IDEA
            if( buffer.Find("IDEA") == 0)     //*
            {
                m_source = m_source_buf;
//              m_animate.Play(0, -1, -1);
                m_destination.Empty();
                m_password.Empty();
                m_destination = "idea_en.buf";    //*
                m_password = "aaaaaaaaaaaaaaaa";          //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
    do
    {
        nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
        for(n2=1; n2<=8; n2++)
        {
          m_byte2 = m_byte & 1;
          if(m_byte2 == 1) { n_source_1++; }
          if(m_byte2 == 0) { n_source_0++; }
          m_byte = _rotr(m_byte, 1);
        }
      }
    }while(nBytesRead==fileread);
  //find bit 0 1

  fileSource.Close();

    ////////////start timer//////////////
                diff_sec = 0;
                  diff_mil = 0;

                time( &start_s );
                _ftime( &start_m );
    ////////work
                  idea_N_crypt();                                    //*
    ////////work
                time( &finish_s );
                _ftime( &finish_m );
    ////////////end timer//////////////
  globalData.n_key[n] = 16;

  CStdioFile fileSink(m_destination,
                      CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_destination[n] = fileSink.GetLength();

  //find bit 0 1
  n_destination_0 = 0;
  n_destination_1 = 0;
    do
```

```
    {
            nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
            {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
            }
        }
    }while(nBytesRead==fileread);
  //find bit 0 1

  fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "idea_en.buf";  //*
                m_destination = "idea_de.buf";    //*
                m_password = "aaaaaaaaaaaaaaaa";        //*

    ////////////start timer2//////////////
                diff_sec2 = 0;
                diff_mil2 = 0;

                time( &start_s2 );
                _ftime( &start_m2 );
    ////////work
                idea_D_crypt();                                 //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ Blowfish
            if( buffer.Find("Blowfish") == 0) //*
            {
                m_source = m_source_buf;
//              m_animate.Play(0, -1, -1);
                m_destination.Empty();
                m_password.Empty();
                m_destination = "blowfish_en.buf";      //*
                m_password = "aaaaaaaaaaaaaaaa";        //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
    do
    {
            nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
```

```
            {
                    m_byte2 = m_byte & 1;
                    if(m_byte2 == 1) { n_source_1++; }
                    if(m_byte2 == 0) { n_source_0++; }
                    m_byte = _rotr(m_byte, 1);
            }
        }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer//////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
               _ftime( &start_m );
  ////////work
                    blowfish_N_crypt();                                    //*
  ////////work
              time( &finish_s );
               _ftime( &finish_m );
  ////////////end timer//////////////
globalData.n_key[n] = 16;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "blowfish_en.buf";     //*
                m_destination = "blowfish_de.buf";      //*
                m_password = "aaaaaaaaaaaaaaaa";        //*

  ////////////start timer2//////////////
              diff_sec2 = 0;
                diff_mil2 = 0;

              time( &start_s2 );
               _ftime( &start_m2 );
```

```
    ////////work
                    blowfish_D_crypt();                                    //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2///////////////

            }
//************

//************ CAST128
            if( buffer.Find("CAST128") == 0)  //*
            {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "cast128_en.buf"; //*
                    m_password = "aaaaaaaaaaaaaaaa";          //*

   CStdioFile fileSource(m_source,
                         CFile::modeRead | CFile::typeBinary);
   globalData.n_filesize_source = fileSource.GetLength();

   //find bit 0 1
   n_source_0 = 0;
   n_source_1 = 0;
       do
       {
            nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
            {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
            }
        }
       }while(nBytesRead==fileread);
   //find bit 0 1

   fileSource.Close();

     ////////////start timer///////////////
                diff_sec = 0;
                  diff_mil = 0;

                time( &start_s );
                _ftime( &start_m );
     ////////work
                  cast128_N_crypt();                                   //*
     ////////work
                time( &finish_s );
                _ftime( &finish_m );
     ////////////end timer///////////////
   globalData.n_key[n] = 16;

   CStdioFile fileSink(m_destination,
                       CFile::modeRead | CFile::typeBinary);
   globalData.n_filesize_destination[n] = fileSink.GetLength();

   //find bit 0 1
```

```
    n_destination_0 = 0;
    n_destination_1 = 0;
        do
        {
              nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
               for(n2=1; n2<=8; n2++)
               {
                     m_byte2 = m_byte & 1;
                     if(m_byte2 == 1) { n_destination_1++; }
                     if(m_byte2 == 0) { n_destination_0++; }
                     m_byte = _rotr(m_byte, 1);
               }
          }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                     m_source.Empty();
                     m_destination.Empty();
                     m_password.Empty();
                     m_source = "cast128_en.buf";      //*
                     m_destination = "cast128_de.buf"; //*
                     m_password = "aaaaaaaaaaaaaaaa";          //*

      ////////////start timer2/////////////
                    diff_sec2 = 0;
                     diff_mil2 = 0;

                   time( &start_s2 );
                   _ftime( &start_m2 );
      ////////work
                     cast128_D_crypt();                                  //*
      ////////work
                   time( &finish_s2 );
                   _ftime( &finish_m2 );
      ////////////end timer2/////////////

               }
//************

//************ GOST
               if( buffer.Find("GOST") == 0)     //*
               {
                     m_source = m_source_buf;
//                   m_animate.Play(0, -1, -1);
                     m_destination.Empty();
                     m_password.Empty();
                     m_destination = "gost_en.buf";    //*
                     m_password = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";//*

    CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_source = fileSource.GetLength();

    //find bit 0 1
    n_source_0 = 0;
    n_source_1 = 0;
        do
        {
              nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file
```

```
      if(nBytesRead==fileread)
      {
           for(n2=1; n2<=8; n2++)
           {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
           }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer//////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
               _ftime( &start_m );
  ////////work
                gost_N_crypt();                                    //*
  ////////work
              time( &finish_s );
               _ftime( &finish_m );
  ////////////end timer//////////////
globalData.n_key[n] = 32;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
           for(n2=1; n2<=8; n2++)
           {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
           }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "gost_en.buf";  //*
                m_destination = "gost_de.buf";    //*
                m_password = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";//*

  ////////////start timer2//////////////
              diff_sec2 = 0;
                diff_mil2 = 0;
```

```
                time( &start_s2 );
                _ftime( &start_m2 );
    ////////work
                    gost_D_crypt();                                    //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//***********

//*********** MARS
            if( buffer.Find("MARS") == 0)     //*
            {
                m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                m_destination.Empty();
                m_password.Empty();
                m_destination = "mars_en.buf";    //*
                m_password = "aaaaaaaaaaaaaaaa";          //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
     do
     {
        nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
        for(n2=1; n2<=8; n2++)
        {
            m_byte2 = m_byte & 1;
            if(m_byte2 == 1) { n_source_1++; }
            if(m_byte2 == 0) { n_source_0++; }
            m_byte = _rotr(m_byte, 1);
        }
      }
     }while(nBytesRead==fileread);
  //find bit 0 1

  fileSource.Close();

    ////////////start timer//////////////
                diff_sec = 0;
                   diff_mil = 0;

                time( &start_s );
                _ftime( &start_m );
    ////////work
                    mars_N_crypt();                                    //*
    ////////work
                time( &finish_s );
                _ftime( &finish_m );
    ////////////end timer//////////////
  globalData.n_key[n] = 16;

  CStdioFile fileSink(m_destination,
                      CFile::modeRead | CFile::typeBinary);
```

```
    globalData.n_filesize_destination[n] = fileSink.GetLength();

    //find bit 0 1
    n_destination_0 = 0;
    n_destination_1 = 0;
        do
        {
              nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

         if(nBytesRead==fileread)
         {
              for(n2=1; n2<=8; n2++)
              {
                    m_byte2 = m_byte & 1;
                    if(m_byte2 == 1) { n_destination_1++; }
                    if(m_byte2 == 0) { n_destination_0++; }
                    m_byte = _rotr(m_byte, 1);
              }
         }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                    m_source.Empty();
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "mars_en.buf";  //*
                    m_destination = "mars_de.buf";    //*
                    m_password = "aaaaaaaaaaaaaaaa";          //*

      ////////////start timer2//////////////
                   diff_sec2 = 0;
                     diff_mil2 = 0;

                   time( &start_s2 );
                    _ftime( &start_m2 );
      ///////work
                    mars_D_crypt();                                    //*
      ///////work
                   time( &finish_s2 );
                    _ftime( &finish_m2 );
      ////////////end timer2//////////////

              }
//************

//************ RC2
              if( buffer.Find("RC2") == 0)      //*
              {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "rc2_en.buf";     //*
                    m_password = "aaaaaaaaaaaaaaaa";          //*

    CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_source = fileSource.GetLength();

    //find bit 0 1
    n_source_0 = 0;
    n_source_1 = 0;
        do
```

```
    {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer/////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
              _ftime( &start_m );
  ///////work
                rc2_N_crypt();                                          //*
  ///////work
              time( &finish_s );
              _ftime( &finish_m );
  ////////////end timer/////////////
globalData.n_key[n] = 16;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "rc2_en.buf";   //*
                m_destination = "rc2_de.buf";     //*
                m_password = "aaaaaaaaaaaaaaaa";        //*
```

```
      ////////////start timer2//////////////
                  diff_sec2 = 0;
                     diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
      ////////work
                     rc2_D_crypt();                                    //*
      ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
      ////////////end timer2//////////////

               }
//************

//************ Rijndael
               if( buffer.Find("Rijndael") == 0) //*
               {
                     m_source = m_source_buf;
//                   m_animate.Play(0, -1, -1);
                     m_destination.Empty();
                     m_password.Empty();
                     m_destination = "rijndael_en.buf";          //*
                     m_password = "aaaaaaaaaaaaaaaa";            //*

   CStdioFile fileSource(m_source,
                         CFile::modeRead | CFile::typeBinary);
   globalData.n_filesize_source = fileSource.GetLength();

   //find bit 0 1
   n_source_0 = 0;
   n_source_1 = 0;
       do
       {
            nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

         if(nBytesRead==fileread)
         {
            for(n2=1; n2<=8; n2++)
            {
               m_byte2 = m_byte & 1;
               if(m_byte2 == 1) { n_source_1++; }
               if(m_byte2 == 0) { n_source_0++; }
               m_byte = _rotr(m_byte, 1);
            }
         }
       }while(nBytesRead==fileread);
   //find bit 0 1

   fileSource.Close();

      ////////////start timer//////////////
                  diff_sec = 0;
                     diff_mil = 0;

                  time( &start_s );
                  _ftime( &start_m );
      ////////work
                     rijndael_N_crypt();                             //*
      ////////work
                  time( &finish_s );
                  _ftime( &finish_m );
      ////////////end timer//////////////
   globalData.n_key[n] = 16;
```

```
    CStdioFile fileSink(m_destination,
                        CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_destination[n] = fileSink.GetLength();

    //find bit 0 1
    n_destination_0 = 0;
    n_destination_1 = 0;
        do
        {
              nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
              for(n2=1; n2<=8; n2++)
              {
                    m_byte2 = m_byte & 1;
                    if(m_byte2 == 1) { n_destination_1++; }
                    if(m_byte2 == 0) { n_destination_0++; }
                    m_byte = _rotr(m_byte, 1);
              }
          }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                    m_source.Empty();
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "rijndael_en.buf";     //*
                    m_destination = "rijndael_de.buf";      //*
                    m_password = "aaaaaaaaaaaaaaaa";        //*

      ////////////start timer2///////////////
                  diff_sec2 = 0;
                    diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
      ////////work
                    rijndael_D_crypt();                                   //*
      ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
      ////////////end timer2///////////////

              }
//************

//************ Serpent
              if( buffer.Find("Serpent") == 0) //*
              {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "serpent_en.buf"; //*
                    m_password = "aaaaaaaaaaaaaaaa";        //*

    CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_source = fileSource.GetLength();

    //find bit 0 1
```

```
n_source_0 = 0;
n_source_1 = 0;
    do
    {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer///////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
              _ftime( &start_m );
  ////////work
                serpent_N_crypt();                                    //*
  ////////work
              time( &finish_s );
              _ftime( &finish_m );
  ////////////end timer///////////////
globalData.n_key[n] = 16;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "serpent_en.buf";      //*
```

```
                    m_destination = "serpent_de.buf"; //*
                    m_password = "aaaaaaaaaaaaaaaa";        //*

    ////////////start timer2//////////////
                  diff_sec2 = 0;
                    diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
    ////////work
                    serpent_D_crypt();                                  //*
    ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ SHARK
            if( buffer.Find("SHARK") == 0)    //*
            {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "shark_en.buf";   //*
                    m_password = "aaaaaaaaaaaaaaaa";        //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
     do
     {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

       if(nBytesRead==fileread)
       {
            for(n2=1; n2<=8; n2++)
            {
                  m_byte2 = m_byte & 1;
                  if(m_byte2 == 1) { n_source_1++; }
                  if(m_byte2 == 0) { n_source_0++; }
                  m_byte = _rotr(m_byte, 1);
            }
       }
     }while(nBytesRead==fileread);
  //find bit 0 1

  fileSource.Close();

    ////////////start timer//////////////
                  diff_sec = 0;
                    diff_mil = 0;

                  time( &start_s );
                  _ftime( &start_m );
    ////////work
                    shark_N_crypt();                                    //*
    ////////work
                  time( &finish_s );
```

```
                    _ftime( &finish_m );
      ////////////end timer///////////////
    globalData.n_key[n] = 16;

    CStdioFile fileSink(m_destination,
                        CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_destination[n] = fileSink.GetLength();

    //find bit 0 1
    n_destination_0 = 0;
    n_destination_1 = 0;
        do
        {
              nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
               for(n2=1; n2<=8; n2++)
               {
                     m_byte2 = m_byte & 1;
                     if(m_byte2 == 1) { n_destination_1++; }
                     if(m_byte2 == 0) { n_destination_0++; }
                     m_byte = _rotr(m_byte, 1);
               }
          }
       }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                     m_source.Empty();
                     m_destination.Empty();
                     m_password.Empty();
                     m_source = "shark_en.buf"; //*
                     m_destination = "shark_de.buf";  //*
                     m_password = "aaaaaaaaaaaaaaaa";        //*

     ////////////start timer2//////////////
                   diff_sec2 = 0;
                     diff_mil2 = 0;

                   time( &start_s2 );
                   _ftime( &start_m2 );
     ///////work
                      shark_D_crypt();                                  //*
     ///////work
                   time( &finish_s2 );
                   _ftime( &finish_m2 );
     ////////////end timer2//////////////

               }
//************

//************ Square
              if( buffer.Find("Square") == 0)   //*
              {
                     m_source = m_source_buf;
//                   m_animate.Play(0, -1, -1);
                     m_destination.Empty();
                     m_password.Empty();
                     m_destination = "square_en.buf"; //*
                     m_password = "aaaaaaaaaaaaaaaa";        //*

    CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
```

```
globalData.n_filesize_source = fileSource.GetLength();

//find bit 0 1
n_source_0 = 0;
n_source_1 = 0;
    do
    {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer/////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
              _ftime( &start_m );
  ///////work
                square_N_crypt();                                    //*
  ///////work
              time( &finish_s );
              _ftime( &finish_m );
  ////////////end timer/////////////
globalData.n_key[n] = 16;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSink.Close();

                m_source.Empty();
```

```
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "square_en.buf";       //*
                    m_destination = "square_de.buf"; //*
                    m_password = "aaaaaaaaaaaaaaaa";         //*

    ////////////start timer2//////////////
                  diff_sec2 = 0;
                    diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
    ////////work
                    square_D_crypt();                                    //*
    ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//***********

//*********** TEA
            if( buffer.Find("TEA") == 0)      //*
            {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "tea_en.buf";      //*
                    m_password = "aaaaaaaaaaaaaaaa";         //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
     do
     {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

       if(nBytesRead==fileread)
       {
            for(n2=1; n2<=8; n2++)
            {
                  m_byte2 = m_byte & 1;
                  if(m_byte2 == 1) { n_source_1++; }
                  if(m_byte2 == 0) { n_source_0++; }
                  m_byte = _rotr(m_byte, 1);
            }
       }
     }while(nBytesRead==fileread);
  //find bit 0 1

  fileSource.Close();

    ////////////start timer//////////////
                  diff_sec = 0;
                    diff_mil = 0;

                  time( &start_s );
                  _ftime( &start_m );
    ////////work
```

```
                    tea_N_crypt();                                          //*
    ////////work
                time( &finish_s );
                _ftime( &finish_m );
    ////////////end timer//////////////
    globalData.n_key[n] = 16;

    CStdioFile fileSink(m_destination,
                        CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_destination[n] = fileSink.GetLength();

    //find bit 0 1
    n_destination_0 = 0;
    n_destination_1 = 0;
       do
       {
            nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
            {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
            }
        }
       }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                    m_source.Empty();
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "tea_en.buf";   //*
                    m_destination = "tea_de.buf";     //*
                    m_password = "aaaaaaaaaaaaaaaa";          //*

    ////////////start timer2//////////////
                diff_sec2 = 0;
                    diff_mil2 = 0;

                time( &start_s2 );
                _ftime( &start_m2 );
    ////////work
                    tea_D_crypt();                                          //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ ThreeWay
            if( buffer.Find("ThreeWay") == 0) //*
            {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "threeway_en.buf";      //*
                    m_password = "aaaaaaaaaaaa";            //*
```

```
CStdioFile fileSource(m_source,
                      CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_source = fileSource.GetLength();

//find bit 0 1
n_source_0 = 0;
n_source_1 = 0;
    do
    {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer//////////////
            diff_sec = 0;
              diff_mil = 0;

            time( &start_s );
            _ftime( &start_m );
  ///////work
              threeway_N_crypt();                                    //*
  ///////work
            time( &finish_s );
            _ftime( &finish_m );
  ////////////end timer//////////////
globalData.n_key[n] = 12;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1
```

```
    fileSink.Close();

                    m_source.Empty();
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "threeway_en.buf";     //*
                    m_destination = "threeway_de.buf";      //*
                    m_password = "aaaaaaaaaaaa";            //*

      ////////////start timer2//////////////
                  diff_sec2 = 0;
                    diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
      ////////work
                    threeway_D_crypt();                              //*
      ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
      ////////////end timer2//////////////

              }
//************

//************ Twofish
              if( buffer.Find("Twofish") == 0) //*
              {
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "twofish_en.buf"; //*
                    m_password = "aaaaaaaaaaaaaaaa";        //*

    CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_source = fileSource.GetLength();

    //find bit 0 1
    n_source_0 = 0;
    n_source_1 = 0;
        do
        {
              nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
               for(n2=1; n2<=8; n2++)
               {
                     m_byte2 = m_byte & 1;
                     if(m_byte2 == 1) { n_source_1++; }
                     if(m_byte2 == 0) { n_source_0++; }
                     m_byte = _rotr(m_byte, 1);
               }
          }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSource.Close();

      ////////////start timer//////////////
                  diff_sec = 0;
                    diff_mil = 0;
```

```
                time( &start_s );
                _ftime( &start_m );
    ////////work
                  twofish_N_crypt();                                    //*
    ////////work
                time( &finish_s );
                _ftime( &finish_m );
    ////////////end timer//////////////
  globalData.n_key[n] = 16;

  CStdioFile fileSink(m_destination,
                      CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_destination[n] = fileSink.GetLength();

  //find bit 0 1
  n_destination_0 = 0;
  n_destination_1 = 0;
      do
      {
            nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
            {
                  m_byte2 = m_byte & 1;
                  if(m_byte2 == 1) { n_destination_1++; }
                  if(m_byte2 == 0) { n_destination_0++; }
                  m_byte = _rotr(m_byte, 1);
            }
        }
      }while(nBytesRead==fileread);
  //find bit 0 1

  fileSink.Close();

                  m_source.Empty();
                  m_destination.Empty();
                  m_password.Empty();
                  m_source = "twofish_en.buf";      //*
                  m_destination = "twofish_de.buf"; //*
                  m_password = "aaaaaaaaaaaaaaaa";        //*

    ////////////start timer2//////////////
                diff_sec2 = 0;
                  diff_mil2 = 0;

                time( &start_s2 );
                _ftime( &start_m2 );
    ////////work
                  twofish_D_crypt();                                    //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ RSA<pb>
            if( buffer.Find("RSA<pb>") == 0) //*
            {
                  m_source = m_source_buf;
//                m_animate.Play(0, -1, -1);
                  m_destination.Empty();
```

```
                m_password.Empty();
                m_destination = "rsa_en.buf";     //*
                m_password = "key_pb.rsa";        //*

CStdioFile fileSource(m_source,
                      CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_source = fileSource.GetLength();

//find bit 0 1
n_source_0 = 0;
n_source_1 = 0;
    do
    {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
    }while(nBytesRead==fileread);
//find bit 0 1

fileSource.Close();

  ////////////start timer//////////////
              diff_sec = 0;
                diff_mil = 0;

              time( &start_s );
               _ftime( &start_m );
  ////////work
                rsa_N_crypt();                                    //*
  ////////work
              time( &finish_s );
               _ftime( &finish_m );
  ////////////end timer//////////////
globalData.n_key[n] = 634;

CStdioFile fileSink(m_destination,
                    CFile::modeRead | CFile::typeBinary);
globalData.n_filesize_destination[n] = fileSink.GetLength();

//find bit 0 1
n_destination_0 = 0;
n_destination_1 = 0;
    do
    {
          nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
```

```
    }while(nBytesRead==fileread);
  //find bit 0 1

  fileSink.Close();

                  m_source.Empty();
                  m_destination.Empty();
                  m_password.Empty();
                  m_source = "rsa_en.buf";   //*
                  m_destination = "rsa_de.buf";     //*
                  m_password = "key_pv.rsa";        //*

   ////////////start timer2//////////////
                 diff_sec2 = 0;
                   diff_mil2 = 0;

                 time( &start_s2 );
                  _ftime( &start_m2 );
   ////////work
                   rsa_D_crypt();                                    //*
   ////////work
                 time( &finish_s2 );
                  _ftime( &finish_m2 );
   ////////////end timer2//////////////

            }
//************

//************ LUC<pb>
            if( buffer.Find("LUC<pb>") == 0) //*
            {
                  m_source = m_source_buf;
//                m_animate.Play(0, -1, -1);
                  m_destination.Empty();
                  m_password.Empty();
                  m_destination = "luc_en.buf";     //*
                  m_password = "key_pb.luc";        //*

  CStdioFile fileSource(m_source,
                        CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_source = fileSource.GetLength();

  //find bit 0 1
  n_source_0 = 0;
  n_source_1 = 0;
     do
     {
          nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

      if(nBytesRead==fileread)
      {
          for(n2=1; n2<=8; n2++)
          {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_source_1++; }
                if(m_byte2 == 0) { n_source_0++; }
                m_byte = _rotr(m_byte, 1);
          }
      }
     }while(nBytesRead==fileread);
  //find bit 0 1

  fileSource.Close();

   ////////////start timer//////////////
```

```
                diff_sec = 0;
                  diff_mil = 0;

                time( &start_s );
                _ftime( &start_m );
    ////////work
                  luc_N_crypt();                                    //*
    ////////work
                time( &finish_s );
                _ftime( &finish_m );
    ////////////end timer//////////////
  globalData.n_key[n] = 360;

  CStdioFile fileSink(m_destination,
                      CFile::modeRead | CFile::typeBinary);
  globalData.n_filesize_destination[n] = fileSink.GetLength();

  //find bit 0 1
  n_destination_0 = 0;
  n_destination_1 = 0;
      do
      {
            nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

        if(nBytesRead==fileread)
        {
            for(n2=1; n2<=8; n2++)
            {
                m_byte2 = m_byte & 1;
                if(m_byte2 == 1) { n_destination_1++; }
                if(m_byte2 == 0) { n_destination_0++; }
                m_byte = _rotr(m_byte, 1);
            }
        }
      }while(nBytesRead==fileread);
  //find bit 0 1

  fileSink.Close();

                m_source.Empty();
                m_destination.Empty();
                m_password.Empty();
                m_source = "luc_en.buf";   //*
                m_destination = "luc_de.buf";     //*
                m_password = "key_pv.luc";        //*

    ////////////start timer2//////////////
                diff_sec2 = 0;
                  diff_mil2 = 0;

                time( &start_s2 );
                _ftime( &start_m2 );
    ////////work
                  luc_D_crypt();                                    //*
    ////////work
                time( &finish_s2 );
                _ftime( &finish_m2 );
    ////////////end timer2//////////////

            }
//************

//************ Rabin<pb>
            if( buffer.Find("Rabin<pb>") == 0)      //*
            {
```

```
                    m_source = m_source_buf;
//                  m_animate.Play(0, -1, -1);
                    m_destination.Empty();
                    m_password.Empty();
                    m_destination = "rabin_en.buf";   //*
                    m_password = "key_pb.rbn";        //*

    CStdioFile fileSource(m_source,
                          CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_source = fileSource.GetLength();

    //find bit 0 1
    n_source_0 = 0;
    n_source_1 = 0;
        do
        {
              nBytesRead = fileSource.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
              for(n2=1; n2<=8; n2++)
              {
                    m_byte2 = m_byte & 1;
                    if(m_byte2 == 1) { n_source_1++; }
                    if(m_byte2 == 0) { n_source_0++; }
                    m_byte = _rotr(m_byte, 1);
              }
          }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSource.Close();

      ////////////start timer///////////////
                  diff_sec = 0;
                    diff_mil = 0;

                  time( &start_s );
                 _ftime( &start_m );
      ////////work
                    rabin_N_crypt();                                    //*
      ////////work
                  time( &finish_s );
                  _ftime( &finish_m );
      ////////////end timer///////////////
    globalData.n_key[n] = 360;

    CStdioFile fileSink(m_destination,
                        CFile::modeRead | CFile::typeBinary);
    globalData.n_filesize_destination[n] = fileSink.GetLength();

    //find bit 0 1
    n_destination_0 = 0;
    n_destination_1 = 0;
        do
        {
              nBytesRead = fileSink.Read(&m_byte, fileread);  //Read file

          if(nBytesRead==fileread)
          {
              for(n2=1; n2<=8; n2++)
              {
                    m_byte2 = m_byte & 1;
                    if(m_byte2 == 1) { n_destination_1++; }
                    if(m_byte2 == 0) { n_destination_0++; }
```

```
                    m_byte = _rotr(m_byte, 1);
                }
            }
        }while(nBytesRead==fileread);
    //find bit 0 1

    fileSink.Close();

                    m_source.Empty();
                    m_destination.Empty();
                    m_password.Empty();
                    m_source = "rabin_en.buf"; //*
                    m_destination = "rabin_de.buf";   //*
                    m_password = "key_pv.rbn";        //*

     ////////////start timer2//////////////
                  diff_sec2 = 0;
                    diff_mil2 = 0;

                  time( &start_s2 );
                  _ftime( &start_m2 );
     ////////work
                    rabin_D_crypt();                                    //*
     ////////work
                  time( &finish_s2 );
                  _ftime( &finish_m2 );
     ////////////end timer2//////////////

                }
//************


      //cal diff time
     diff_sec = difftime( finish_s, start_s );
    if( start_s == finish_s )
    {
            diff_mil = finish_m.millitm - start_m.millitm;
    }
    else
    {
          if( start_m.millitm > finish_m.millitm )
          {
                    diff_sec--;
                    diff_mil = (1000 - start_m.millitm) + finish_m.millitm;
          }
          else
          {
                    diff_mil = finish_m.millitm - start_m.millitm;
          }
    }
       diff_sec = diff_sec + ( diff_mil/1000 );
     globalData.n_diff_sec_en[n] = diff_sec;
       //cal diff time     end

       //cal diff time     2
     diff_sec2 = difftime( finish_s2, start_s2 );
    if( start_s2 == finish_s2 )
    {
            diff_mil2 = finish_m2.millitm - start_m2.millitm;
    }
    else
    {
          if( start_m2.millitm > finish_m2.millitm )
          {
                    diff_sec2--;
```

```
                    diff_mil2 = (1000 - start_m2.millitm) + finish_m2.millitm;
            }
            else
            {
                    diff_mil2 = finish_m2.millitm - start_m2.millitm;
            }
    }
        diff_sec2 = diff_sec2 + ( diff_mil2/1000 );
     globalData.n_diff_sec_de[n] = diff_sec2;
        //cal diff time     end 2

        //cal spread
   globalData.n_spread_0[n] = abs(((n_destination_0 - n_source_0) * 10000) /
n_source_0);
   globalData.n_spread_1[n] = abs(((n_destination_1 - n_source_1) * 10000) /
n_source_1);
        //cal spread


         }//end loop for

        m_animate.Stop();
        GraphtestDlg dialog;
        dialog.DoModal();

        }
//////////end test///////////////
else
{
        if((m_source.Compare("Select Source FilE->")) == 0)
        {
              MessageBox("File not Select", "File",
                                MB_OK/MB_ICONINFORMATION);
        }
        if(globalData.n_count == 0)
        {
              MessageBox("Algorithm not Select", "Algor",
                                MB_OK/MB_ICONINFORMATION);
        }
        if(globalData.w_weight != 0)
        {
              MessageBox("Weight must Select Empty 100%", "Weight",
                                MB_OK/MB_ICONINFORMATION);
        }
}
    m_source = m_source_buf;
    UpdateData(FALSE);
       m_animate.Stop();

}
```

```
#if !defined(AFX_TAB3_H__C1278362_A0CA_4C1A_8CB0_1EAD0C082ED9__INCLUDED_)
#define AFX_TAB3_H__C1278362_A0CA_4C1A_8CB0_1EAD0C082ED9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// tab3.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// tab3 dialog

class tab3 : public CPropertyPage
{
	DECLARE_DYNCREATE(tab3)

// Construction
public:
	tab3();
	~tab3();

// Dialog Data
	//{{AFX_DATA(tab3)
	enum { IDD = IDD_DIALOG_ABOUT };
		// NOTE - ClassWizard will add data members here.
		//    DO NOT EDIT what you see in these blocks of generated code
!
	//}}AFX_DATA


// Overrides
	// ClassWizard generate virtual function overrides
	//{{AFX_VIRTUAL(tab3)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	// Generated message map functions
	//{{AFX_MSG(tab3)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_TAB3_H__C1278362_A0CA_4C1A_8CB0_1EAD0C082ED9__INCLUDED_)
```

```
// tab3.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "tab3.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// tab3 property page

IMPLEMENT_DYNCREATE(tab3, CPropertyPage)

tab3::tab3() : CPropertyPage(tab3::IDD)
{
	//{{AFX_DATA_INIT(tab3)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}

tab3::~tab3()
{
}

void tab3::DoDataExchange(CDataExchange* pDX)
{
	CPropertyPage::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(tab3)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(tab3, CPropertyPage)
	//{{AFX_MSG_MAP(tab3)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// tab3 message handlers
```

```
#if !defined(AFX_WEIGHTDLG_H__838D6BE4_2832_4C45_80A8_A179EF5881CE__INCLUDED_)
#define AFX_WEIGHTDLG_H__838D6BE4_2832_4C45_80A8_A179EF5881CE__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// WeightDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// WeightDlg dialog

class WeightDlg : public CDialog
{
// Construction
public:
	WeightDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(WeightDlg)
	enum { IDD = IDD_DIALOG_WEIGHT };
	CSliderCtrl	m_slider6;
	CSliderCtrl	m_slider5;
	CSliderCtrl	m_slider4;
	CSliderCtrl	m_slider3;
	CSliderCtrl	m_slider2;
	CSliderCtrl	m_slider1;
	int		m_slidervalue1;
	int		m_slidervalue2;
	int		m_slidervalue3;
	int		m_slidervalue4;
	int		m_slidervalue5;
	int		m_value2;
	int		m_value3;
	int		m_value4;
	int		m_value1;
	int		m_value5;
	int		m_value6;
	int		m_slidervalue6;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(WeightDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual BOOL OnInitDialog(); //*

protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(WeightDlg)
	afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
	virtual void OnOK();
	afx_msg void OnReset();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_WEIGHTDLG_H__838D6BE4_2832_4C45_80A8_A179EF5881CE__INCLUDED_)



// WeightDlg.cpp : implementation file
//

#include "stdafx.h"
#include "begin6.h"
#include "WeightDlg.h"

#include "globals.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// WeightDlg dialog


WeightDlg::WeightDlg(CWnd* pParent /*=NULL*/)
	: CDialog(WeightDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(WeightDlg)
	m_slidervalue1 = 0;
	m_slidervalue2 = 0;
	m_slidervalue3 = 0;
	m_slidervalue4 = 0;
	m_slidervalue5 = 0;
	m_value2 = 0;
	m_value3 = 0;
	m_value4 = 0;
	m_value1 = 0;
	m_value5 = 0;
	m_value6 = 0;
	m_slidervalue6 = 0;
	//}}AFX_DATA_INIT
}


void WeightDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(WeightDlg)
	DDX_Control(pDX, IDC_SLIDER6, m_slider6);
	DDX_Control(pDX, IDC_SLIDER5, m_slider5);
	DDX_Control(pDX, IDC_SLIDER4, m_slider4);
	DDX_Control(pDX, IDC_SLIDER3, m_slider3);
	DDX_Control(pDX, IDC_SLIDER2, m_slider2);
	DDX_Control(pDX, IDC_SLIDER1, m_slider1);
	DDX_Slider(pDX, IDC_SLIDER1, m_slidervalue1);
	DDX_Slider(pDX, IDC_SLIDER2, m_slidervalue2);
	DDX_Slider(pDX, IDC_SLIDER3, m_slidervalue3);
	DDX_Slider(pDX, IDC_SLIDER4, m_slidervalue4);
	DDX_Slider(pDX, IDC_SLIDER5, m_slidervalue5);
```

```
        DDX_Text(pDX, IDC_EDIT2, m_value2);
        DDX_Text(pDX, IDC_EDIT3, m_value3);
        DDX_Text(pDX, IDC_EDIT4, m_value4);
        DDX_Text(pDX, IDC_EDIT1, m_value1);
        DDX_Text(pDX, IDC_EDIT5, m_value5);
        DDX_Text(pDX, IDC_EDIT6, m_value6);
        DDX_Slider(pDX, IDC_SLIDER6, m_slidervalue6);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(WeightDlg, CDialog)
        //{{AFX_MSG_MAP(WeightDlg)
        ON_WM_VSCROLL()
        ON_BN_CLICKED(ID_RESET, OnReset)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// WeightDlg message handlers
BOOL WeightDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);        // Set small icon

        // TODO: Add extra initialization here
        m_slider1.SetRange(0, 100);
        m_slider2.SetRange(0, 100);
        m_slider3.SetRange(0, 100);
        m_slider4.SetRange(0, 100);
        m_slider5.SetRange(0, 100);
        m_slider6.SetRange(0, 100);

        m_slidervalue1 = 100-globalData.w_weight;
        m_slidervalue2 = 100-globalData.w_speed_n;
        m_slidervalue3 = 100-globalData.w_speed_d;
        m_slidervalue4 = 100-globalData.w_key;
        m_slidervalue5 = 100-globalData.w_spread;
        m_slidervalue6 = 100-globalData.w_filesize;

        m_value1 = 100-m_slidervalue1;
        m_value2 = 100-m_slidervalue2;
        m_value3 = 100-m_slidervalue3;
        m_value4 = 100-m_slidervalue4;
        m_value5 = 100-m_slidervalue5;
        m_value6 = 100-m_slidervalue6;

        CButton *Fast, *Slow;
        Fast = (CButton*)GetDlgItem(IDC_RADIO_FAST);
        Slow = (CButton*)GetDlgItem(IDC_RADIO_SLOW);

        Fast->SetCheck(1);

        UpdateData(FALSE);


        return TRUE;  // return TRUE  unless you set the focus to a control

}
```

```
void WeightDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
	// TODO: Add your message handler code here and/or call default

	int buffer1 = m_slidervalue1;
	int buffer2 = m_slidervalue2;
	int buffer3 = m_slidervalue3;
	int buffer4 = m_slidervalue4;
	int buffer5 = m_slidervalue5;
	int buffer6 = m_slidervalue6;

		UpdateData(TRUE);

	if(m_slidervalue1 < 100)
	{

		if( ((buffer2 - m_slidervalue2) <= (100 - m_slidervalue1)) &&
			((buffer3 - m_slidervalue3) <= (100 - m_slidervalue1)) &&
			((buffer4 - m_slidervalue4) <= (100 - m_slidervalue1)) &&
			((buffer5 - m_slidervalue5) <= (100 - m_slidervalue1)) &&
			((buffer6 - m_slidervalue6) <= (100 - m_slidervalue1))
		  )
		{

		   m_slidervalue1 = m_slidervalue1 + (buffer2 - m_slidervalue2);
		   m_slidervalue1 = m_slidervalue1 + (buffer3 - m_slidervalue3);
		   m_slidervalue1 = m_slidervalue1 + (buffer4 - m_slidervalue4);
		   m_slidervalue1 = m_slidervalue1 + (buffer5 - m_slidervalue5);
		   m_slidervalue1 = m_slidervalue1 + (buffer6 - m_slidervalue6);
		}

	}
	else
	{


		if( ((buffer2 - m_slidervalue2) < 0) ||
			((buffer3 - m_slidervalue3) < 0) ||
			((buffer4 - m_slidervalue4) < 0) ||
			((buffer5 - m_slidervalue5) < 0) ||
			((buffer6 - m_slidervalue6) < 0)
		  )
		{
		   m_slidervalue1 = m_slidervalue1 + (buffer2 - m_slidervalue2);
		   m_slidervalue1 = m_slidervalue1 + (buffer3 - m_slidervalue3);
		   m_slidervalue1 = m_slidervalue1 + (buffer4 - m_slidervalue4);
		   m_slidervalue1 = m_slidervalue1 + (buffer5 - m_slidervalue5);

		   m_slidervalue1 = m_slidervalue1 + (buffer6 - m_slidervalue6);

		}
		else
		{
		m_slidervalue1 = buffer1;
		m_slidervalue2 = buffer2;
		m_slidervalue3 = buffer3;
		m_slidervalue4 = buffer4;
		m_slidervalue5 = buffer5;
		m_slidervalue6 = buffer6;
		}

	}

		m_value1 = 100-m_slidervalue1;
```

```
            m_value2 = 100-m_slidervalue2;
            m_value3 = 100-m_slidervalue3;
            m_value4 = 100-m_slidervalue4;
            m_value5 = 100-m_slidervalue5;
            m_value6 = 100-m_slidervalue6;

            UpdateData(FALSE);


        CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

void WeightDlg::OnOK()
{
        // TODO: Add extra validation here
    globalData.w_weight     = m_value1;
    globalData.w_speed_n = m_value2;
    globalData.w_speed_d = m_value3;
    globalData.w_key        = m_value4;
    globalData.w_spread  = m_value5;
    globalData.w_filesize= m_value6;

        CButton *Fast, *Slow;
        Fast = (CButton*)GetDlgItem(IDC_RADIO_FAST);
        Slow = (CButton*)GetDlgItem(IDC_RADIO_SLOW);

        if((Fast->GetCheck())==1)
        {
            globalData.c_fast = 1;
        }
        if((Slow->GetCheck())==1)
        {
            globalData.c_fast = 0;
        }

        CDialog::OnOK();  ///end
}

void WeightDlg::OnReset()
{
        // TODO: Add your control notification handler code here
        globalData.w_weight       = 100;
        globalData.w_speed_n    = 0;
        globalData.w_speed_d     = 0;
        globalData.w_key         = 0;
        globalData.w_spread      = 0;
        globalData.w_filesize    = 0;

        m_slidervalue1 = 100-globalData.w_weight;
        m_slidervalue2 = 100-globalData.w_speed_n;
        m_slidervalue3 = 100-globalData.w_speed_d;
        m_slidervalue4 = 100-globalData.w_key;
        m_slidervalue5 = 100-globalData.w_spread;
        m_slidervalue6 = 100-globalData.w_filesize;

        m_value1 = 100-m_slidervalue1;
        m_value2 = 100-m_slidervalue2;
        m_value3 = 100-m_slidervalue3;
        m_value4 = 100-m_slidervalue4;
        m_value5 = 100-m_slidervalue5;
        m_value6 = 100-m_slidervalue6;

        UpdateData(FALSE);
}
```