การนำสมาร์ทการ์ดมาประยุกต์ใช้งานในด้านต่างๆ

(THE SMART CARD IN APPLICATIONS)

โดย

นาย พนม ศลิษฎ์อรรถกร

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชา วิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การนำสมาร์ทการ์ดมาประยุกต์ใช้งาน

ผู้จัดทำ

1.นาย พนม ศลิษฎ์อรรถกร 36014279

(อาจารย์มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การนำสมาร์ทการ์ดมาประยุกต์ใช้งานในด้านต่างๆ

THE SMART CARD IN APPLICATIONS

1.นาย พนม ศลิษฏ์อรรถกร 36014279

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

(อาจารย์มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การนำสมาร์ทการ์ดมาประยุกต์ใช้งาน

นาย พนม ศลิษฎ์อรรถกร
อาจารย์ที่ปรึกษา รศ.ดร.มนัส สังวรศิลป์
ปีการศึกษา 2539

บทคัดย่อ

ปัจจุบันบัตรสมาร์ทการ์ดเริ่มเข้ามามีบทบาทแทนที่บัตรแม่เหล็กในงานที่ต้องการความปลอดภัยสูง เนื่องจากความไม่คงทนและความปลอดภัยต่ำของบัตรแม่เหล็กที่ก่อปัญหาทางด้านการโจรกรรมเงินในบัตร ,การปลอมแปลงบัตรและการสูญหายของข้อมูลในบัตร ทำให้ในประเทศต่างๆ นำระบบสมาร์ทการ์ดเข้ามาแก้ปัญหา

ปริญญานิพนธ์นี้เป็นการเขียนโปรแกรมเพื่อติดต่อสื่อสารข้อมูลในระบบสมาร์ทการ์ด และจัดการเรื่องระบบการรักษาความปลอดภัยของบัตร รวมทั้งการนำบัตรสมาร์ทการ์ดมาประยุกต์ใช้งานในลักษณะของบัตรประจำตัวนักศึกษา และ บัตรเงินสด

THE SMART CARD IN APPLICATIONS

Phanom Slisatkorn

Assoc.Prof.Dr.Manas Sangvorasil

Advisor

1996

Abstract

Now smart cards are comming to substitue magnetic cards in the applications which require high security. Because less of durability and security of the magnetic card take it to easy to counterfeit, copy or destroy the data in the card . So the smart card system is designed to reduce these problems.

This thesis is about the software to communicate the data in smart card system, manage the security in the card used in applications such as identification card, data card and electronic purse card for registry the subject courses.

สารบัญ

ภาคผนวก

กิตติกรรมประกาศ

สารบัญภาพ

บทที่ 1
บทนำ

1.1 วัตถุประสงค์ของโครงงาน

- เพื่อศึกษาถึงโครงสร้างและการทำงานของสมาร์ทการ์ดซึ่งเป็นเทคโนโลยีที่สำคัญในอนาคต
- เรียนรู้การเขียนโปรแกรมเพื่อควบคุมการทำงานและนำไปประยุกต์ใช้งาน
- กระตุ้นให้เกิดความสนใจในเทคโนโลยีสมาร์ทการ์ด รวมทั้งเป็นพื้นฐานในการนำไปพัฒนาต่อไป

1.2 ความเป็นมาของโครงงาน

สมาร์ทการ์ดเทคโนโลยีเป็นเทคโนโลยีที่มีความปลอดภัยของข้อมูลสูง เมื่อนำมาประยุกต์ใช้งานจะเป็นการช่วยอำนวยความสะดวกสบายแก่ผู้ใช้ (เช่นจับจ่ายด้วยเงินดิจิตอลในระบบออนไลน์) ช่วยประหยัดทรัพยากรธรรมชาติ(ลดการใช้กระดาษเก็บข้อมูลในรูปแบบข้อมูลดิจิตอล) และทรัพยากรมนุษย์ (การทำงานเช่นการซื้อขายด้วยเครื่องจักรอัตโนมัติ) และประโยชน์อื่นๆ

จากการเล็งเห็นความสำคัญของบัตรสมาร์ทการ์ดที่จะมีบทบาทอย่างมากในอนาคตและเนื่องจากโปรแกรมที่ใช้ในควบคุมการทำงานของสมาร์ทการ์ดและการประยุกต์ใช้งานยังต้องสั่งซื้อจากต่างประเทศ จึงมีการริเริ่มจัดทำโครงงานนี้ขึ้นเพื่อชี้ให้เห็นความสำคัญของเทคโนโลยีนี้และเป็นการส่งเสริมให้เกิดการพัฒนาของเทคโนโลยีนี้ภายในประเทศ

1.3 ขอบเขตของโครงงาน

- โปรแกรมการควบคุมการทำงานเบื้องต้นของสมาร์ทการ์ด
- โปรแกรมประยุกต์ใช้งานในลักษณะของบัตรประจำตัว, บัตรเงินสด และ บัตรสุขภาพ
- ฐานข้อมูล

1.4 ขั้นตอนการดำเนินงาน

แบ่งออกเป็น 2 ช่วงโดยในครึ่งแรกจะเป็นการสร้างต้นแบบของโปรแกรมมีรายละเอียดดังนี้

- ศึกษาและค้นคว้าข้อมูล
- ออกแบบและเขียนโปรแกรมในการควบคุมพื้นฐาน
- สร้างต้นแบบโปรแกรมการประยุกต์ใช้งาน

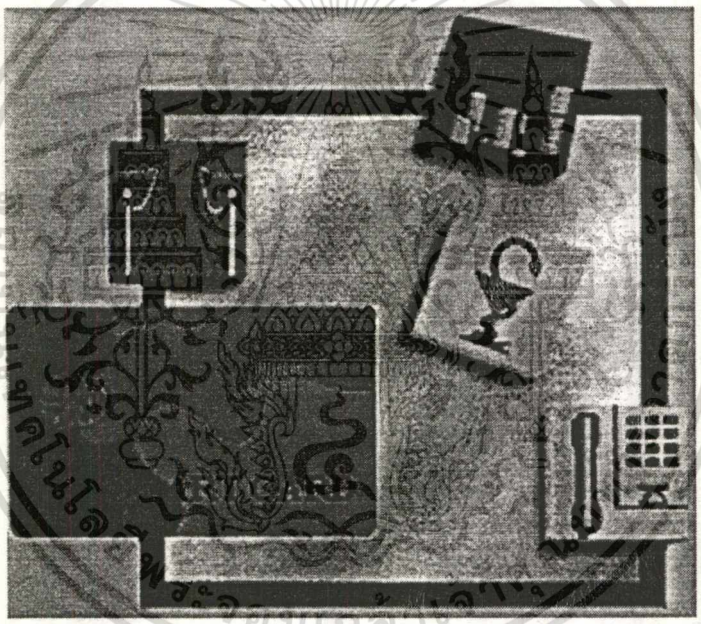
ช่วงที่ 2 จะเป็นการปรับปรุงต้นแบบโปรแกรมให้เป็นโปรแกรมที่ใช้งานจริง

- ประยุกต์โปรแกรมต้นแบบให้มีรายละเอียดตามการประยุกต์ใช้งาน
- จัดทำหน้าจอและโปรแกรมให้มีความสวยงามและง่ายต่อการใช้งาน

บทที่ 2
ทฤษฎีและรายละเอียดพื้นฐานของสมาร์ทการ์ด

2.1 สมาร์ทการ์ดคืออะไร ,มีกี่ประเภท

สมาร์ทการ์ด เป็น บัตรพลาสติกขนาดเท่าบัตรเครดิตหรือเอทีเอ็มที่มีหน่วยเก็บข้อมูลและหน่วยประมวลผล เรียกว่า ไมโครชิพ (microchip) หรือ ชิพ (chip) ติดอยู่บนบัตร ซึ่งข้อมูลนี้อาจจะอยู่ในรูปของตัวเลขหรือตัวอักษรก็ได้ โดยมีกลไกในการเขียน และการอ่านข้อมูลที่ซับซ้อนทำให้ยากต่อการปลอมแปลง จึงสามารถนำมาใช้ประโยชน์ในด้านต่างๆ เช่น ด้านการเงินการธนาคาร ด้านโทรศัพท์ โทรคมนาคม ด้านงานทะเบียน ด้านการแพทย์ ด้านการศึกษา และการรักษาความปลอดภัย เป็นต้น



รูปที่ 2.1 บัตรสมาร์ทการ์ด

สมาร์ทการ์ดเป็นบัตรประเภทหนึ่งของไมโครชิพการ์ด ซึ่งสำหรับไมโครชิพการ์ด สามารถแบ่ง ได้ 2 ประเภท คือ

1. Memory Card คือ การ์ดที่มีเฉพาะหน่วยความจำไม่มีหน่วยประมวลผล สามารถเข้าถึงข้อมูลในหน่วยความจำได้โดยตรง โดยไม่มีการควบคุมในการอ่านเขียนข้อมูลมากนัก เช่นบัตรโทรศัพท์

2. Microprocessor Card คือ การ์ดที่มีทั้งหน่วยความจำและหน่วยประมวลผลหรือซีพียู สามารถทำงานเสมือนไมโครคอมพิวเตอร์ ; นำข้อมูลเข้า ,ออก ,เก็บรักษาข้อมูล และประมวลผล

ข้อมูล; โดยไม่สามารถเข้าถึงข้อมูลภายในหน่วยความจำได้โดยตรง การเข้าถึงข้อมูลจะต้องผ่านหน่วยประมวลผลเท่านั้น ซึ่งมีการป้องกันการเข้าถึงข้อมูลโดยใช้กุญแจรหัสและการเข้ารหัสแบบต่างๆ ซึ่งการที่บัตรสามารถทำงานได้แบบนี้จึงถูกเรียกว่าบัตรสมาร์ทการ์ด (บัตรฉลาด)

สมาร์ทการ์ดถือกำเนิดมาจากประเทศฝรั่งเศส โดยอาจารย์ Roland Moreno เป็นผู้ประดิษฐ์ขึ้น เนื่องจากการที่ฝรั่งเศสมีการใช้บัตรพลาสติกในการจับจ่ายสินค้าสูงมาก รวมทั้งมีปัญหาการปลอมแปลง,การโจรกรรมและการสูญหายของข้อมูลในบัตรสูงเช่นกัน ซึ่งก่อให้เกิดความเสียหายทางเศรษฐกิจเป็นอย่างมาก จึงมีการประดิษฐ์บัตรที่มีความปลอดภัยสูงขึ้นมา

2.2 ข้อแตกต่างระหว่างการ์ดแถบแม่เหล็ก (magnetic stripe card) และ สมาร์ทการ์ด (smart card)

การ์ดแถบแม่เหล็ก

1. ความปลอดภัยต่ำ เนื่องจากบัตรแม่เหล็กสามารถปลอมแปลงโดยการใช้หัวเทปบันทึกข้อมูลได้ทันที และข้อมูลสามารถเข้าถึงได้โดยตรง การอ่าน, เปลี่ยนแปลง และ ทำลายข้อมูล จึงทำได้ง่าย
2. ใช้การเข้ารหัสข้อมูลแบบการEncoding ซึ่งสามารถถอดรหัสได้ง่าย
3. ความทนทานมีน้อย เพราะแถบแม่เหล็กง่ายแก่การรบกวนจากสภาพแวดล้อม
4. ในบัตรใบหนึ่งสามารถใช้บริการได้เพียงประเภทเดียว
5. เมมโมรี่มีขนาดจำกัด เพราะสามารถเก็บข้อมูลที่เข้ารหัสแล้วในแถบแม่เหล็กได้เพียง 1 กิโลบิตเท่านั้น
6. สามารถใช้เป็นเมมโมรี่การ์ดได้เท่านั้น
7. ไม่มีส่วน extended memory เพื่อรองรับบริการประเภทใหม่ๆที่จะมีได้

สมาร์ทการ์ด

1. มีความปลอดภัยสูง การเข้าถึงข้อมูลนั้น สามารถเข้าถึงได้โดยการผ่านซีพียูเท่านั้น จึงยากแก่การอ่าน, เปลี่ยนแปลง และทำลายข้อมูล
2. ใช้การเข้ารหัสข้อมูลแบบการEncription ซึ่งถอดรหัสได้ยากกว่าการEncoding
3. มีความทนทานสูง
4. สามารถใช้บริการได้หลายประเภทในบัตรเดียว
5. เมมโมรี่มีขนาดใหญ่ เพราะสามารถเก็บข้อมูลได้มากถึง 64 กิโลบิต และจะเพิ่มขึ้นกว่านี้อีกในอนาคต

6. มีความสามารถในการคำนวณ, en / decryption, สร้างจำนวนที่ได้จากการสุ่ม และอื่นๆ อีก

7. มีส่วน extended memory เพื่อรองรับบริการประเภทใหม่ๆในอนาคต เพราะเครื่องอ่าน (reader), เครือข่ายการทำงาน (networking) และอัลกอริทึม (algorithm) สามารถขยายเพิ่มได้

2.3 สมาร์ทการ์ดกับการประยุกต์ใช้งานในด้านต่างๆ

2.3.1 การประยุกต์ใช้งานด้านการสื่อสาร(The Smart Card Applications in telecommunications)

2.3.1.1ใช้ในการให้บริการโทรศัพท์สาธารณะ(PUBLIC PAYPHONES) เป็นการนำสมาร์ทการ์ดไปใช้ในรูปแบบบัตรโทรศัพท์ (phone card) ซึ่งได้รับการต้อนรับอย่างดีจากผู้ใช้ทั่วโลกเห็นได้จากตัวเลขอ้างอิงดังนี้

-มีใช้ใน 60 ประเทศทั่วโลกในปี 1993

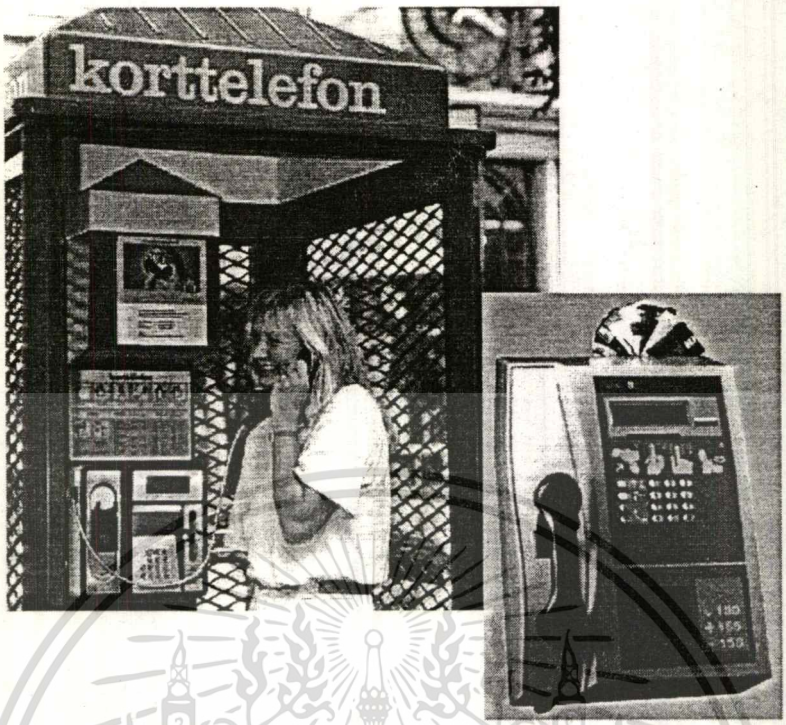
-มียอดจำหน่ายบัตรสูงถึง 100ล้านใบในฝรั่งเศสปี 1993

เนื่องจากมีข้อได้เปรียบบัตรโทรศัพท์ประเภทอื่นดังนี้

-มีความปลอดภัยสูง ถูกป้องกันโดยระบบความปลอดภัยในบัตรชนิดหน่วยความจำ และชนิดไมโครโปรเซสเซอร์ (มีข้อมูลเพิ่มเติมในบทระบบความปลอดภัยในบัตรสมาร์ทการ์ด)

-ในบัตรสมาร์ทมการ์ด1ใบสามารถใช้งานได้หลายประเภท หรืออาจจะกล่าวได้ว่าบัตรสมาร์ทการ์ดหลายประเภทสามารถใช้เป็นบัตรโทรศัพท์ได้ เช่นบัตรเงินสด(PREPAID CARD) บัตรเครดิต(CREDIT CARD) และบางประเภทสามารถใช้ได้ทั่วโลก ซึ่งเป็นการอำนวยความสะดวกแก่ผู้ใช้เป็นอย่างมาก

-มีราคาถูกเมื่อเทียบกับประโยชน์ที่ได้รับ (มีความเชื่อถือได้สูง,ระบบและเครื่องอ่านมีราคาถูก)



รูปที่ 2.2 สมาร์ทการ์ทกับการใช้งานในลักษณะบัตรโทรศัพท์

2.3.1.2 ใช้กับโทรศัพท์เคลื่อนที่(MOBILE PHONES) เป็นการใช้สมาร์ทการ์ดในลักษณะการเก็บเบอร์โทรศัพท์ของผู้ใช้ และรหัสผ่านลงในบัตร โดยเป็นการเอื้อประโยชน์แก่ผู้ใช้บัตรดังนี้

-มีความปลอดภัยสูง และมีระบบการล็อกเครื่องเมื่อกดรหัสผิด

-ใช้เป็นบัตรอ้างอิงเบอร์ของผู้ใช้เมื่อใช้งานกับเครื่องต่างเครื่องได้ เช่นเมื่อเกิดความเสียหายกับเครื่องโทรศัพท์ของตน สามารถนำบัตรไปใช้งานกับเครื่องอื่นๆได้โดยไม่ต้องเปลี่ยนเบอร์ หรือในกรณีการใช้งานในต่างประเทศ สามารถนำบัตรไปใช้งานกับเครื่องที่ต่างประเทศได้ โดยไม่ต้องนำเครื่องไปและไม่ต้องเปลี่ยนเลขหมายในการใช้งาน

รูปที่ 2.3 สมาร์ทการ์ดกับการประยุกต์ใช้ในโทรศัพท์เคลื่อนที่

รายชื่อประเทศที่มีการนำไปใช้งาน สำรวจในปี 1993

ALGERIA	ANDORRA	ARGENTINA	AZERBAIJAN
BURKINA FASO	CAMEROON	CENTRAL AFRICAN REP	CHILE
COLOMBIA	COMOROS	CONGO	COSTA RICA
CROATIA	CZECH REP	DENMARK	DJIBOUTI
EQ. GUINEA	FINLAND	FRANCE	FR POLYNESIA
GABON	GAMBIA	GERMANY	GHANA
GREECE	GUERNSEY	HUNGARY	ICELAND
INDIA	IRAN	IRELAND	LEBANON
LIBYA	LUXEMBURG	MALI	MALTA
MEXICO	MONACO	MOROCCO	NEW CALEDONIA
NIGERIA	NORWAY	PAKISTAN	PERU
PORTUGAL	RUMANIA	RUSSIA	SENEGAL
SLOVAQUIA	SOUTH AFRICA	SPAIN	ST MARTIN
SWEDEN	SWITZERLAND	TUNISIA	UNITED STATES
VANUATU	VENEZUELA	WALLIS&FUTUNA	

2.3.2 ด้านการเงิน การธนาคาร (Finance & Banking) เช่น

2.3.2.1 ใช้เป็นบัตรที่ใช้แทนเงินสด (cash card) เพราะให้ความสะดวกและความมั่นใจในการป้องกันการโกงและการปลอมแปลงบัตร ดังนั้นในต่างประเทศจึงใช้แทนเงินสดในการชำระค่าสินค้า หรือบริการที่มีมูลค่าไม่สูงนัก แต่จะเน้นประโยชน์ในด้านความสะดวกเป็นหลัก ซึ่งโดยทั่วไปแล้ว บัตรที่ใช้แทนเงินสดได้นี้สามารถแบ่งออกได้เป็น 2 ประเภท ได้แก่

- บัตรประเภทเติมเงินได้ หรือ Electronic Purse Card ที่สามารถเติมจำนวนเงินลงไปใหม่ได้เรื่อยๆ(refillable) คือ เป็นกระเป๋าเงินอิเล็คทรอนิก ซึ่งโดยมากจะออกโดยธนาคารเพื่อทำการโอนเงินไปมาได้ระหว่างบัญชีและบัตร

- บัตรประเภทเติมเงินไม่ได้ หรือ Token Card ที่ใช้หมดแล้วเติมไม่ได้อีกคือ ใช้ได้ครั้งเดียว (disposable) ที่นิยมกันมากในต่างประเทศได้แก่ บัตรโทรศัพท์ บัตรโดยสาร ฯลฯ

บัตรทั้งสองประเภทใช้ชิพชนิดเดียวกันหรือไม่ ?

ไมโครชิพบนบัตรจะต้องเหมาะกับการใช้งานคือ บัตรชนิดแรกจะใช้ชิพแบบไมโครโพรเซสเซอร์ (microprocessor) ที่มีหน่วยประมวลผลในตัวเองในตัวเองสามารถป้องกันการแก้ไขและการเข้าถึงข้อมูลได้ ส่วนบัตรชนิดที่สองจะใช้ชิพแบบหน่วยความจำธรรมดา หรือ เหมือนกับบัตรชนิดแรกก็ได้

2.3.2.2ใช้เป็นบัตรเครดิต(cardit card) เป็นการนำเอาสมาร์ทการ์ดไปใช้ในลักษณะบัตรเครดิต โดยใช้ร่วมกับแถบแม่เหล็ก และ hologram เพื่อทำให้เกิดความปลอดภัยยิ่งขึ้น

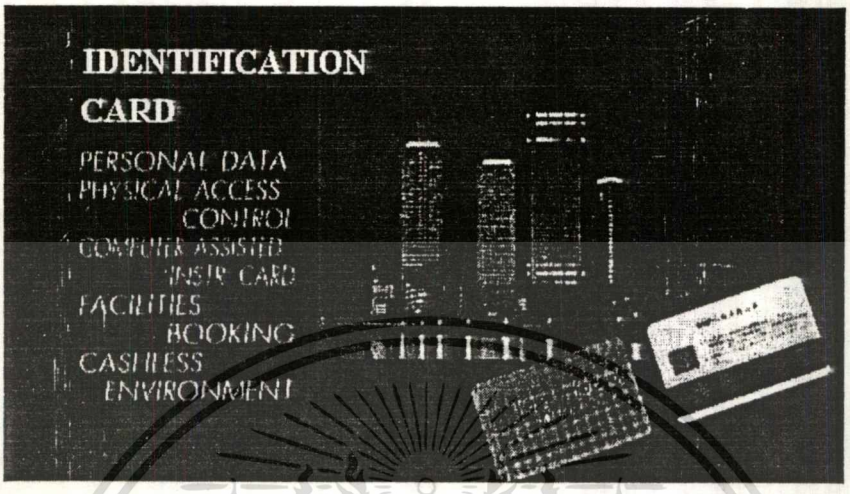
2.3.2.3 อื่นๆ เช่น -BANK POS SYSTEM

-HOME BANKING

-ASSET MANGEMENT

รูปที่ 2.4 การประยุกต์ใช้งานสมาร์ทการ์ดด้านการเงินการธนาคาร

2.3.3 ใช้เป็นบัตรประจำตัวต่างๆ (Identification Card) เช่น



รูปที่ 2.5 สมาร์ทการ์ดกับการใช้งานในลักษณะบัตรประจำตัว

2.3.3.1 บัตรพนักงาน (Company Card) ใช้เป็น

-บัตรอ้างอิงของพนักงาน เก็บประวัติส่วนตัว ,ประวัติสุขภาพและประวัติการทำงาน เวลาในการเข้า, ออกงาน

-บัตรเข้าออกในบางบริเวณ

-ใช้เป็นบัตรเงินในการจับจ่าย เช่น ค่าอาหาร ค่าบริการต่างๆ

2.3.3.2 บัตรนักศึกษา ใช้เป็นบัตรอ้างอิงของนักศึกษา เก็บประวัติการศึกษาและใช้บริการอื่นๆในสถานศึกษา เช่น ชำระค่าลงทะเบียน , การใช้บริการห้องสมุด ,การใช้บริการห้องคอมพิวเตอร์ ,เป็นบัตรผ่านเข้าหอพัก, ชำระค่าบริการบางประเภท เช่นการถ่ายเอกสารและการใช้บริการห้องอาหารเป็นต้น ซึ่งการใช้บัตรสมาร์ทการ์ดในงานต่างๆนั้นนอกจากจะช่วยอำนวยความสะดวกแล้ว ยังเป็นการช่วยลดการใช้เจ้าหน้าที่ในการปฏิบัติงานและเป็นการลดการใช้กระดาษในงานต่างๆ ซึ่งเป็นการช่วยประหยัดทรัพยากรธรรมชาติอีกทางหนึ่งด้วย

ตัวอย่างการใช้งานของสถานศึกษาต่างๆทั่วโลกในปี 1993

Univ. of South Australia	ใช้ระบบCAMMS System,ในลักษณะ Data Card
Univ. of Melbourne	ใช้ระบบCAMMS System,ในลักษณะ Data Card
Univ. of Calgary	มีการใช้Data Card ที่มีขนาดของEEPROM 2Kbyteจำนวน10,000ใบ

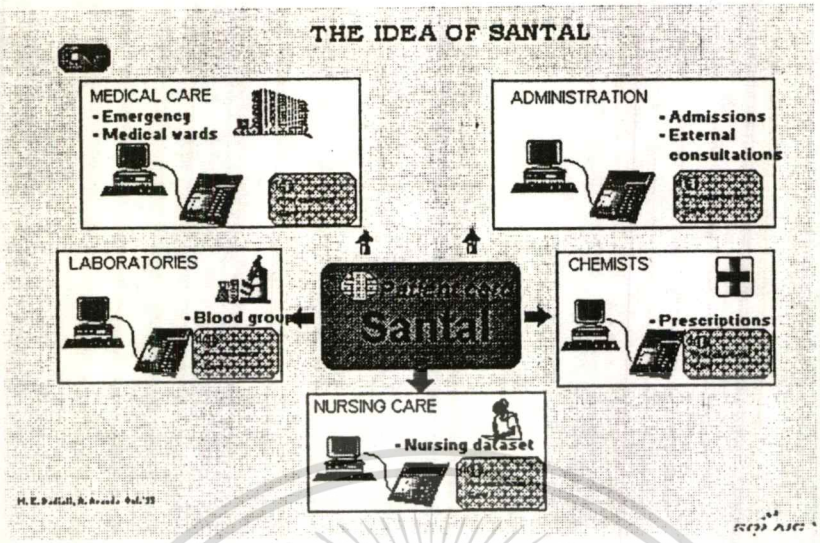
Univ. of Lille	ใช้ระบบ Card Sub โดยSligos,การ์ดของPhilips 45,000ใบ
Univ. of Rome	การ์ด CP8 300,000ใบ
Univ. of Floridia	การ์ดของ AT&T

และสถานศึกษาอื่น ๆในอเมริกาภว่า 150 สถานศึกษา

2.3.4 ด้านการแพทย์

รูปที่ 2.6 สมาร์ทการ์ดกับการใช้งานทางการแพทย์

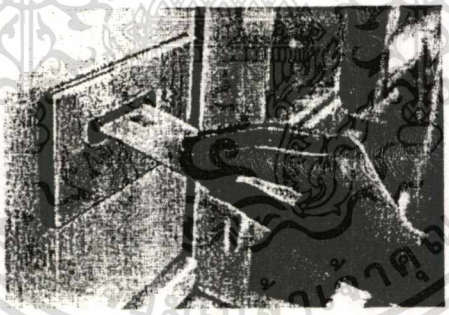
ใช้เป็นบัตรสุขภาพ (health card) ซึ่งจะมีประวัติการรักษาพยาบาล, การแพ้ยา เป็นต้น นอกจากนี้ยังมีประโยชน์คือใช้เป็นบัตรอ้างอิงตัวผู้ถือบัตรพร้อมประวัติทางสุขภาพในกรณีเกิดอุบัติเหตุฉุกเฉินได้



รูปที่ 2.7 ตัวอย่างระบบบัตรสุขภาพที่พัฒนาโดยบริษัท Solaic

2.3.5 สมาร์ทการ์ดกับระบบรักษาความปลอดภัย

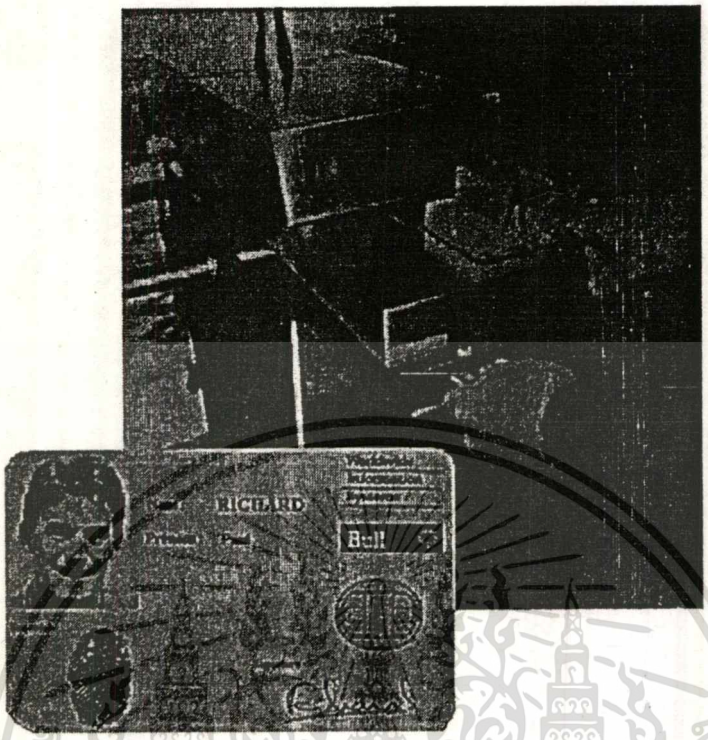
เป็นการนำเอาบัตรสมาร์ทการ์ดมาใช้ในการอ้างอิงผู้ถือบัตรและผ่านเข้าออกในพื้นที่ (access control)



รูปที่ 2.8 สมาร์ทการ์ดกับการใช้งานในรูปแบบของบัตรผ่านเข้าออกในพื้นที่

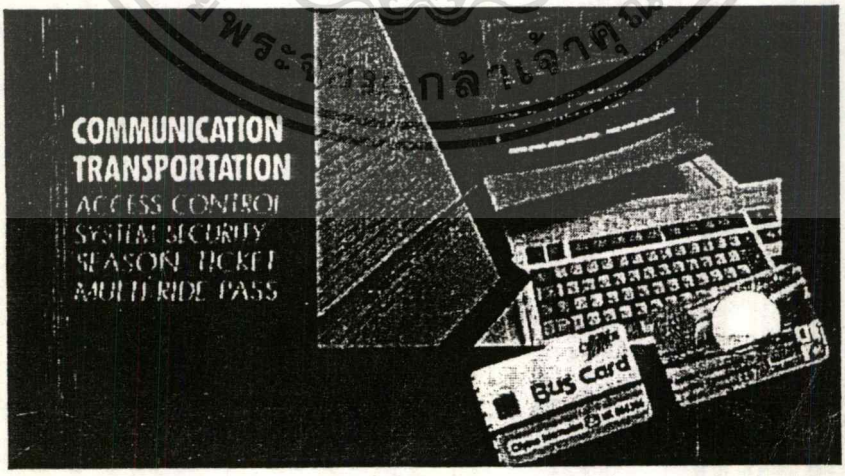
เช่นการนำไปใช้เป็นบัตรประจำห้องพักของโรงแรม ,บัตรผ่านของพนักงาน

สำหรับระบบที่ต้องการความปลอดภัยสูงมีการนำเอาสมาร์ทการ์ดไปใช้ร่วมกับการตรวจสอบลายนิ้วมือ



รูปที่ 2.9 การนำเอาสมาร์ทการ์ดไปใช้ร่วมกับการตรวจสอบลายนิ้วมือ

2.3.6 สมาร์ทการ์ดในส่วนของการขนส่งมวลชน (SMART CARD APPLICATIONS IN THE TRANSPORTATION SECTOR)

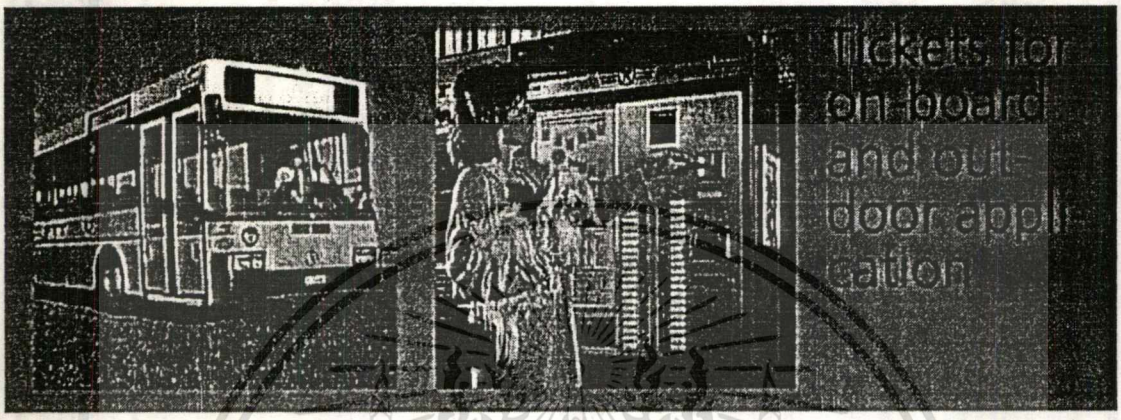


รูปที่ 2.10 สมาร์ทการ์ดกับการใช้งานในส่วนการขนส่งมวลชน

2.3.6.1 ในระบบการจอดรถ(parking systems) เป็นการใช้สมาร์ทการ์ดในการชำระเงินค่าจอดรถซึ่งเป็นตัวช่วยอำนวยความสะดวกในการชำระเงินโดยไม่ต้องมีขั้นตอนการทอนเงิน

2.3.6.2 ในระบบควบคุมการเข้าออกในบางบริเวณ (access control to some areas)

2.3.6.3 ใช้ในระบบการชำระค่าผ่านทางและค่าโดยสาร เช่น บัตรทางด่วน บัตรรถโดยสาร



รูปที่ 2.11 เครื่องขายบัตรโดยสารอัตโนมัติ

2.3.7 ด้านการบริการแก่ผู้บริโภคในการซื้อขายสินค้า (Customer Services Shopping)

รูปที่ 2.12 สมาร์ทการ์ดกับการใช้งานในการซื้อขายสินค้า

เป็นการนำเอาสมาร์ทการ์ดมาใช้ในรูปของบัตรสมาชิก ,บัตรเงินสด และบัตรเครดิต ทำให้ผู้ขายทราบข้อมูลการใช้บริการของผู้ถือบัตร ซึ่งเป็นประโยชน์อย่างมากในทางการตลาด และยังสามารถใช้ในการสะสมแต้ม การคืนกำไร

2.3.9 สมาร์ทการ์ดทีวี (smart card TV)

เป็นการนำสมาร์ทการ์ดมาใช้งานในระบบเคเบิ้ลทีวี, การรับส่งสัญญาณดาวเทียม โดยสัญญาณจะถูกเข้ารหัส ส่งผ่านสายเคเบิ้ลทีวีหรือดาวเทียมไปยังตัวถอดรหัส (decoder)ของผู้รับบริการ ซึ่งผู้รับบริการจะต้องมีบัตรสมาร์ทการ์ดที่มีรหัสที่จะใช้ในการถอดรหัสสัญญาณจึงสามารถรับชมภาพได้ มีการบรรจุข้อมูลเกี่ยวกับผู้ถือบัตรเช่นจำนวนช่องที่รับชมได้ นอกจากนี้อาจมีการบรรจุตัวเงินลงในบัตรเพื่อใช้ชำระค่าบริการตามเวลาที่รับชมและการจับจ่ายสินค้าทางเคเบิ้ลทีวีได้อีกด้วย ซึ่งระบบนี้เริ่มมีการใช้งานแล้วในยุโรป เช่น SKY CHANNEL , SKY MOVIES , CANNAL+ ,PREMIERE ,CANNAL SATELLITE และอื่น ๆ

รายชื่อประเทศที่มีการนำไปใช้งาน สำรวจในปี 1993

BELGIUM	CAMEROON	DENMARK	EGYPT
FINLAND	FRANCE	GABON	GERMANY
GUADELOUPE	IVORY COAST	LEBANON	NETHERLANDS
NORWAY	REUNION	SENGAL	SPAIN
SWEDEN	SWITZERLAND	TUNISIA	UNITED KINGDOM

รูปที่ 2.13 สมาร์ทการ์ดทีวี

2.3.10.ใช้ในการรักษาความปลอดภัยในระบบคอมพิวเตอร์

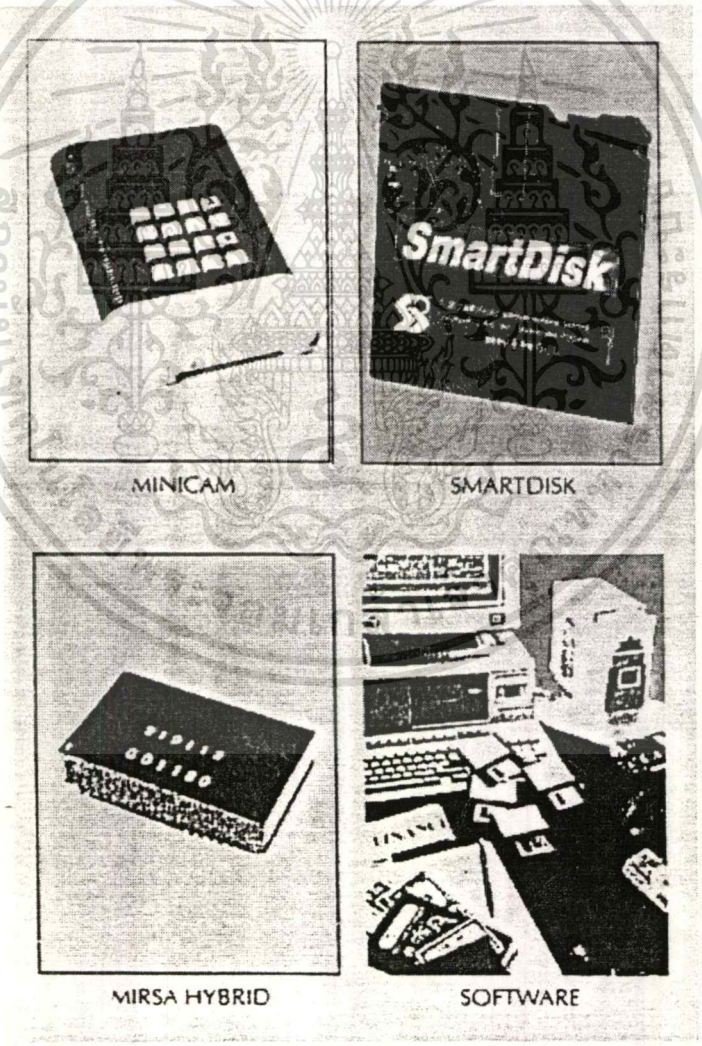
เป็นการนำสมาร์ทการ์ดมาประยุกต์ใช้ในด้านการรักษาความปลอดภัยในการสื่อสารข้อมูลระหว่างคอมพิวเตอร์ ,การเก็บข้อมูล และการเข้าถึงข้อมูล ซึ่งมีหลายรูปแบบเช่น

■ Minicam เป็นเครื่องที่ใช้รักษาความปลอดภัยในการรับส่งข้อมูล โดยจะมีการเข้ารหัสข้อมูลด้วยวิธีเอนคริ้ป (encript)ในการส่งข้อมูล โดยเครื่องที่จะรับข้อมูลได้จะต้องมีการถอดรหัสด้วยวิธีการดีคริ้ป(decript) แบบเดียวกัน และจะต้องมีรหัส (key) เท่านั้นจึงสามารถรับข้อมูลได้

■ Smart Disk เป็นสมาร์ทการ์ดที่อยู่ในรูปแบบของแผ่นดิสก์ขนาด 3 .5 นิ้วโดยในการอ่านเขียนข้อมูลจะต้องมีรหัสผ่านเท่านั้นจึงสามารถเข้าถึงข้อมูลได้

■ Mirsa เป็นชิปที่ใช้ในการเข้ารหัสแบบ RSA และ DES

■ Software เป็นsoftwareที่ใช้ในการเข้ารหัสและถอดรหัส



รูปที่ 2.14 สมาร์ทการ์ดกับการประยุกต์ใช้ในด้านการรักษาความปลอดภัยในการสื่อสารข้อมูล

2.4 ขั้นตอนการผลิตบัตรสมาร์ทการ์ด

การผลิตบัตรสมาร์ทการ์ดอาจแบ่งได้เป็น 2 ขั้นตอน

2.4.1 ขั้นตอนการออกแบบและผลิตบัตร Plastic card

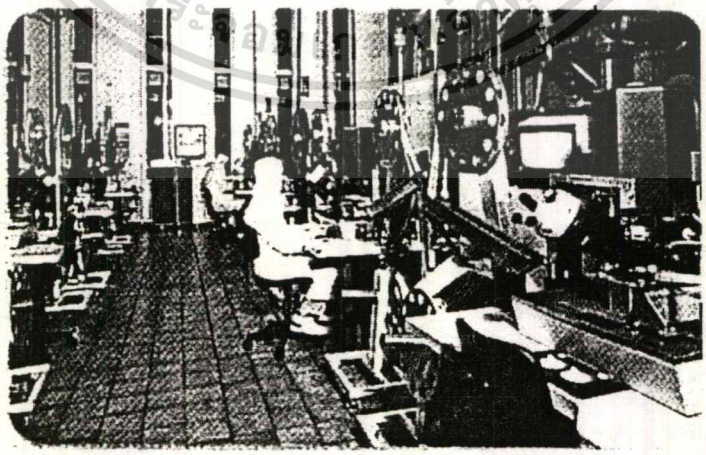
2.4.2 การผลิตสมาร์ทการ์ด

ขั้นตอนการออกแบบและผลิตบัตร Plastic card

1. ขั้นตอนการออกแบบพลาสติกการ์ดโดยใช้เครื่องคอมพิวเตอร์ช่วยในการออกแบบตามแนวความต้องการของลูกค้า หรือออกแบบใหม่เพื่อเสนอลูกค้า
2. ทำการ Pre-Print โดยใช้เครื่องพิมพ์ที่มีความละเอียดของสีใกล้เคียงกับงานพิมพ์จริง เพื่อให้ลูกค้าเห็นภาพเป็นแนวทางในการตัดสินใจ
3. ขั้นตอนการพิมพ์ และรวบรวมชิ้นงานเข้าด้วยกัน ซึ่งเป็นการพิมพ์โดยการแยกพิมพ์ด้านหน้าและด้านหลังเนื่องจากจะมีปริมาณของเสียน้อยกว่าการพิมพ์บนแผ่นเดียวกัน
4. ติดตั้งตัวเลือกต่าง ๆ ตามลูกค้าต้องการเช่น
   - แถบแม่เหล็ก (Magnatic Stripe)
   - ภาพโฮโลแกรม (Hologram)
   - แถบลายเซ็น
   - พิมพ์ตัวนูน
   - รูปถ่าย

การผลิตสมาร์ทการ์ด

ขั้นตอนการผลิตทุกขั้นตอนจะทำในห้องClean room

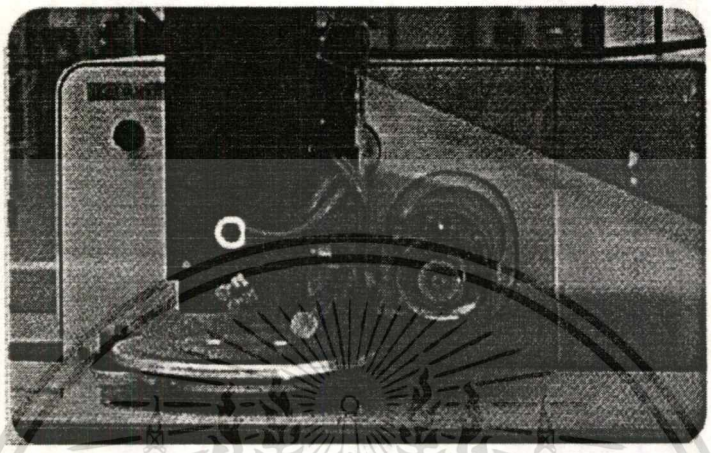


รูปที่ 2.15 ห้องClean room

1ใส่ข้อมูลและทดสอบเงื่อนไขต่างๆ (Data loading and conditioning)

-เป็นขั้นตอนการใส่ข้อมูลต่างๆ ลงในหน่วยความจำภายในชิพ

-ทดสอบการเขียนอ่านข้อมูล



รูปที่ 2.16 แสดงขั้นตอนการใส่ข้อมูลและทดสอบเงื่อนไขลงในชิพ

2 เจาะรูแผ่นพลาสติก (Drilling)

ขั้นตอนนี้มีความสำคัญมาก เนื่องจากถ้ารูเจาะไม่สมบูรณ์อาจทำให้ตำแหน่งชิพผิดพลาดได้

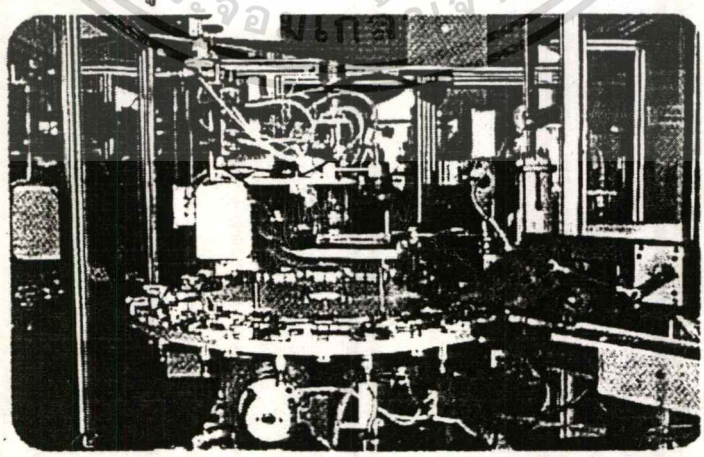
3 แปะชิพและทดสอบ (Embedding and test)

-ทดสอบการใช้งานของไมโครชิพ

-ตัดไมโครชิพ ที่ใช้งานได้ออกมาเป็นชิ้นที่พร้อมจะแปะเข้ากับรูเจาะ

-ทากาวที่รูเจาะ

-นำไมโครชิพมาแปะที่รู



รูปที่ 2.18 แสดงขั้นตอนในการแปะชิพ

บทที่ 3

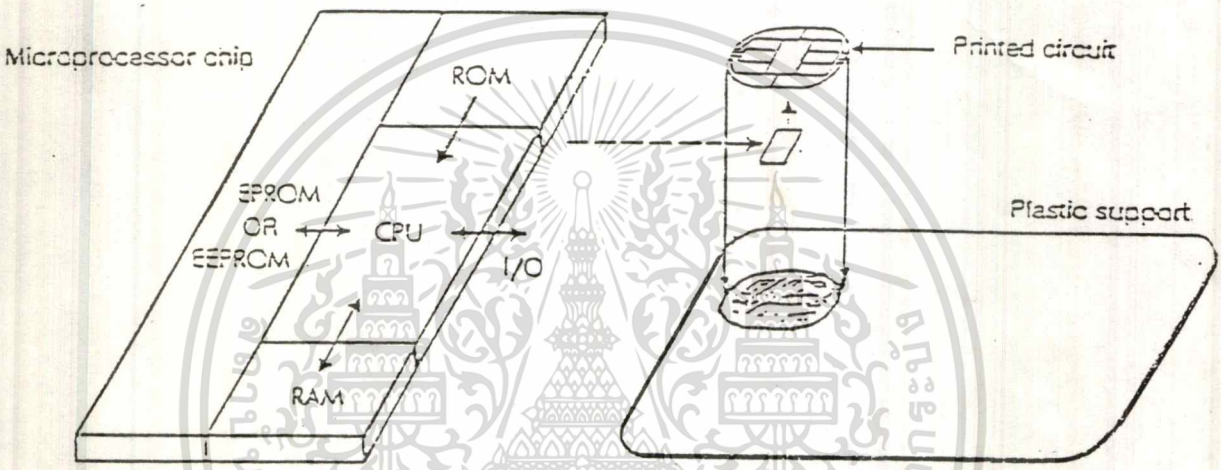
รายละเอียดภายในสมาร์ทการ์ด

3.1 ส่วนประกอบของบัตรสมาร์ทการ์ด

ประกอบด้วยส่วนประกอบ 3 ส่วนดังนี้

- แผ่นพลาสติก (plastic support)
- ลายวงจร (printed circuit)
- ชิป ทั้งเมมโมรี่และไมโครโปรเซอร์

ดังแสดงได้ดังรูป



รูปที่ 3.1 โครงสร้างของบัตรสมาร์ทการ์ด

แผ่นพลาสติกและลายวงจรจะต้องเป็นไปตามมาตรฐาน ISO 7810, 7816-1 และ 7816-2

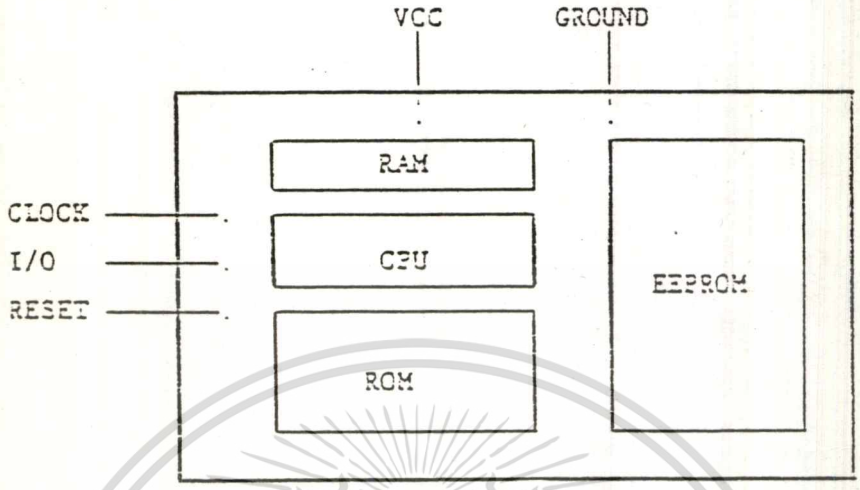
ไมโครโปรเซสเซอร์ชิป(Microprocessor chip)

ไมโครโปรเซสเซอร์ชิปที่ถูกผลิตมาแต่ละรุ่นแต่ละบริษัทจะมีส่วนประกอบและโครงสร้างที่แตกต่างกัน ในที่นี้จะยกตัวอย่างไมโครโปรเซอร์ชิปของBull TB100 SPOM21

สถาปัตยกรรม(Architecture)

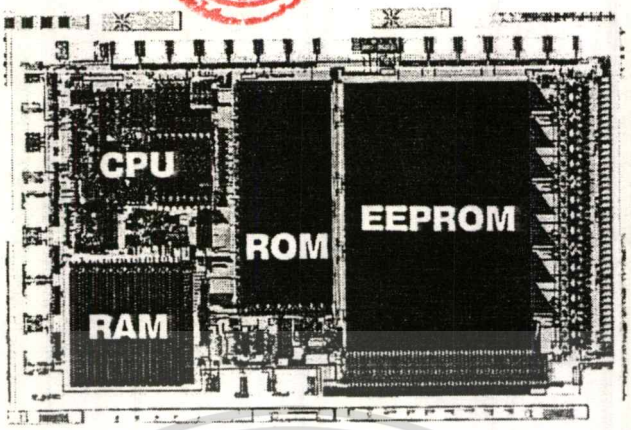
SPROM 21 ประกอบด้วย

- CPU (Central Processing Unit)
- RAM (Random Access Memory) ขนาด 128 ไบท์
- ROM (Read Only Memory) ขนาด 6 กิโลไบท์
- EEPROM (Electrical Erasable Programmable Memory) ขนาด 3 กิโลไบท์
- จุดเชื่อมต่อ 5 จุด ได้แก่ VCC ,Ground, Clock, Reset, I/O



รูปที่ 3.2 สถาปัตยกรรมภายในไมโครชิพ

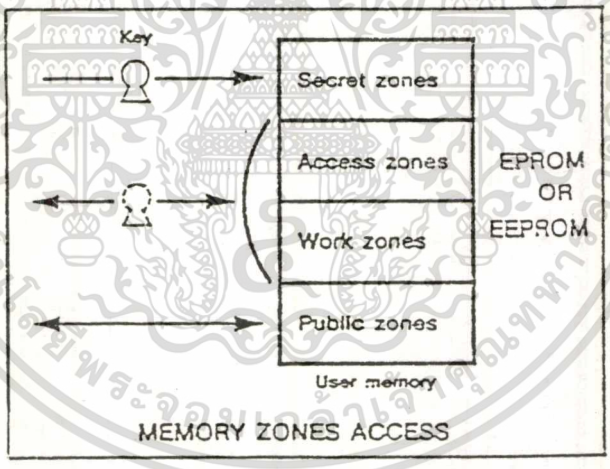
RAM	ถูกใช้เป็นรีจิสเตอร์ (register) หรือสำหรับเก็บข้อมูลชั่วคราว (temporary storage)
ROM	ใช้เก็บระบบดำเนินการ (Operating System)
EEPROM	ส่วนใหญ่เป็นพื้นที่ของผู้ใช้ไว้สำหรับเก็บข้อมูล และมีส่วนหนึ่งถูกสงวนไว้ สำหรับระบบดำเนินการเพื่อการจัดการหน่วยความจำ
CPU	ดำเนินการตามระบบดำเนินการ และ ควบคุมสายส่งข้อมูลภายใน (internal bus) อุปกรณ์ภายนอกไม่สามารถเข้าถึงข้อมูลในหน่วยความจำโดยตรงได้ ต้องผ่านทาง CPU
VCC	ต่อกับแหล่งจ่ายแรงดัน 5 โวลต์
Clock	ซิงโครไนต์ (synchronize) กับ ไมโครโปรเซสเซอร์ ความถี่ถูกกำหนดที่ 3.57 MHz สำหรับการติดต่อที่ความเร็ว 9600 bit/s
Reset	สัญญาณรีเซต 0 โวลต์ เป็นตัวกระตุ้นให้เข้าสู่ขั้นตอนแรกของการทำงาน โดยระบบดำเนินการจะส่งข้อความตอบรับการรีเซต (answer to reset) แล้วรอรับคำสั่ง
I/O	จุดเชื่อมต่อแบบอนุกรม ที่ความเร็ว 9600 bit/s สำหรับความถี่ 3.57 MHz โดยมีโปรโตรคอลในการสื่อสารเป็นไปตามมาตรฐาน ISO 7816-3



รูปที่ 3.3 ส่วนประกอบภายในไมโครชิพ

3.2 การจัดสรรพื้นที่ในหน่วยความจำ

หน่วยความจำสามารถแบ่งออกเป็นบริเวณต่างๆ ตามเงื่อนไขในการเข้าถึงได้ดังนี้



รูปที่ 3.4 พื้นที่ของหน่วยความจำแบ่งตามการเข้าถึง(memory zone access)

พื้นที่ลับ (secret zone) ข้อมูลที่เก็บในพื้นที่นี้จะถูกเก็บเป็นความลับเราไม่สามารถอ่านขึ้นมาได้ พื้นที่ส่วนนี้จะถูกเข้าถึงโดยไมโครโปรเซสเซอร์เท่านั้น

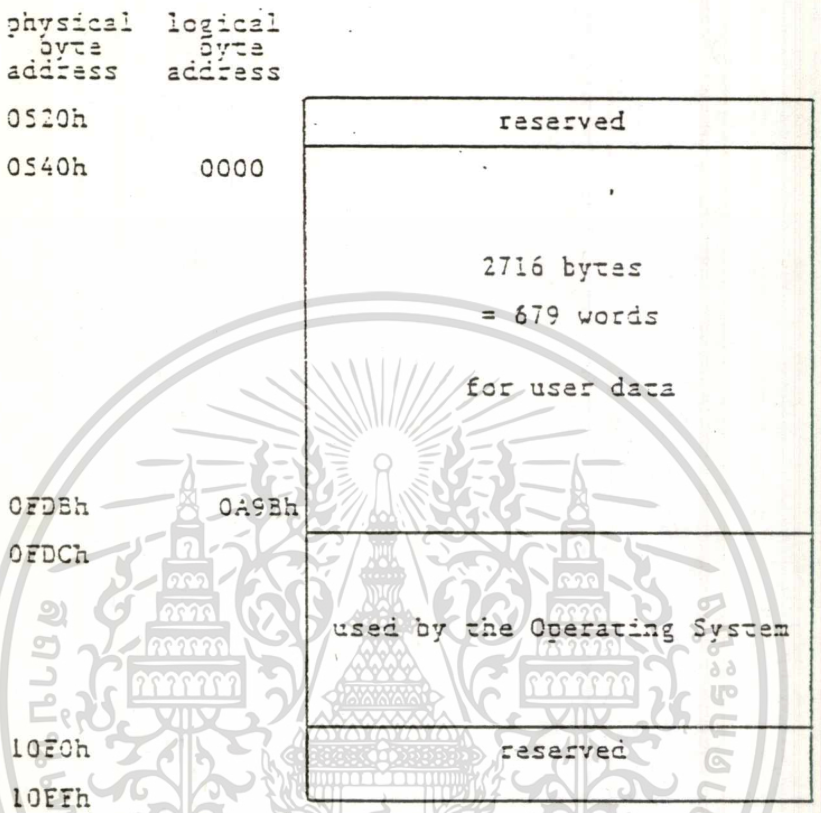
พื้นที่เก็บการเข้าถึง (access zone) พื้นที่ส่วนนี้จะใช้เก็บผลของการใส่กุญแจรหัส (ถูกต้องหรือไม่ถูกต้อง) โดยการใส่รหัสผิด 3 ครั้งจะเป็นการเพิ่มค่าของการใส่รหัสผิด ซึ่งจะทำให้บัตรถูก lock

พื้นที่ใช้งาน (working zone) ใช้เก็บข้อมูลในการประยุกต์ใช้งานต่างๆ ซึ่งสามารถกำหนดการป้องกันในการเข้าถึงหรือไม่ก็ได้

พื้นที่สาธารณะ (public zone) พื้นที่นี้สามารถเข้าถึงโดยไม่ต้องมีการใส่รหัส

การจัดสรรพื้นที่ของหน่วยความจำ

ยกตัวอย่างจากบัตร TB-100 พื้นที่ 3 กิโลไบต์ของ EEPROM ถูกจัดสรรโดยระบบดำเนินการดังรูป

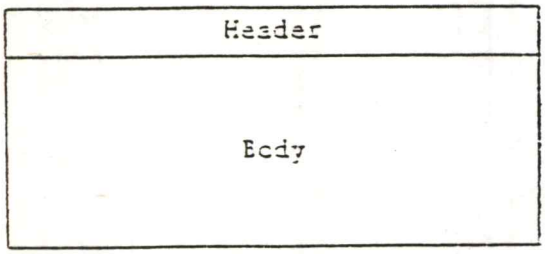


รูปที่ 3.5 การจัดสรรพื้นที่ในหน่วยความจำ (Memory Map)

3.3 โครงสร้างของข้อมูล

โครงสร้างของข้อมูลหรือบล็อกของหน่วยความจำประกอบด้วย

- หัวไฟล์ (header) เป็นส่วนที่ใช้เก็บข้อมูลเพื่อใช้ในการจัดการและอ้างอิงบล็อก เช่น ชื่อบล็อก เงื่อนไขในการเข้าถึง
- เนื้อหาภายใน (body) ใช้เก็บข้อมูลทั่วไปในการใช้งานต่างๆ



รูปที่ 3.6 โครสร้างข้อมูลในหน่วยความจำ

3.4โปรโตรคอลและชุดคำสั่ง (Protocol and Command set)

ในการเขียนโปรแกรมเพื่อนำระบบสมาร์ทการ์ดไปประยุกต์ใช้งาน ซึ่งต้องมีการเขียนโปรแกรมเพื่อติดต่อกันระหว่างเครื่องคอมพิวเตอร์ เครื่องอ่านบัตรสมาร์ทการ์ด และ ไมโครชิพในบัตรสมาร์ทการ์ด ซึ่งในแต่ละส่วนจะมีระบบดำเนินการ (Opreating System) มีโปรโตรคอล (Protocol)ในการแลกเปลี่ยนข้อมูลและชุดคำสั่งเป็นของตนเอง และนอกจากนี้ในเครื่องอ่านและการ์ดแต่ละชนิด แต่ละรุ่น หรือคนละบริษัทก็จะมีโปรโตรคอลและชุดคำสั่งที่แตกต่างกันอีกด้วย เช่น

โปรโตรคอลในการส่งข้อมูลให้เครื่องอ่านสมาร์ทการ์ด TLP224 ของ BULL CP8

<ACK><LN><MESSAGE><LRC>

ACK Previous block acknowledge

LN Length of data

LRC Longitudinal Redundancy Check

ซึ่งในการส่งคำสั่งนั้นจะต้องเปลี่ยนเป็น ASCII แล้วนำมาแตกเป็น 2 ไบต์ เช่น

ชุดคำสั่งASCII	60h		01h		4Dh		2Ch		03
แตกเป็น	36	30	30	31	34	44	32	43	03

ชุดคำสั่งของบัตรสมาร์ทการ์ด PocketBook V 3.1

- ASK RANDOM
- CHANGE KEY
- CREATE FILE
- DECREASE
- EXTERNAL AUTHENTICATION
- GET RESPONSE
- GIVE RANDOM
- INCREASE
- INVALIDATE
- READ BINARY
- READ RECORD
- SELECT FILE
- UPDATE BINARY
- UPDATE CEILING

- UPDATE RECORD
- VERIFY PIN
- WRITE BINARY
- WRITE RECORD

3.5 ระบบความปลอดภัย (Security)

ความปลอดภัยของข้อมูลในบัตรสมาร์ทการ์ดนอกจากจะห้ามไม่ให้เข้าถึงข้อมูลโดยตรงแล้ว ยังมีระบบควบคุมการเข้าถึงต่างๆ และการเข้ารหัสดังนี้

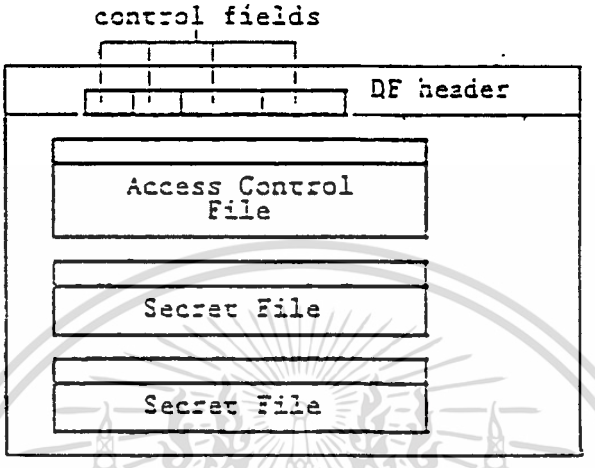
ระบบควบคุม (control systems)

ระบบควบคุมของไฟล์ เป็นสิ่งต่างๆ ที่ใช้ป้องกันไฟล์ ซึ่งในการ์ดแต่ละรุ่นจะมีระบบควบคุมที่ต่างกัน ส่วนใหญ่จะเป็นลักษณะของกุญแจในการทำงานต่าง ๆ และเงื่อนไขในการทำงานต่างๆ

ชนิดของกุญแจและรหัส (type of keys and codes)

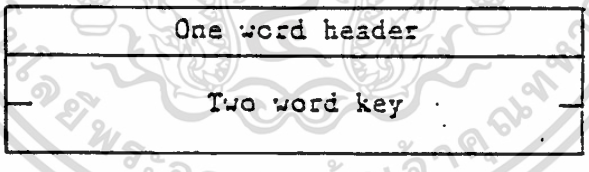
- manufacturing และ personalization keys
  - MK : manufacturer key
  - PK : personalization key
- issuer authentication keys
  - IK : issuer key
  - SK : secondary key หรือ secret keyสำหรับการ์ดบางชนิด
  - EK : erase key
- card and terminal authentication key
  - AK : authentication key
- cryptographic keys
  - GK : MAC generation key
  - VK : MAC verfication key
  - GVK : MAC generation and verification key
- cardholder authentication codes
  - PIN : personal identification number
  - AID : alternate identification code

โดยในการใช้งานจะอยู่ในรูปแบบของเงื่อนไขในหัวไฟล์ เช่น กำหนดว่าต้องแสดง IK และ PIN ก่อนจึงสามารถเขียนข้อมูลได้



รูปที่ 3.7 ส่วนประกอบของระบบควบคุม

กุญแจของ TB 100 ยกเว้น MKและ PK จะถูกเก็บไว้ในส่วนเนื้อหาของไฟล์ลับ โดยไฟล์ลับหนึ่งไฟล์จะเก็บกุญแจ 1 ดอกดังรูป



รูปที่ 3.8 โครงสร้างของไฟล์ลับ

การเข้ารหัส

ใช้ในการแสดงกุญแจรหัส โดยกุญแจรหัสจะถูกผสมกับตัวเลขแบบสุ่มที่การ์ดส่งมาให้แล้วจึงส่งค่าที่ได้ไปให้การ์ด สาเหตุที่ต้องมีการเข้ารหัสเพื่อป้องการการดักจับกุญแจรหัสระหว่างทาง การเข้ารหัสกับตัวเลขแบบสุ่มทำให้ค่าที่ส่งไปให้การ์ดในแต่ละครั้งไม่เหมือนกันขึ้นกับตัวเลขแบบสุ่มที่การ์ดส่งมาให้ ข้อมูลที่ส่งไปให้การ์ดจะถูกถอดรหัสเพื่อตรวจสอบว่าเป็นกุญแจรหัสที่ถูกต้องหรือไม่

DATA ENCRIPTION STANDARD

DES เป็นการเข้ารหัสข้อมูลที่ได้รับการเผยแพร่โดย National Bureau of Standard ประเทศสหรัฐอเมริกา(ปัจจุบันเปลี่ยนชื่อเป็น National Institute of Standards and Technology) ในปี 1977 โดยมีจุดม่งหมายเพื่อใช้ในการเข้ารหัสข้อมูลทางธุรกิจและข้อมูลของรัฐบาลที่ไม่เกี่ยวกับความลับระดับประเทศ อัลกอริทึมของ DES ดัดแปลงมาจากอัลกอริทึมLucifer Cipher ที่พัฒนาโดย IBM ในปี 1977 ซึ่งได้พัฒนาต่อร่วมกับ National Security Agency (NSA) ทำให้DES เป็นที่ยอมรับโดยทั่วไปและได้รับการยอมรับเป็นมาตรฐานของ American National Standards (ANSI) ในปี 1980 โดยในระยะเริ่มต้นมีข้อกำหนดสำหรับ DES ดังนี้

- มีระดับความปลอดภัยสูง
- มีการระบุรายละเอียดอย่างสมบูรณ์ และง่ายต่อความเข้าใจ
- ตัวอัลกอริทึมต้องมีความปลอดภัย และความปลอดภัยต้องไม่ขึ้นกับความปลอดภัยของอัลกอริทึม
- ผู้ใช้สามารถนำไปใช้และดัดแปลงสำหรับ Application ต่างๆได้
- ต้องมีประสิทธิภาพและประหยัดเพื่อนำไปใช้กับอุปกรณ์อิเล็กทรอนิกส์ได้
- ต้องสามารถตรวจสอบความถูกต้องได้

นิยามเบื้องต้น

Plaintext	ข้อมูลที่จะนำมาเข้ารหัส
Ciphertext	ข้อมูลที่ผ่านการเข้ารหัสแล้ว
Key	กุญแจข้อมูลขนาด 64 บิทที่ใช้ในการเข้ารหัสข้อมูล
Permutation	การสลับตำแหน่งของข้อมูลแต่ละบิท
Break DES	การหา Key ที่จะเปลี่ยน Plaintext เป็น Ciphertext

ภาพรวมของ DES

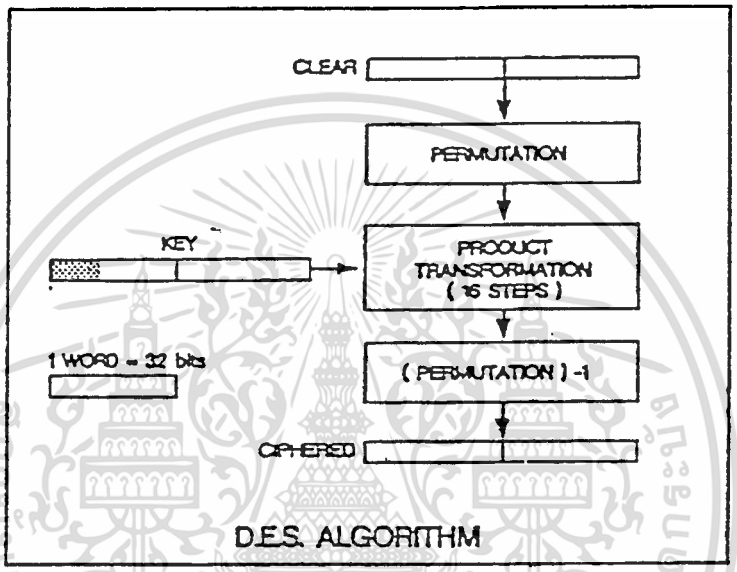
DES เป็นการนำข้อมูลขนาด 64 บิท(Plaintext) มากระทำการทั้งหมด 16 รอบด้วยเทคนิคของการเข้ารหัสที่ถูกกำหนดโดย key

DES กระทำการโดยนำข้อมูลขนาด 64 บิท มาสลับตำแหน่งข้อมูล (Initial Permutation) แล้วแบ่งข้อมูลเป็น 2 ชุดนำ เรียกชุดข้อมูลซ้าย L_i และชุดข้อมูลขวา R_i ขนาด 32 บิท แล้วนำมากระทำการทั้งหมด 16 รอบด้วยเทคนิคของการเข้ารหัสที่ถูกกำหนดโดย key โดยในการกระทำการแต่ละรอบแสดงเป็นสมการได้ดังนี้

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

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

หลังจากการกระทำการ16รอบ นำข้อมูลทั้ง2ชุดมารวมกันแล้วนำมาสลับตำแหน่ง(Final Permutation) ได้ผลลัพธ์เป็นข้อมูลที่ผ่านการเข้ารหัส (Ciphertext) ซึ่งอัลกอริทึมทั้งหมดสามารถแสดงเป็นขั้นตอนได้ดังรูป



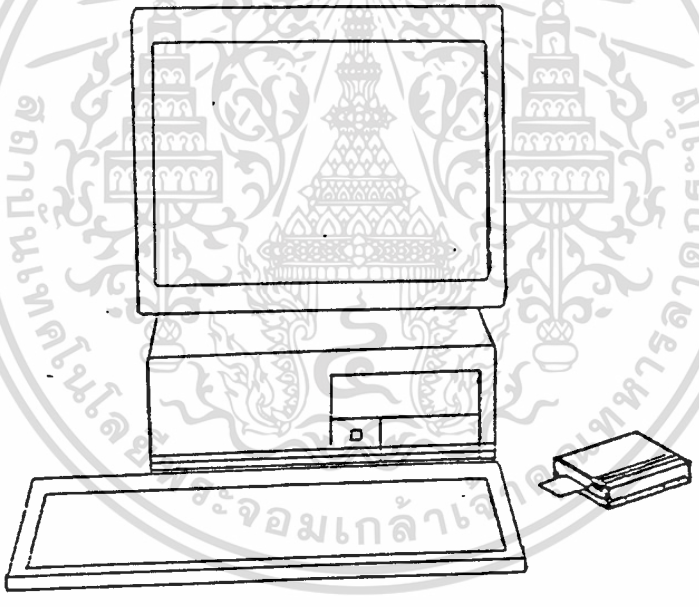
รูปที่ 3.9 การเข้ารหัสแบบ DES

บทที่ 4
โครงงาน

โครงงานนี้เป็นการเขียนโปรแกรมโดยใช้ภาษา C และ Visual Basicควบคุมการทำงานของสมาร์ทการ์ดเพื่อนำไปประยุกต์ใช้งาน โดยมีการทำงานเกี่ยวเนื่องกับส่วนประกอบ 3 ส่วนดังนี้

1. เครื่องคอมพิวเตอร์
2. เครื่องอ่านบัตรสมาร์ทการ์ด (Smart Card Terminal) ในโครงงานนี้ใช้เครื่องอ่านบัตรสมาร์ทการ์ด TLP 224 ของ BULL CP8
3. บัตรสมาร์ทการ์ด ในโครงงานนี้ใช้บัตรสมาร์ทการ์ดรุ่น PocketBook V3.1 ของSolaic

โดยการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์ และเครื่องอ่านบัตรสมาร์ทการ์ดใช้มาตรฐานการสื่อสารข้อมูล RS-232 ส่งแบบ Asynchronous ที่ความเร็ว 9,600 บอร์ด



รูปที่ 4.1 ส่วนประกอบระบบบัตรสมาร์ทการ์ด

โดยการเขียนโปรแกรมสามารถแบ่งออกเป็น 2 ส่วนใหญ่ ๆ ดังนี้

1. โปรแกรมย่อยในการสื่อสารข้อมูลในระบบและโปรแกรมย่อยของชุดคำสั่งต่างๆ เครื่องคอมพิวเตอร์ เครื่องอ่านบัตรสมาร์ทการ์ด และไมโครชิปในบัตรสมาร์ทการ์ด มีระบบดำเนินการ (Operating System) มีโปรโตรคอล(Protocal)ในการแลกเปลี่ยนข้อมูลและชุดคำสั่งของตนเอง ดังนั้นการสื่อสารข้อมูลในระบบสมาร์ทการ์ดจึงต้องมีการเขียนโปรแกรมเพื่อจัดการข้อมูลให้อยู่ในรูปแบบโปรโตรคอลที่กำหนด และชุดคำสั่งต่างๆของเครื่องอ่านบัตรสมาร์ทการ์ดและบัตรสมาร์ท

การ์ดอยู่ในรูป Opcode เลขฐาน 16 ซึ่งยากแก่การนำไปใช้งาน จึงต้องมีการเขียนโปรแกรมให้ชุดคำสั่งอยู่ในรูปของโปรแกรมย่อยที่ง่ายต่อการนำไปใช้งาน

2. โปรแกรมประยุกต์ใช้งาน เป็นโปรแกรมที่พัฒนาโดยการนำโปรแกรมย่อยจากข้อที่1 มาเขียนเป็นโปรแกรมเพื่อประยุกต์ใช้งานต่างๆ

4.1 โปรแกรมย่อยในการสื่อสารข้อมูลในระบบและโปรแกรมย่อยของชุดคำสั่งต่างๆ

4.1.1 โปรแกรมในการส่งข้อมูลตามโปรโตรคอล

โปรโตรคอลของเครื่องอ่านบัตรสมาร์ทการ์ด TLP224 มีลักษณะดังนี้

<ACK><LN><MESSAGE><LRC><ETX>

ACK Previous block acknowledge เป็นส่วนบอกจุดเริ่มต้นของบล็อกข้อมูล มีค่า 60h

LN Length of data ความยาวของข้อมูลรวมทั้งบล็อก

MESSAGE คำสั่งและข้อมูลที่จะส่ง

LRC Longitudinal Redundancy Check ไบต์ที่เป็นผลรวมของการXORของข้อมูลรวมทั้งบล็อก

ETX เป็นส่วนบอกจุดสิ้นสุดของบล็อก มีค่า 03h

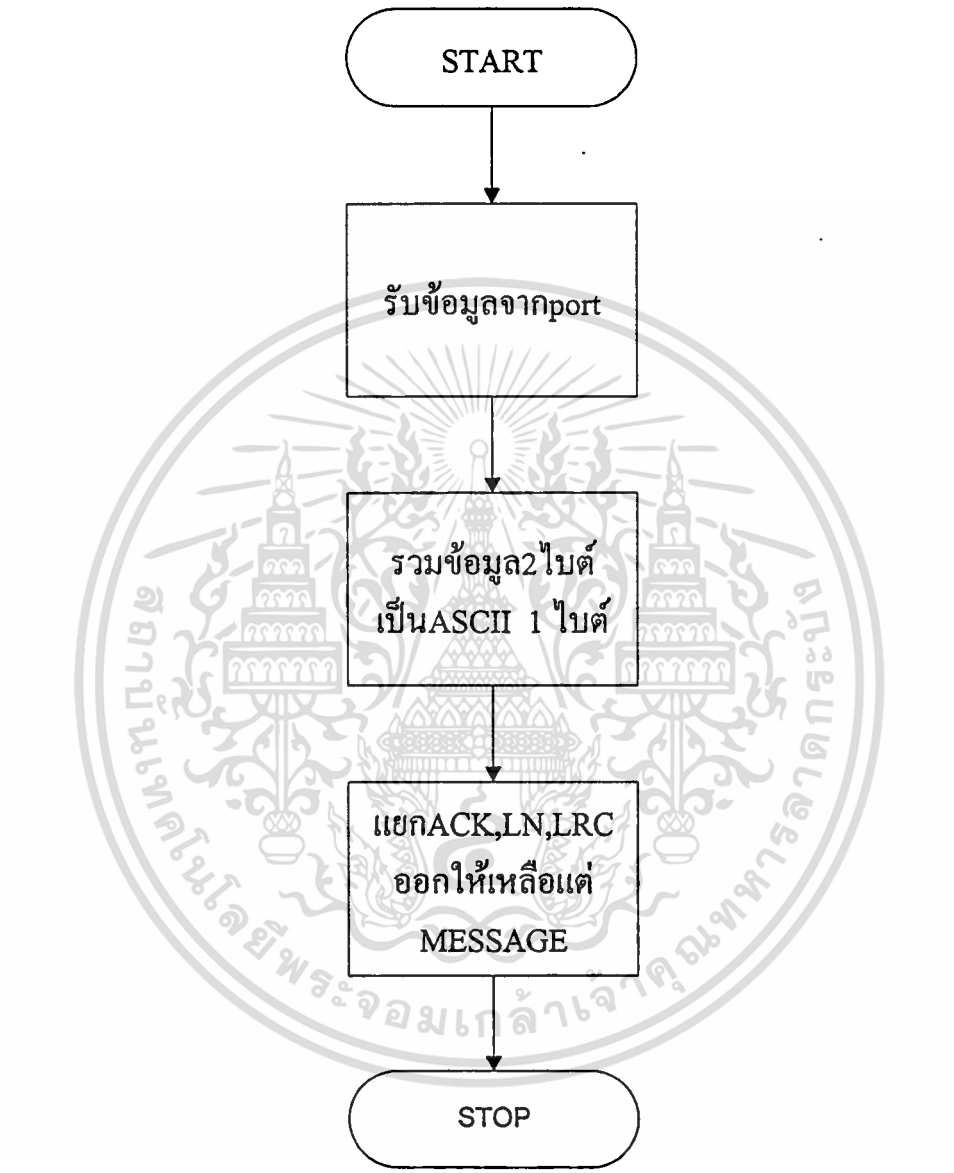
ซึ่งในการส่งคำสั่งนั้นจะต้องเปลี่ยนข้อมูลให้อยู่ในรูป ASCII code แล้วนำOctal codeแต่ละไบต์ (ยกเว้นETX) มาแตกเป็นASCII code 2 ไบต์เช่น

	ACK	LN	MESSAGE	LRC	ETX
ชุดคำสั่งASCII	60h	01h	4Dh	2Ch	03
แตกเป็น	36 30	30 31	34 44	32 43	03

รูปที่4.2 FLOWCHARTแสดงขั้นตอนของโปรแกรมในการส่งข้อมูลตามโปรโตรคอล

4.1.2 โปรแกรมในการรับข้อมูลตามโปรโตรคอล

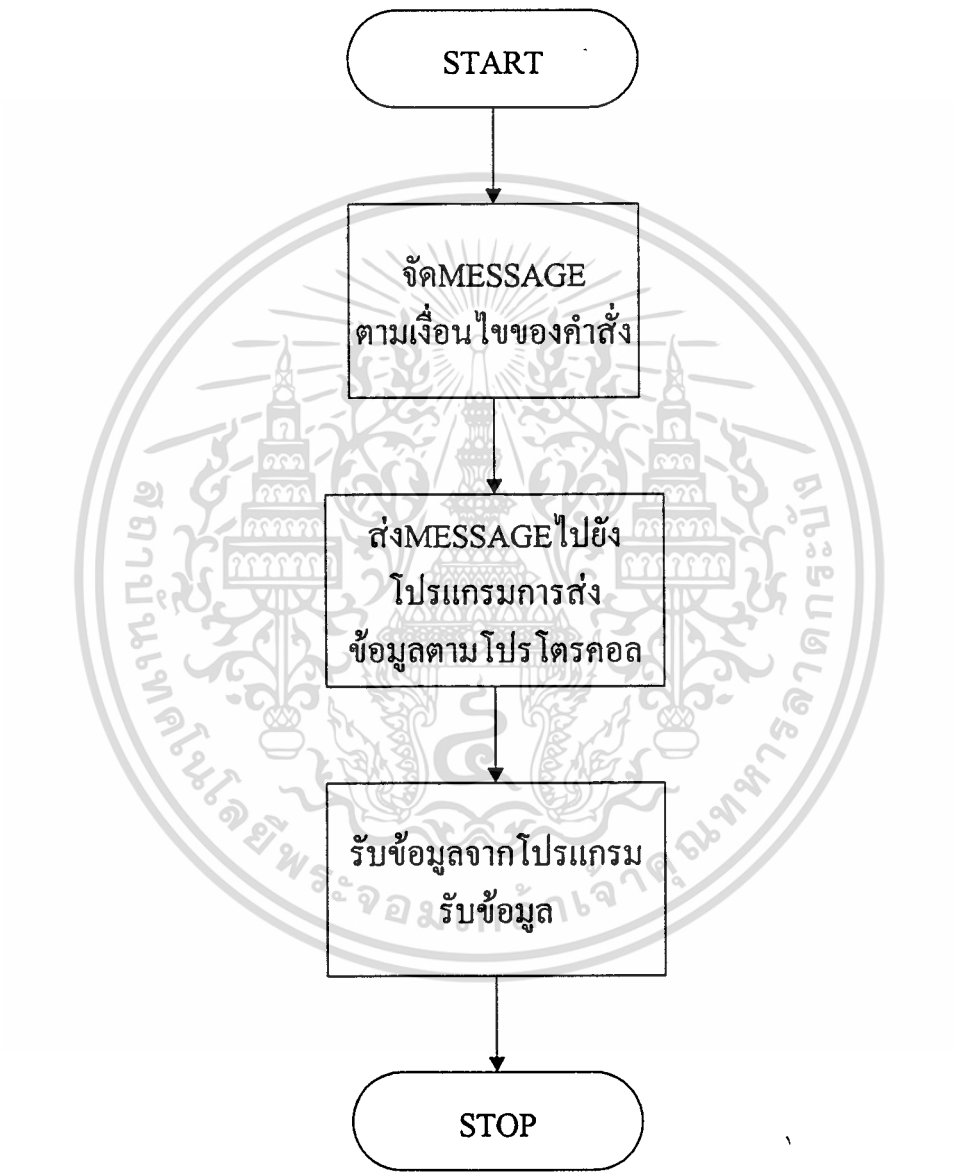
มีการทำงานกลับกับโปรแกรมในการส่งข้อมูล โดยมีขั้นตอนดังนี้



รูปที่4.3 FLOWCHARTแสดงขั้นตอนของโปรแกรมในการรับข้อมูลตามโปรโตรคอล

4.1.3 โปรแกรมย่อยของชุดคำสั่งต่างๆ

เป็นโปรแกรมที่นำค่าต่าง ๆ มาเรียงกันตามเงื่อนไขของคำสั่งที่ต้องการ แล้วส่งไปยังโปรแกรมในการส่งข้อมูลตามโปรโตรคอล และโปรแกรมในการรับข้อมูลถ้ามีการรับข้อมูลกลับจากเครื่องอ่านและบัตร



รูปที่4.4 FLOWCHARTแสดงขั้นตอนของโปรแกรมย่อยของชุดคำสั่งต่างๆ

ในการสั่งให้เครื่องอ่านและบัตรทำงานตามคำสั่งจะต้องมีการส่งรูปแบบ MESSAGE ดังนี้

<ความยาวของMESSAGE><ชุดคำสั่งของเครื่องอ่าน><ชุดคำสั่งของบัตร>

INSTR.	MEANING/REMARKS
6E	Power up (and/or reset) smart card.
4D	Power down smart card.
DA	"Transparent" incoming instruction. On receiving this instruction the TLP 224 NV sends to the smart card the bytes constituting the incoming instruction and then manages the exchanges according to ISO standard 7816-3.
DB	"Transparent" outgoing instruction. On receiving this instruction the TLP 224 NV sends to the smart card the bytes constituting the outgoing instruction and then manages the exchanges according to ISO standard 7816-3.
72	Read Phonecard.
BO	Write Phonecard.

รูปที่4.5 ชุดคำสั่งของเครื่องอ่านบัตรสมาร์ทการ์ด

Command	Opcode
ASK RANDOM	84h
CHANGE KEY	24h
CREATE FILE	E0h
DECREASE	30h
EXTERNAL AUTHENTICATION	82h
GET RESPONSE	C0h
GIVE RANDOM	86h
INCREASE	32h
INVALIDATE	04h
READ BINARY	B0h
READ RECORD	B2h
SELECT FILE	A4h
UPDATE BINARY	D6h
UPDATE CEILING	D6h
UPDATE RECORD	DCh
VERIFY PIN	20h
WRITE BINARY	D0h
WRITE RECORD	D2h

รูปที่4.6 ชุดคำสั่งของบัตรสมาร์ทการ์ด

Byte	Contents
Class	FAh
Command	84h
P1	not used
P2	not used
P3	08h
DATA	8-byte random number

รูปที่4.7 รูปแบบของคำสั่งให้การ์ดส่งตัวเลขแบบสุ่ม

รูปแบบคำสั่งอื่นๆดูจากหนังสืออ้างอิง

เช่นต้องการให้เครื่องสั่งให้บัตรสมาร์ทการ์ดหยุดทำงาน จากชุดคำสั่งได้op code ของคำสั่งเป็น 4Dh จะต้องต่อ MESSAGE ดังนี้

MESSAGE[0] = 1 ความยาวของชุดคำสั่ง

MESSAGE[1] = 4Dh คำสั่งของเครื่องอ่าน

ถ้าต้องการให้เครื่องอ่านส่งคำสั่งไปให้บัตรสมาร์ทการ์ดทำงานตามคำสั่งของบัตร

เช่นให้การ์ดส่งค่าตัวเลขแบบสุ่มออกมาจะต้องต่อ MESSAGE ดังนี้

MESSAGE[0] = 6 ความยาวของชุดคำสั่ง

MESSAGE[1] = DBh Instrของชุดคำสั่งของเครื่องอ่าน

MESSAGE[2] = FAh Classของชุดคำสั่งของบัตร

MESSAGE[3] = 84h Commandของชุดคำสั่งของบัตร

MESSAGE[4] = 00h P1 ของชุดคำสั่งของบัตร

MESSAGE[5] = 00h P2 ของชุดคำสั่งของบัตร

MESSAGE[6] = 08h P3 ของชุดคำสั่งของบัตร

จากขั้นตอนเหล่านี้จะได้เป็นโปรแกรมย่อยของชุดคำสั่งที่พร้อมจะเรียกใช้งานได้อย่างสะดวกดังนี้

โปรแกรมย่อยของชุดคำสั่งของเครื่องอ่านบัตรสมาร์ทการ์ด

- POWER_ON_CARD
- POWER_OFF_CARD

โปรแกรมย่อยของชุดคำสั่งของบัตรสมาร์ทการ์ด

- ASK RANDOM
- CHANGE KEY
- CREATE FILE
- DECREASE
- EXTERNAL AUTHENTICATION
- GET RESPONSE
- GIVE RANDOM
- INCREASE
- INVALIDATE
- READ BINARY
- READ RECORD
- SELECT FILE
- UPDATE BINARY
- UPDATE CEILING
- UPDATE RECORD
- VERIFY PIN

นอกจากนี้ยังมีโปรแกรมย่อยในการเข้ารหัสแบบDESซึ่งเป็นโปรแกรมมาตรฐานสามารถหาได้จากInternet

4.1.4 ขั้นตอนในการทำงาน

สมาร์ทการ์ดมีระบบในการเข้าถึงข้อมูลโดยการใช้กุญแจรหัสต่างๆ เช่นกุญแจรหัสในการเขียนข้อมูล กุญแจรหัสในการอ่านข้อมูล และมีการเข้ารหัสเพื่อป้องการการดักจับกุญแจรหัสระหว่างทาง โดยเข้ารหัสกับตัวเลขแบบสุ่มทำให้ค่าที่ส่งไปให้การ์ดในแต่ละครั้งไม่เหมือนกันขึ้นกับตัวเลขแบบสุ่มที่การ์ดส่งมาให้ ดังนั้นขั้นตอนในการทำงานจึงต้องมีการสั่งให้การ์ดส่งตัวเลขแบบสุ่ม, การเข้ารหัสกับตัวเลขแบบสุ่มและการแสดงรหัสตามเงื่อนไขก่อนการเข้าถึงข้อมูลใด ๆ

การแสดงรหัส(Present Key)

- สั่งให้การ์ดส่งตัวเลขแบบสุ่มโดยคำสั่ง ASK RANDOM
- นำตัวเลขแบบสุ่มมาเข้ารหัสกับกุญแจรหัส ด้วยคำสั่ง DES
- ส่งกุญแจที่เข้ารหัสแล้วไปให้การ์ดโดยคำสั่ง EXTERNAL AUTHENTICATION

การเปลี่ยนรหัส (Change Key)

ก่อนการทำงานคำสั่งนี้ต้องมีการแสดงกุญแจรหัสเดิมด้วยขั้นตอนในการแสดงรหัสข้างต้น

- สั่งให้การ์ดส่งตัวเลขแบบสุ่มโดยคำสั่ง ASK RANDOM
- นำตัวเลขแบบสุ่มมาเข้ารหัสกับกุญแจรหัส ด้วยคำสั่ง DES
- เปลี่ยนเป็นกุญแจรหัสใหม่โดยนำกุญแจใหม่ที่เข้ารหัสแล้วมาส่งให้การ์ดโดยคำสั่งCHANGE KEY

การสร้างไฟล์(Create file)

- ก่อนจะสร้างไฟล์จะต้องมีการแสดงกุญแจรหัส IK ก่อนโดยใช้ขั้นตอนในการแสดงรหัสข้างต้น
- สร้างไฟล์โดยใช้คำสั่ง CREATE FILE

โดยขั้นตอนในการสร้างไฟล์นี้จะมีการกำหนดชนิดของไฟล์และเงื่อนไขในการเข้าถึงข้อมูลในไฟล์ด้วยว่าในการเขียน อ่าน หรือเปลี่ยนแปลงข้อมูลจะต้องมีการแสดงกุญแจรหัสใดจึงสามารถกระทำได้

การเข้าถึงข้อมูลในไฟล์

- เลือกไฟล์ที่จะเข้าถึงโดยคำสั่ง SELECT FILE
- แสดงกุญแจรหัสตามเงื่อนไขในการเข้าถึงที่กำหนดไว้ โดยใช้ขั้นตอนในการแสดงรหัส
- ใช้คำสั่งในการอ่านเขียนข้อมูลและเปลี่ยนแปลงข้อมูลตามชนิดของไฟล์ดังนี้

ไฟล์ชนิดTransparent file

- READ BINARY
- WRITE BINARY
- UPDATE BINARY

ไฟล์ชนิด Record file

- READ RECORD
- WRITE RECORD
- UPDATE RECORD

ไฟล์ชนิด Electronic purse file

- INCREASE
- DECREASE
- UPDATE CEILING

ณ จุดนี้เราทราบขั้นตอนในการทำงานทั้งหมดและมีโปรแกรมย่อยต่างๆ ที่พร้อมจะนำไปใช้งานในการเขียนโปรแกรมเพื่อประยุกต์ใช้งาน

4.2 โปรแกรมประยุกต์ใช้งาน

โครงงานนี้เป็นการนำเอาสมาร์ทการ์ดมาประยุกต์ใช้งานในด้านการลงทะเบียนรวมทั้งการจ่ายค่าลงทะเบียนอัตโนมัติ ซึ่งจะเป็นการช่วยอำนวยความสะดวก,ลดการใช้เจ้าหน้าที่ในการปฏิบัติงานและช่วยให้การลงทะเบียนเป็นไปได้อย่างรวดเร็ว นอกจากนี้ยังสามารถนำข้อมูลไปใช้ทางด้านการเงินการทะเบียนและบัญชีต่างๆได้อีกด้วย ซึ่งในการนำบัตรสมาร์ทการ์ดมาลงทะเบียนจะต้องมีการจัดสรรพื้นที่ในบัตรและคุณสมบัติดังนี้

- สร้างTransparent file สำหรับเก็บข้อมูลส่วนตัวของผู้ถือบัตร เป็นบัตรประจำตัวเพื่อบ่งชี้เจ้าของบัตร (ทำงานในลักษณะ Identification Card) โดยมีการกำหนดเงื่อนไขในการอ่านเขียนดังนี้

เงื่อนไขในการเขียน (access to write)	DK
เงื่อนไขในการเปลี่ยนแปลงแก้ไขข้อมูล (access to update)	DK
เงื่อนไขในการอ่านข้อมูล (access to read)	free

- สร้างTransparent file สำหรับเก็บข้อมูลด้านการลงทะเบียน (ทำงานในลักษณะ Data Card) โดยมีการกำหนดเงื่อนไขในการอ่านเขียนดังนี้

เงื่อนไขในการเขียน (access to write)	DK
เงื่อนไขในการเปลี่ยนแปลงแก้ไขข้อมูล (access to update)	DK
เงื่อนไขในการอ่านข้อมูล (access to read)	free

- สร้างElectronic Purse File สำหรับเก็บจำนวนเงินในบัตร (ทำงานในลักษณะของ Electronic Purse Card) โดยมีการกำหนดเงื่อนไขในการอ่านเขียนดังนี้

เงื่อนไขในการเพิ่มเงิน (access to increte)	DK
เงื่อนไขในการใช้เงิน (access to decrete)	PIN
เงื่อนไขในการอ่าน (access to read)	free

โดย DK จะเป็นกุญแจรหัสที่เก็บโดยผู้ออกบัตร

PIN จะเป็นกุญแจรหัสที่เก็บโดยถือบัตร

free เข้าถึงได้อิสระโดยไม่ต้องมีการแสดงกุญแจใด ๆ

โปรแกรมในการลงทะเบียน

มีขั้นตอนในการทำงานดังนี้

รูปที่4.8 ขั้นตอนในการลงทะเบียน

โปรแกรมสำหรับผู้ออกบัตรประกอบด้วย

- โปรแกรมสำหรับสร้างไฟล์ต่างๆ ตามเงื่อนไขที่ต้องการ ดูขั้นตอนและรายละเอียดจากขั้นตอนในการสร้างไฟล์
- โปรแกรมในการกำหนดค่าเริ่มต้นต่างๆ เช่น ข้อมูลของผู้ใช้บัตร จำนวนเงินสูงสุดที่มีได้ในบัตร ดูขั้นตอนและรายละเอียดจากขั้นตอนในการเข้าถึงข้อมูล
- โปรแกรมในการปลดล็อกบัตร เนื่องจากระบบรักษาความปลอดภัยของสมาร์ทการ์ดจะทำการล็อกบัตรเมื่อมีการแสดงกุยแจรหัสผิด 3 ครั้ง ดูขั้นตอนและรายละเอียดจากขั้นตอนในแสดงกุญแจรหัส โดยกุญแจรหัสที่ต้องแสดงในการปลดล็อก คือ Unlock key(UK)
- โปรแกรมในการเติมเงิน ดูขั้นตอนและรายละเอียดจากขั้นตอนในการเข้าถึงข้อมูล

4.3 วิธีการใช้งาน

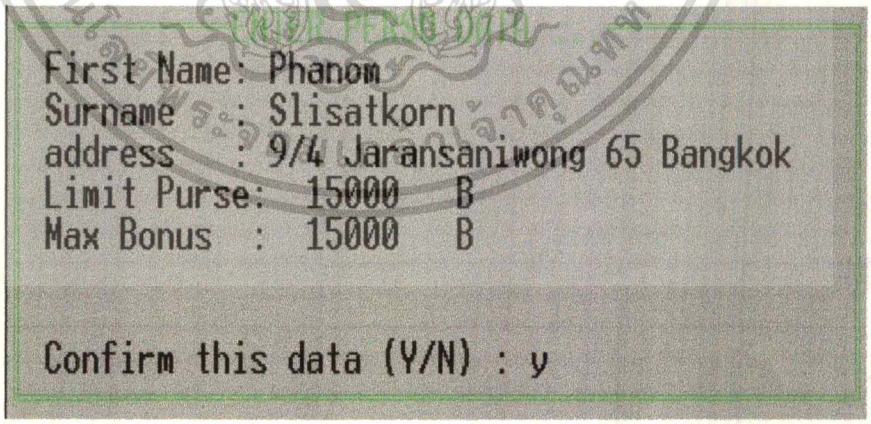
4.3.1 โปรแกรมการใช้งานสำหรับผู้ออกบัตร

รูปที่ 4.9 หน้าตาของโปรแกรมสำหรับผู้ออกบัตร

โดยแต่ละตัวเลือกทำหน้าที่ดังนี้

■ **Personalize** เป็นขั้นตอนแรกของการออกบัตร โดยขั้นตอนนี้จะเป็นขั้นตอนในการสร้างไฟล์ กำหนดเงื่อนไขในการเข้าถึงต่าง ๆ (กุญแจรหัส) และใส่ข้อมูลเริ่มต้นดังนี้

-เมื่อเลือกตัวเลือก Personalize จะปรากฏหน้าจอดังรูปที่ 4.10 เพื่อให้ข้อมูลเริ่มต้นของบัตร เมื่อใส่ข้อมูลเรียบร้อยทำการยืนยันข้อมูลโดยใส่ Y ในช่อง Confirm this data



รูปที่ 4.10 หน้าต่างสำหรับใส่ข้อมูลเริ่มต้นของบัตร

- จะปรากฏหน้าต่างให้ใส่ กุญแจรหัส ดังรูปที่ 4.11

รูปที่ 4.11 หน้าต่างในการใส่ KEY

-ใส่กุญแจรหัส เป็นอันเสร็จสิ้นขั้นตอนในการpersonalize

■ **Create file & Initial data** เป็นขั้นตอนการสร้างไฟล์ข้อมูลเกี่ยวกับนักศึกษาในการลงทะเบียน

- เลือกตัวเลือกCreate file & Initial data จะปรากฏหน้าจอให้ใส่ข้อมูลดังรูป 4.12

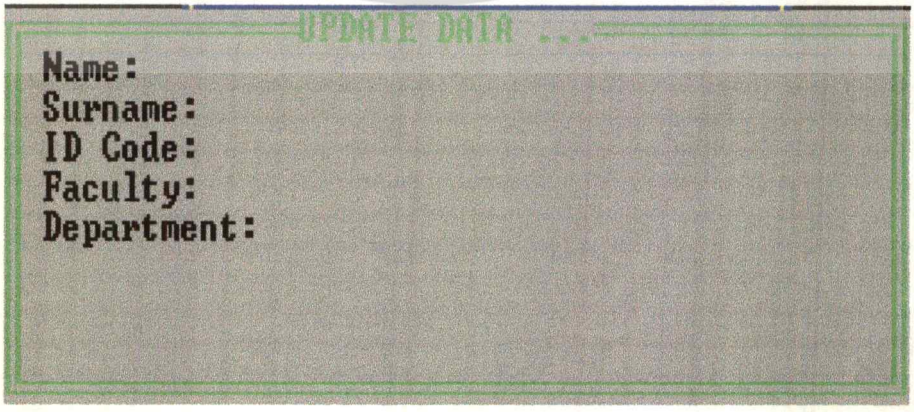
รูปที่ 4.12 หน้าต่างสำหรับใส่ข้อมูลเริ่มต้นของบัตร

- เมื่อใส่ข้อมูลเรียบร้อยจะปรากฏหน้าจอให้ใส่กุญแจรหัสเช่นเดียวกับรูป 4.11

-ใส่กุญแจรหัส เป็นอันเสร็จสิ้นขั้นตอนในการ Create file & Initial data

■ **Update data** เป็นขั้นตอนการแก้ไขข้อมูลภายในบัตร

- เลือกตัวเลือก Update data จะปรากฏหน้าจอให้ใส่ข้อมูลดังรูป 4.13



รูปที่ 4.13 หน้าต่างสำหรับใส่ข้อมูลใหม่

- เมื่อใส่ข้อมูลเรียบร้อยจะปรากฏหน้าจอให้ใส่กุญแจรหัสเช่นเดียวกับรูป 4.11
-ใส่กุญแจรหัส เป็นอันเสร็จสิ้นขั้นตอนในการ Update data

■ **Display & Operate** เป็นคำสั่งที่สั่งให้แสดงข้อมูลเกี่ยวกับนักศึกษาและรหัสวิชาที่ลงทะเบียน เมื่อเลือกตัวเลือกนี้จะปรากฏหน้าจอและข้อมูลที่อยู่ในบัตรดังรูป

รูปที่ 4.14 หน้าต่างแสดงข้อมูลในบัตร

■ **Unlock Pin** เป็นคำสั่งในการปลดล็อกบัตร ใช้ในการปลดล็อกบัตรเมื่อบัตรถูกล็อกเนื่องจากมีการแสดงกุญแจรหัสผิด 3 ครั้ง

■ **Credit Purse** เป็นตัวเลือกในการเติมเงินลงในบัตร เมื่อคลิกตัวเลือกนี้จะปรากฏหน้าจอแสดงข้อมูลเกี่ยวกับบัตร, จำนวนเงินสูงสุดที่เก็บได้, จำนวนเงินคงเหลือ และรายการในการใช้จ่ายเงินดังรูปที่ 4.15

รูปที่ 4.15 หน้าจอในการเติมเงิน

- เมื่อใส่จำนวนเงินที่จะเพิ่มเสร็จจะปรากฏหน้าจอให้ใส่กุญแจรหัสเช่นเดียวกับรูป 4.11

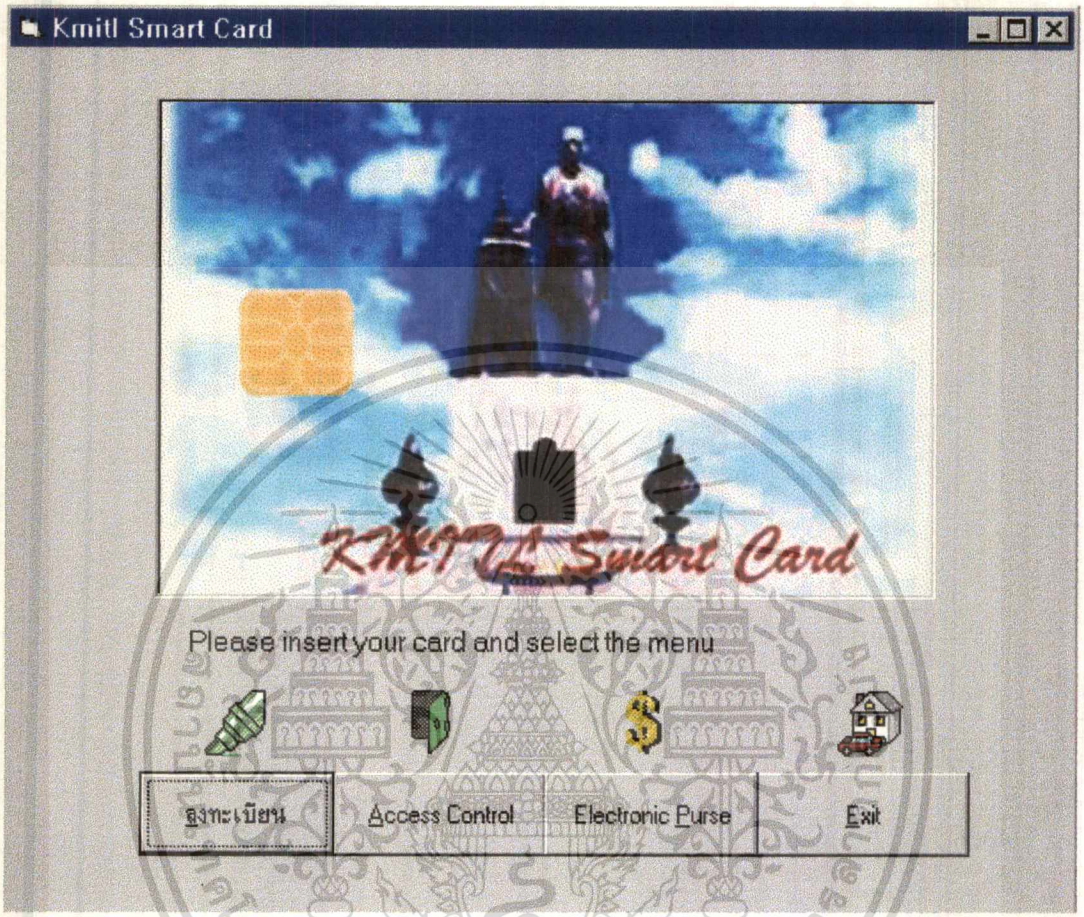
-ใส่กุญแจรหัส เป็นอันเสร็จสิ้นขั้นตอนในการ Credit purse

4.3.2 โปรแกรมการใช้งานสำหรับผู้ใช้บัตร

โปรแกรมนี้เป็นการใช้งานในลักษณะการลงทะเบียนวิชาเรียนของนักศึกษา โดยจะมีวิชาเรียนให้เลือกทางด้านซ้ายดังรูป 4.16 เมื่อเลือกวิชาเรียนเสร็จแล้ว เลือกตัวเลือก PURSE เพื่อจ่ายเงินค่าลงทะเบียน จะปรากฏหน้าจอให้ใส่กุญแจรหัสเช่นเดียวกับรูป 4.11 ใส่กุญแจรหัส วิชาเรียนจะถูกเขียนลงในบัตรเป็นอันเสร็จสิ้นขั้นตอนการลงทะเบียนวิชาเรียน

รูปที่ 4.16 หน้าจอในการลงทะเบียน

4.3.3 โปรแกรมการใช้งานบน Windows



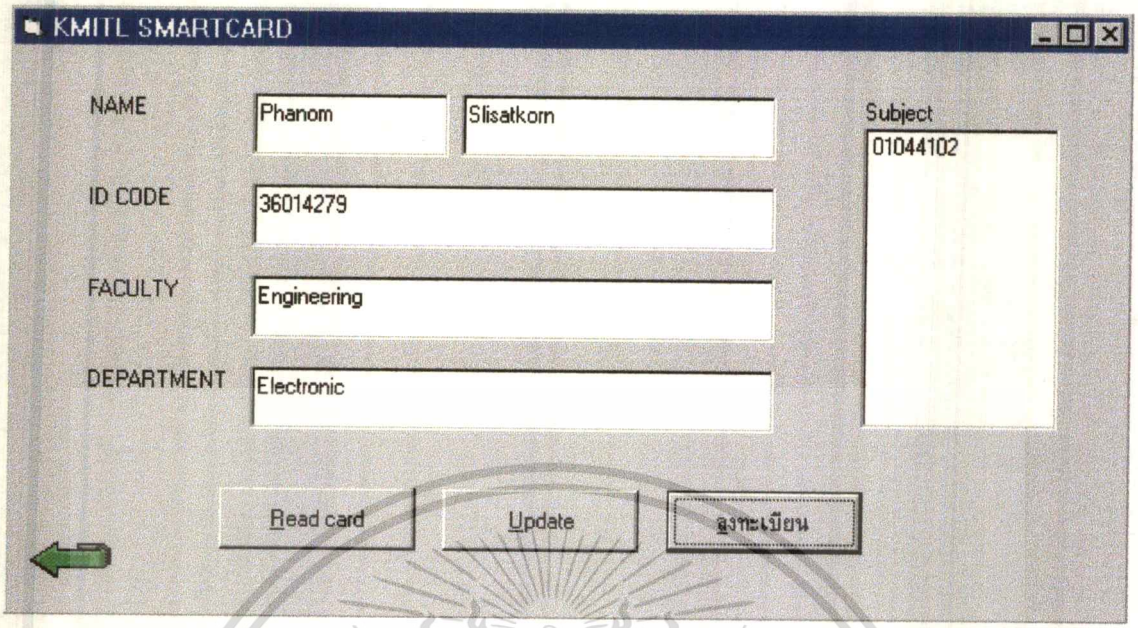
รูปที่ 4.17 User Interface ของโปรแกรมการใช้งานบน Windows

แบ่งได้เป็น 3 ส่วนดังนี้

- โปรแกรมในการลงทะเบียน
- โปรแกรมที่ใช้ในการควบคุมการเข้าออกบริเวณ (Access Control)
- โปรแกรมในการจับจ่ายเงิน (Electronic Purse)

<u>โปรแกรมในการลงทะเบียน</u> โปรแกรมในส่วนนี้จะเป็นการลงทะเบียนวิชาเรียน จ่ายค่าลงทะเบียน โดยจะมีการเขียนรหัสของวิชาเรียนลงในบัตร พร้อมหักจ่ายเงินจากบัตร มีขั้นตอนดังนี้

เลือกปุ่มลงทะเบียนจาก main menu รูปที่ 4.17 จะปรากฏหน้าจอดังรูปที่ 4.18



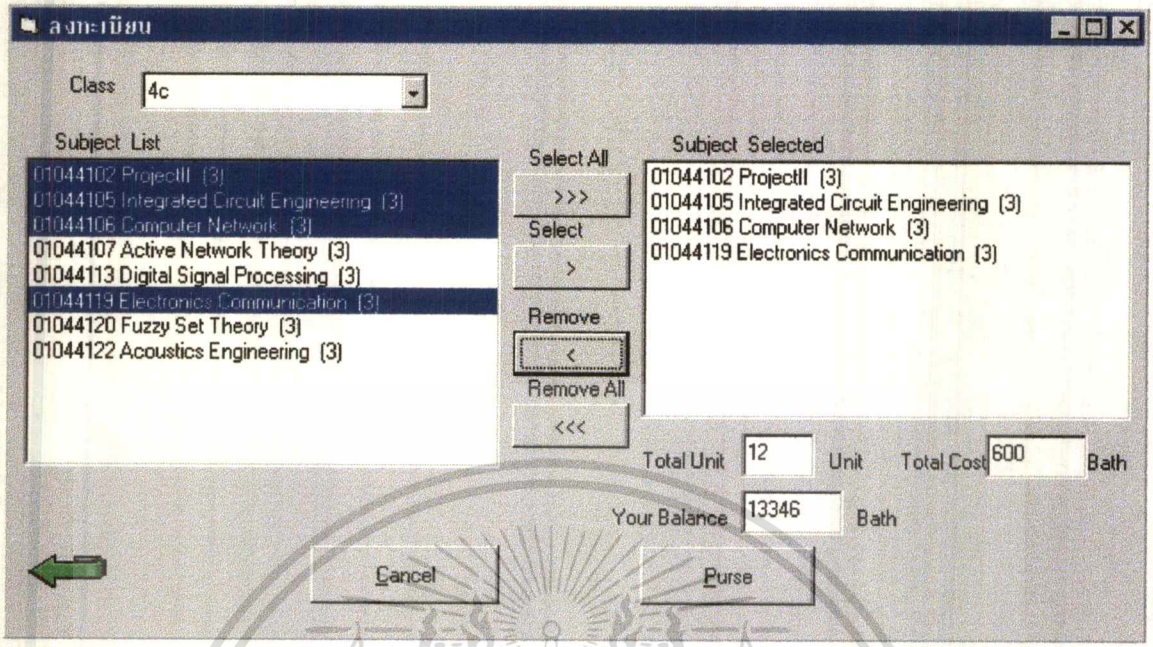
รูปที่ 4.18 หน้าจอในการอ่านข้อมูลเกี่ยวกับนักศึกษา

- **Read Card** เป็นการสั่งให้เครื่องอ่านข้อมูลภายในบัตรมาแสดงบนหน้าจอดังรูป 4.18
- **Update** เป็นคำสั่งที่ใช้เปลี่ยนแปลงข้อมูลในบัตร เมื่อเราแก้ไขข้อมูลในหน้าจอรูปที่ 4.18 คลิกปุ่ม Update จะปรากฏหน้าจอที่ 4.19 ให้เราใส่กุญแจรหัส เมื่อเราใส่กุญแจรหัสเสร็จเป็นอันเสร็จสิ้นการแก้ไขข้อมูล

รูปที่ 4.19 หน้าต่างในการใส่กุญแจรหัส

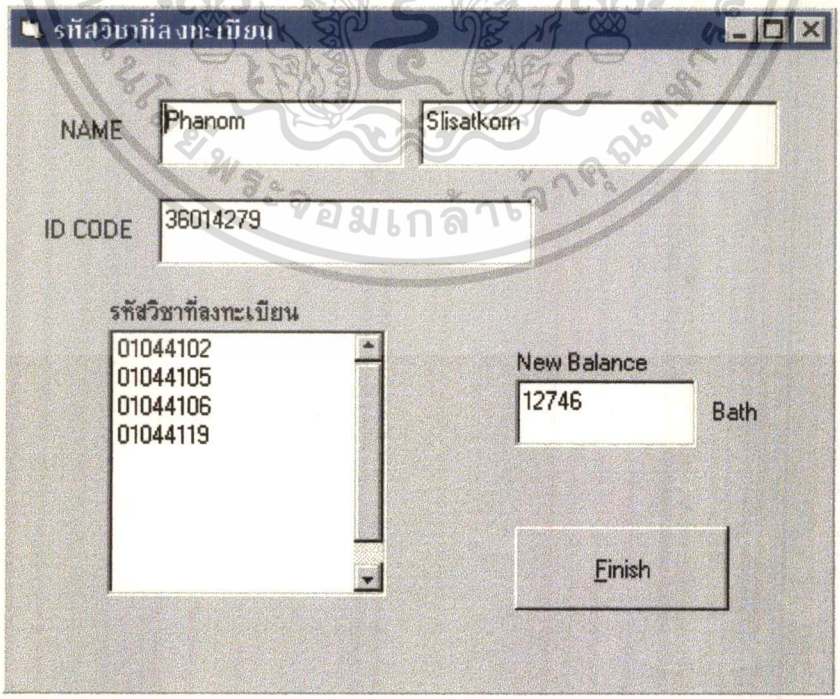
- **ลงทะเบียน** มีขั้นตอนดังนี้

  - เลือกปุ่มลงทะเบียน จะปรากฏหน้าจอดังรูป 4.20
  - เลือก class ของชั้นเรียน จะปรากฏวิชาของชั้นเรียนนั้นบน Subject List
  - เลือกวิชาเรียนโดยการเลือกใน Subject List และใช้คำสั่ง Select หรือใช้คำสั่ง Select All ในการเลือกวิชาเรียนทั้งหมด วิชาเรียนที่ถูกเลือกจะปรากฏใน Subject Selected พร้อมทั้งแสดงจำนวนหน่วยกิจและจำนวนเงินที่ต้องจ่าย



รูปที่ 4.20 หน้าจอในการลงทะเบียน

- คลิกปุ่ม Purse เพื่อจ่ายเงินจะปรากฏหน้าจอให้ใส่กุญแจรหัสดังรูปที่ 4.19
- เมื่อใส่รหัสเสร็จเรียบร้อยเครื่องจะหักเงินพร้อมทั้งบันทึกรหัสวิชาที่ลงทะเบียนลงในบัตร และแสดงหน้าจอดังรูปที่ 4.21 เป็นอันเสร็จสิ้นการลงทะเบียน

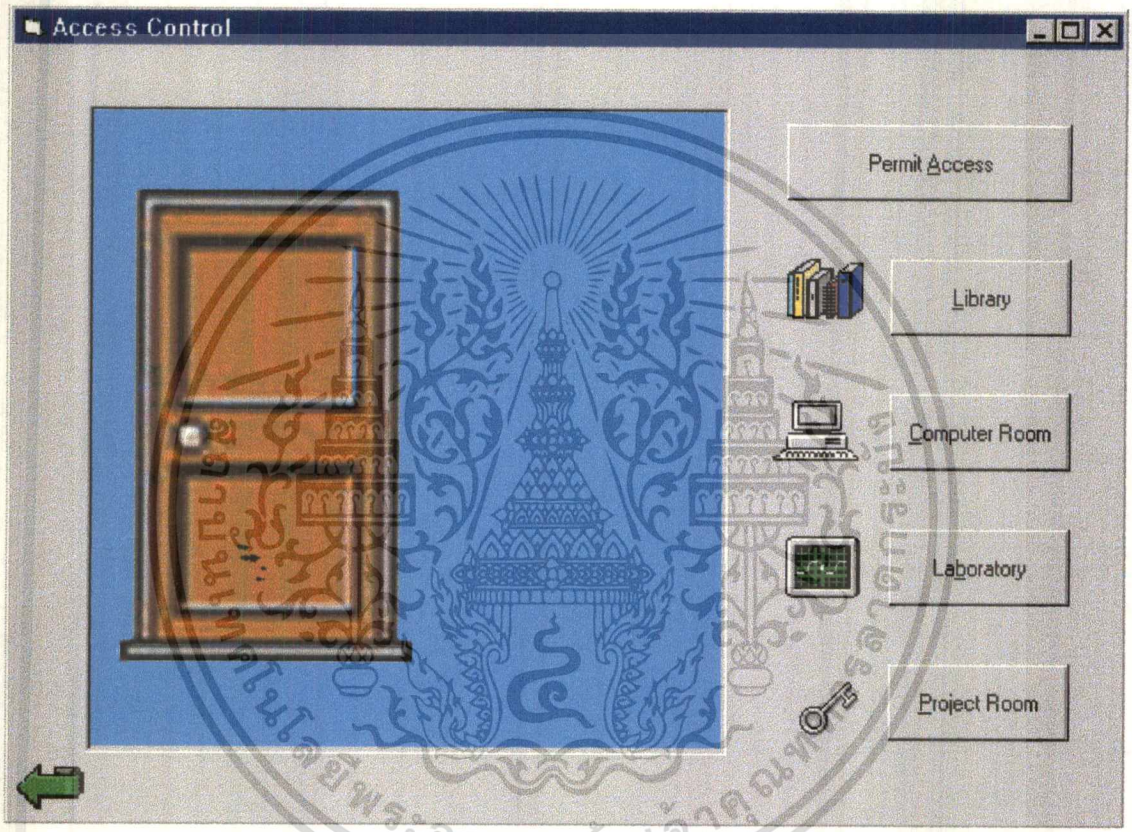


รูปที่ 4.21 หน้าจอแสดงรหัสวิชาที่ลงทะเบียน

<u>โปรแกรมที่ใช้ในการควบคุมการเข้าออกบริเวณ (Access Control)</u>

โปรแกรมในส่วนนี้จะเป็นการควบคุมการเข้าออกบริเวณ ในลักษณะ off line โดยจะมีการกำหนดการอนุญาตในการเข้าออกบริเวณใดบริเวณหนึ่งลงบนบัตร และตรวจสอบการอนุญาตจากบัตร

เลือกปุ่ม Access Control จาก main menu จะปรากฏหน้าจอดังรูปที่ 4.22

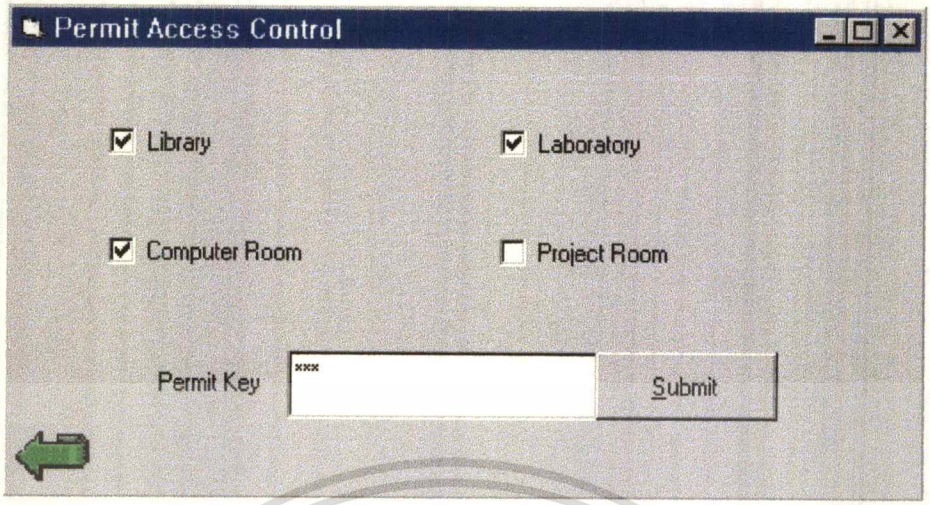


รูปที่ 4.22 หน้าจอของโปรแกรมที่ใช้ในการควบคุมการเข้าออกบริเวณ

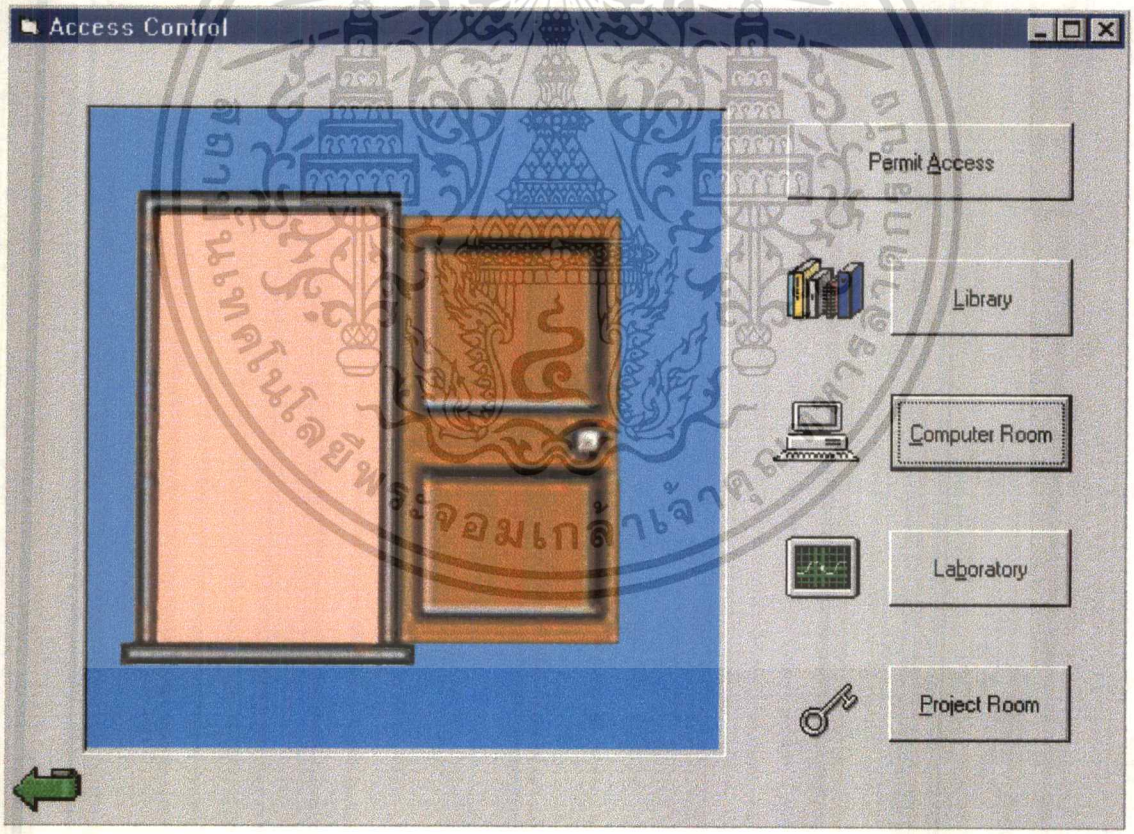
การกำหนดการอนุญาต เลือกปุ่ม Permit Access จะปรากฏหน้าจอดังรูปที่ 4.23 เลือกสถานที่ที่จะอนุญาตให้ผ่านเข้าออก ทำการใส่กุญแจรหัส แล้ว submit key

หลังจากการกำหนดการอนุญาตแล้วสถานที่ใดที่มีการกำหนดการอนุญาตจะสามารถใช้บัตรในการเปิดประตูได้ดังรูปที่ 4.24

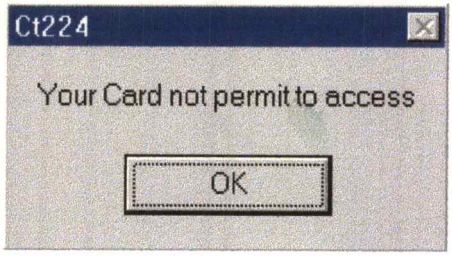
ส่วนสถานที่ที่มิได้กำหนดการอนุญาตจะปรากฏหน้าจอดังรูปที่ 4.25



รูปที่ 4.23 หน้าจอในการกำหนดการอนุญาตการเข้าออกบริเวณ



รูปที่ 4.24 หน้าจอแสดงการอนุญาตในการผ่านเข้าออกสถานที่

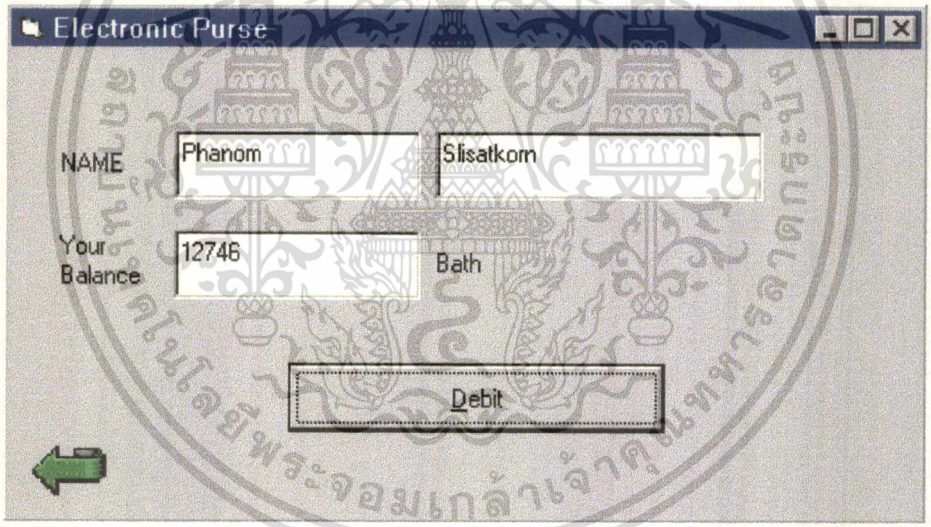


รูปที่ 4.25 หน้าจอแสดงการไม่อนุญาตในการผ่านเข้าออกสถานที่

โปรแกรมในการจับจ่ายเงิน (Electronic Purse)

เป็นการหักเงินจากบัตรในการจับจ่ายสินค้าต่างๆ

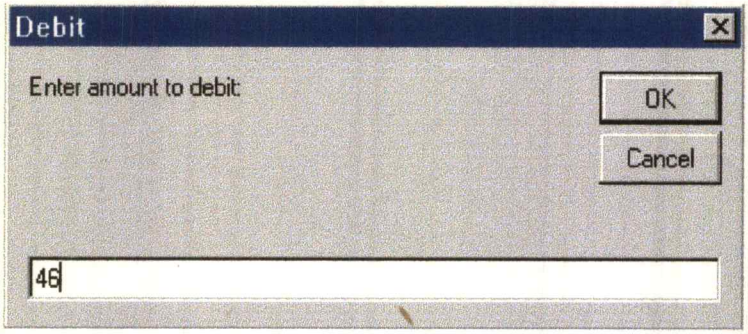
เลือกปุ่ม Electronic Purse จาก main menu จะปรากฏหน้าจอดังรูปที่ 4.26 แสดงข้อมูลของผู้ถือบัตรและยอดเงินคงเหลือ



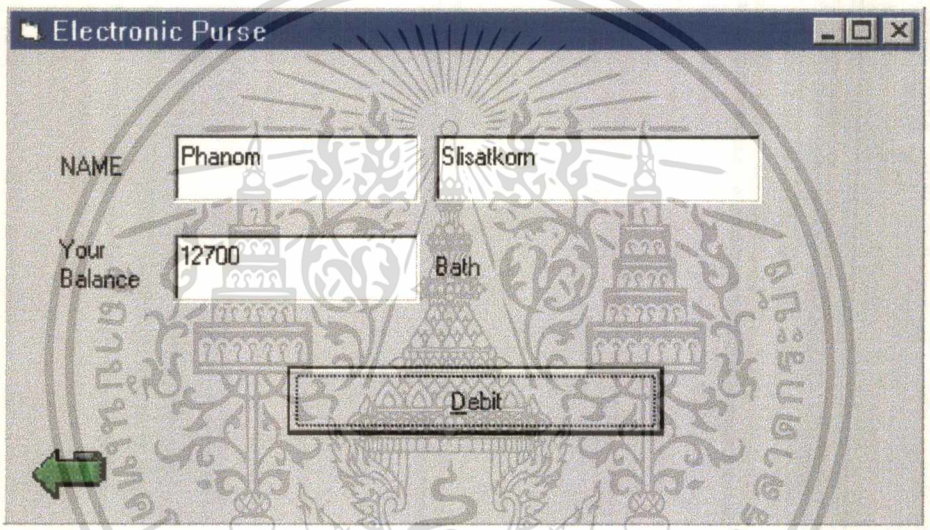
รูปที่ 4.26 หน้าจอของโปรแกรมในการจับจ่ายเงิน

เลือกปุ่ม Debit จะปรากฏหน้าจอให้ใส่จำนวนเงินที่จะจ่ายดังรูปที่ 4.27

ใส่จำนวนเงินแล้วกดปุ่ม OK จะปรากฏหน้าจอให้ใส่กุญแจรหัสดังรูปที่ 4.19 เป็นอันเสร็จสิ้นขั้นตอนในการจ่ายเงิน จะปรากฏหน้าจอดังรูป 4.28 แสดงยอดเงินคงเหลือในบัตร



รูปที่ 4.27 หน้าจอในการรับค่าจำนวนเงินที่ต้องการจ่าย



รูปที่ 4.28 หน้าจอแสดงยอดเงินคงเหลือในบัตร

บทที่ 5
สรุปและวิจารณ์

สมาร์ทการ์ดเป็นเทคโนโลยีที่เอื้อประโยชน์อย่างมากแก่มนุษย์ เนื่องจากช่วยประหยัดเวลาและทรัพยากรมนุษย์ในการปฏิบัติงาน มีความปลอดภัยสูง ใช้งานได้หลายประเภทในใบบัตรเดียว และยังสามารถใช้งานกับระบบเครือข่ายต่าง ๆ เสมือนมีกระเป๋าเงินอิเล็กทรอนิกส์ที่เก็บข้อมูลต่างๆ และมีมูลค่าของเงินในกระเป๋าที่พร้อมจะจับจ่ายในได้ทันที ผ่านทางสื่อ On line และ Off line โดยไม่จำเป็นต้องทำการเคลื่อนย้ายตัวเงินโดยตรง ซึ่งช่วยเพิ่มความปลอดภัยให้แก่ผู้บริโภค และอำนวยความสะดวกในการจับจ่ายในรูปเงินดิจิตอล

โครงงานนี้จึงเป็นการริเริ่มนำเอาสมาร์ทการ์ดมาใช้งาน เพื่อกระตุ้นให้เกิดความสนใจ , เกิดการนำสมาร์ทการ์ดไปใช้งานและการพัฒนาเทคโนโลยีนี้ในประเทศ โดยโครงงานนี้เป็นการนำสมาร์ทการ์ดมาใช้ในลักษณะบัตรประจำตัวนักศึกษา ใช้ในการควบคุมการผ่านเข้าออกสถานที่ (Access control) ,ใช้ในการจับจ่ายสินค้า(Electronic purse) และใช้ในการลงทะเบียนวิชาเรียน ซึ่งแต่เดิมการลงทะเบียนต้องใช้เจ้าหน้าที่ในการดำเนินการจำนวนมาก ต้องติดต่อกับหลายแผนก และใช้เวลาในการดำเนินการมาก การใช้สมาร์ทการ์ดในลักษณะเครื่องลงทะเบียนอัตโนมัติ ซึ่งสามารถดำเนินงานหลาย ๆ อย่าง เช่น การลงวิชาเรียน การจ่ายค่าลงทะเบียน การอนุญาตในการใช้บริการต่าง ๆ เช่นห้องสมุด ห้องคอมพิวเตอร์ ในเวลาอันสั้น และยังสามารถพัฒนาไปสู่การลงทะเบียนแบบ on line ผ่านทางเครือข่ายต่าง ๆได้อีกด้วย

ข้อจำกัดในการทำงาน

เนื่องจากเป็นเทคโนโลยีใหม่จึงมีอุปสรรคในการหาข้อมูล หนังสืออ้างอิง และอุปกรณ์ต่าง ๆ จึงทำให้ขีดความสามารถในการพัฒนามีข้อจำกัด

แนวทางในการพัฒนา

พัฒนาระบบในการใช้งานผ่านเครือข่ายและDatabase Server เช่น SQL server โดยจำเป็นต้องคำนึงถึงความปลอดภัยของข้อมูลด้วย อาจใช้โปรแกรมในการเข้ารหัส(Encription data), การตรวจสอบการส่งข้อมูลระหว่างผู้ส่งกับผู้รับ(certified data) หรือการใช้ชิปในการเข้ารหัส

หนังสืออ้างอิง

1. Steve Oualline, "Advanced C Programming " Brady Publishing,1992
2. "Real Time Graphics and User Interface Tools",Quinn-curtis,Inc
3. "UCRS-1 V1.5 Manual Reference",Solaic Smart Card Industries,1995
4. "PocketBook V3.1 Smart Card User's Manual" Solaic Smart Card Industries,1994
5. "Manufacturing Chip Cards with Solaic" Solaic Smart Card Industries,1994
6. "TLP 224 NV User's Manual" , Bull CP8
7. "CP8 Products Smart Cards" , Bull CP8
8. " The TB100 IC-Card Operating System User's Guide", Bull CP8
9. "CP-TPC Tool Box Training Session Documentation", Dassault AT,1994

หมายเหตุ ต้องการข้อมูลเพิ่มเติมติดต่อ tel. 4330219, 4355582

ภาคผนวก

project file consist of

- form main
- form1
- form2
- form3
- form4
- form5
- form6
- form7
- form8
- com.bas
- function.bas
- tlp224.bas

form main

```
Private Sub Command10_Click()
 Dim  q As Integer
 Dim aa,  dm, x,  result As String

    q =  openport(2, "9600,N,8,1")

      q =  PowerOn( result)
      q =  InsertCard( result, "Insert card into reader.")
     'ShowText  result
     PowerOnFlag = True
   Text1.Text = "faa40000029901"
    dm =  MakeCmdStr( COMMAND_INCMD, Text1.Text)
    q =  SendCommand(1, UCase( dm),  result)
   'ShowText  result
    Text1.Text = "fab000400f"
```

```
 dm =  MakeCmdStr( COMMAND_OUTCMD, Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
'text4.Text =  result
'ShowText  result
Text5.Text = AsciiToChar( result)
'Text1.Text = "100f"
'q = Read(Text1.Text)
Text1.Text = "fab000300f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, Text1.Text)
q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
text4.Text = AsciiToChar( result)
Text1.Text = "fab000200f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Text3.Text = AsciiToChar( result)
Text1.Text = "fab000100f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
```

```
Text2.Text = AsciiToChar( result)
Text1.Text = "fab000000f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Text1.Text = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
 q =  SendCommand(1, UCase( dm),  result)

 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0000008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(0) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0000808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(1) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0001008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(2) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0001808")
```

```
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(3) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0002008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(4) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0002808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(5) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(6) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
List1.List(7) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004008")
```

```
    q =  SendCommand(1, UCase( dm),  result)
    x = Len( result) - 6
    Mid( result, 1) = "00000"
    Mid( result, x) = "000000"
    List1.List(8) = AsciiToChar( result)
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004808")
    q =  SendCommand(1, UCase( dm),  result)
    x = Len( result) - 6
    Mid( result, 1) = "00000"
    Mid( result, x) = "000000"
    List1.List(9) = AsciiToChar( result)
    CloseCom
End Sub



Private Sub Command1_Click()
  Dim  q, i, j As Integer,  result As String
  Dim PIN As String * 16
  Dim aa,  dm, x, a, a1, a2, a3, a4, a5, b1 As String

     q =  openport(2, "9600,N,8,1")
     q =  PowerOn( result)
     q =  InsertCard( result, "Insert card into reader.")
    'ShowText  result
    PowerOnFlag = True

    dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029901")
    q =  SendCommand(1, UCase( dm),  result)
```

```
'Form1.Show
PIN = InputBox("Enter your PIN:", "PIN")
dm =  MakeCmdStr( COMMAND_INCMD, "fa20000008" & PIN)
q =  SendCommand(1, UCase( dm),  result)
result = Right( result, 7)
result = Left( result, 4)
result =  result Xor 9000
q = Update(Text1.Text, "000010")
q = Update(Text2.Text, "001010")
q = Update(Text3.Text, "002010")
q = Update(text4.Text, "003010")
q = Update(Text5.Text, "004010")

CloseCom
End Sub



Private Sub Command2_Click()
Dim  q As Integer
Dim t1, t2, t3, t4 As Long
Dim r5 As String
Dim aa,  dm, x, r1, r2, r3, r4,  result As String
q =  openport(2, "9600,N,8,1")


    q =  PowerOn( result)
    q =  InsertCard( result, "Insert card into reader.")
   'ShowText  result
    PowerOnFlag = True
'dm =  MakeCmdStr( COMMAND_INCMD, "fa200000089999999999999999")
```

```
    q =  SendCommand(1, UCase( dm),  result)

    dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
    q =  SendCommand(1, UCase( dm),  result)
   'ShowText  result

    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab200040f")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
   r1 = Left( result, 22)
   r5 = Right(r1, 6)
   Form2.Text2.Text = (HextoDec(r5)) / 100
   main.Hide
   Form2.Show
End Sub
```

form1

```
Private Sub Command1_Click()

   Form1.Hide
   form3.Show
End Sub
```

form2

```
Private Sub Form_Activate()
  Dim  q As Integer
```

```
Dim t1, t2, t3, t4 As Long
Dim r5 As String
Dim aa,  dm, x, r1, r2, r3, r4,  result As String
  List1.Clear
  Combo1.AddItem "2c"
  Combo1.AddItem "3c"
  Combo1.AddItem "4c"
  Combo1.AddItem "2d"
  List1.AddItem "01044102 ProjectII  (3)"
  List1.AddItem "01044105 Integrated Circuit Engineering  (3)"
  List1.AddItem "01044106 Computer Network  (3)"
  List1.AddItem "01044107 Active Network Theory  (3)"
  List1.AddItem "01044113 Digital Signal Processing  (3)"
  List1.AddItem "01044119 Electronics Communication  (3)"
  List1.AddItem "01044120 Fuzzy Set Theory  (3)"
  List1.AddItem "01044122 Acoustics Engineering  (3)"
  List2.Clear
  Text3.Text = ""
  Text1.Text = ""

End Sub


Private Sub Combo1_Click()
  Select Case Combo1.Text
  Case "4c"
    List1.Clear
    List1.AddItem "01044102 ProjectII  (3)"
    List1.AddItem "01044105 Integrated Circuit Engineering  (3)"
```

```
    List1.AddItem "01044106 Computer Network  (3)"
    List1.AddItem "01044107 Active Network Theory  (3)"
    List1.AddItem "01044113 Digital Signal Processing  (3)"
    List1.AddItem "01044119 Electronics Communication  (3)"
    List1.AddItem "01044120 Fuzzy Set Theory  (3)"
    List1.AddItem "01044122 Acoustics Engineering  (3)"
  Case "3c"
    List1.Clear
    List1.AddItem "01043112 Electrnics Circuit ApplicationII  (3)"
    List1.AddItem "01043107 Advanced Electronics Engineering  (3)"
    List1.AddItem "01043108 Communication Network and Transmission Line  (3)"
    List1.AddItem "01043109 Power Electronics  (3)"
    List1.AddItem "01043110 Signal Analysis  (3)"
    List1.AddItem "01043111 Electronics Laboratory III  (2)"
  Case "2c"
  Case "2d"
  End Select
End Sub


Private Sub Command3_Click()
Dim b(9) As String * 8
 Dim u2, unit As Double

  Dim i, j, k

  For i = 0 To List1.ListCount - 1
    k = 0
```

```
        For j = 0 To List2.ListCount
           If List2.List(j) = List1.List(i) Then
              k = 1
           End If
        Next j
        If k <> 1 Then
           List2.AddItem List1.List(i)
        End If

    Next i
    For i = 0 To 9
       b(i) = ""
    Next i
    unit = 0
    For i = 0 To List2.ListCount - 1
       b(i) = CutCode(List2.List(i))
       u1 = CutUnit(List2.List(i))
       u2 = CDbl(u1)
       unit = unit + u2
    Next i
    Text1.Text = unit
    Text3.Text = unit * 50

End Sub


Private Sub Command1_Click()
   Dim b(9) As String * 8
   Dim u2, unit As Double
```

```
    Dim i, j, k
    Dim d As String
    For i = 0 To List1.ListCount - 1
      k = 0
      If List1.Selected(i) Then
        For j = 0 To List2.ListCount
          If List2.List(j) = List1.List(i) Then
            k = 1
          End If
        Next j
        If k <> 1 Then
          List2.AddItem List1.List(i)
        End If
      End If
    Next i
   For i = 0 To 9
      b(i) = ""
    Next i
    unit = 0
    For i = 0 To List2.ListCount - 1
      b(i) = CutCode(List2.List(i))
      u1 = CutUnit(List2.List(i))
      u2 = CDbl(u1)
      unit = unit + u2
    Next i
    Text1.Text = unit
    Text3.Text = unit * 50
  End Sub
```

```
Private Sub Command2_Click()
Dim b(9) As String * 8
 Dim u2, unit As Double
Dim i, j

   For i = 0 To List2.ListCount - 1
     If List2.Selected(i) Then
       j = i

     End If
   Next i
   If List2.ListCount > 0 Then
     List2.RemoveItem (j)
   End If
For i = 0 To 9
     b(i) = ""
   Next i
   unit = 0
   For i = 0 To List2.ListCount - 1
     b(i) = CutCode(List2.List(i))
     u1 = CutUnit(List2.List(i))
     u2 = CDbl(u1)
     unit = unit + u2
   Next i
   Text1.Text = unit
   Text3.Text = unit * 50
End Sub
```

```
Private Sub Command4_Click()
 List2.Clear
 Text1.Text = "0"
 Text3.Text = "0"
End Sub


Private Sub Command6_Click()
  Dim  q, i, j, l As Integer,  result As String
   Dim r5 As String
 Dim r1, r2, r3, r4 As String
  Dim PIN As String * 16
 Dim aa,  dm, x, a, a1, a2, a3, a4, a5, b1 As String
 Dim cost As String


    Dim u1 As String
    Dim u2, unit As Double
    For i = 0 To 9
       z(i) = ""
    Next i
    unit = 0
    For i = 0 To List2.ListCount - 1
       z(i) = CutCode(List2.List(i))
       u1 = CutUnit(List2.List(i))
       u2 = CDbl(u1)
```

```
    unit = unit + u2
  Next i
  Text1.Text = unit
  cost1 = unit * 50 * 100
  cost = hex(cost1)

  For l = Len(cost) To 5
    cost = "0" & cost
  Next l
  If cost <> "000000" Then
     q =  openport(2, "9600,N,8,1")
     q =  PowerOn( result)
     q =  InsertCard( result, "Insert card into reader.")
    'ShowText  result
    PowerOnFlag = True

  dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
  q =  SendCommand(1, UCase( dm),  result)
  Form4.Text1.Text = cost
  Form4.Show
  End If
End Sub
```

form3

```
Private Sub Command1_Click()
 Dim  q As Integer
Dim aa,  dm, x,  result As String
   main.Text1.Text = ""
   main.Text2.Text = ""
  main.Text3.Text = ""
  main.text4.Text = ""
  main.Text5.Text = ""
 main.List1.Clear
  PowerOnFlag = False
     q =  openport(2, "9600,N,8,1")
     q =  PowerOn( result)
    x = Left( result, 6)
    'If x = "601000" Then

     q =  InsertCard( result, "Insert card into reader.")
     x = Left( result, 6)
    If x = "601000" Then
    main.Text1.Text = "faa40000029901"
   dm =  MakeCmdStr( COMMAND_INCMD, main.Text1.Text)
   q =  SendCommand(1, UCase( dm),  result)
  'ShowText  result
  main.Text1.Text = "fab000400f"
   dm =  MakeCmdStr( COMMAND_OUTCMD, main.Text1.Text)

   q =  SendCommand(1, UCase( dm),  result)
  x = Len( result) - 6
```

```
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
'text4.Text =  result
'ShowText  result
main.Text5.Text = AsciiToChar( result)
'Text1.Text = "100f"
' q = Read(Text1.Text)
main.Text1.Text = "fab000300f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, main.Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.text4.Text = AsciiToChar( result)
main.Text1.Text = "fab000200f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, main.Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.Text3.Text = AsciiToChar( result)
main.Text1.Text = "fab000100f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, main.Text1.Text)
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.Text2.Text = AsciiToChar( result)
main.Text1.Text = "fab000000f"
 dm =  MakeCmdStr( COMMAND_OUTCMD, main.Text1.Text)
```

```
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.Text1.Text = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_INCMD, "faa40000029902")
q = SendCommand(1, UCase( dm), result)

dm = MakeCmdStr( COMMAND_OUTCMD, "fab0000008")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(0) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0000808")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(1) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0001008")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(2) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0001808")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
```

```
Mid( result, x) = "000000"
main.List1.List(3) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0002008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(4) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0002808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(5) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(6) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
main.List1.List(7) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
```

```
    Mid( result, x) = "000000"
    main.List1.List(8) = AsciiToChar( result)
     dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004808")
     q =  SendCommand(1, UCase( dm),  result)
    x = Len( result) - 6
    Mid( result, 1) = "00000"
    Mid( result, x) = "000000"
    main.List1.List(9) = AsciiToChar( result)
    CloseCom

      Else
      MsgBox CardAbsent + "Please insert Smart Card and Click Read Card"
      End If
      form3.Hide
      main.Show
End Sub



Private Sub Command2_Click()
    form3.Hide
    Form5.Show
End Sub



Private Sub Command4_Click()
 Dim q As Integer
Dim  dm,  result As String
Dim r1, x, r5 As String
```

```
q =  openport(2, "9600,N,8,1")
      q =  PowerOn( result)
      q =  InsertCard( result, "Insert card into reader.")
     'ShowText  result
     PowerOnFlag = True
   dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029901")
    q =  SendCommand(1, UCase( dm),  result)

    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab000000f")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
   Form6.Text1.Text = AsciiToChar( result)
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab000100f")
   ,q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
   Form6.Text2.Text = AsciiToChar( result)

    dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
    q =  SendCommand(1, UCase( dm),  result)
   'ShowText  result

    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab200040f")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
```

```
    r1 = Left( result, 22)
    r5 = Right(r1, 6)
    Form6.Text3.Text = (HextoDec(r5)) / 100
    form3.Hide
    Form6.Show

End Sub

Private Sub Command3_Click()
    End
End Sub
```

from4

```
Private Sub Form_Activate()
cost = Text1.Text
Text1.Text = ""

End Sub


Private Sub Command1_Click()
Dim  q, i, j, l As Integer,  result As String
   Dim r5 As String
 Dim r1, r2, r3, r4 As String
  Dim PIN As String * 16
 Dim aa,  dm, x, a, a1, a2, a3, a4, a5, b1 As String
```

```
Dim b(9) As String * 8

Dim u1 As String
Dim u2, unit As Double



PIN = Text1.Text
 dm =  MakeCmdStr( COMMAND_INCMD, "fa20000008" & PIN)
 q =  SendCommand(1, UCase( dm),  result)
 result = Right( result, 7)
 result = Left( result, 4)
 result =  result Xor 9000
If  result = 0 Then
Y = datee
 dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
 q =  SendCommand(1, UCase( dm),  result)
 dm =  MakeCmdStr( COMMAND_INCMD, "fa300000008000000" & Y & cost)
 q =  SendCommand(1, UCase( dm),  result)
 result = Right( result, 7)
 result = Left( result, 4)
 result =  result Xor 9000
   If  result = 0 Then
 dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
 q =  SendCommand(1, UCase( dm),  result)


 q = Update(z(0), "000008")
 q = Update(z(1), "000808")
 q = Update(z(2), "001008")
 q = Update(z(3), "001808")
```

```
q = Update(z(4), "002008")
q = Update(z(5), "002808")
q = Update(z(6), "003008")
q = Update(z(7), "003808")
q = Update(z(8), "004008")
q = Update(z(9), "004808")


dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029901")
q =  SendCommand(1, UCase( dm),  result)

dm =  MakeCmdStr( COMMAND_OUTCMD, "fab000000f")
q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.Text1.Text = AsciiToChar( result)
dm =  MakeCmdStr( COMMAND_OUTCMD, "fab000100f")
q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.Text2.Text = AsciiToChar( result)
dm =  MakeCmdStr( COMMAND_OUTCMD, "fab000200f")
q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.text4.Text = AsciiToChar( result)
dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
```

```
q = SendCommand(1, UCase( dm), result)

dm = MakeCmdStr( COMMAND_OUTCMD, "fab0000008")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(0) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0000808")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(1) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0001008")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(2) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0001808")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(3) = AsciiToChar( result)
dm = MakeCmdStr( COMMAND_OUTCMD, "fab0002008")
q = SendCommand(1, UCase( dm), result)
x = Len( result) - 6
Mid( result, 1) = "00000"
```

```
Mid( result, x) = "000000"
Form1.List1.List(4) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0002808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(5) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(6) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0003808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(7) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004008")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
Form1.List1.List(8) = AsciiToChar( result)
 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0004808")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
```

```
Mid( result, x) = "000000"
Form1.List1.List(9) = AsciiToChar( result)
 q =  openport(2, "9600,N,8,1")



 dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
 q =  SendCommand(1, UCase( dm),  result)
'ShowText  result

 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab200040f")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
 r1 = Left( result, 22)
r5 = Right(r1, 6)
Form1.Text3.Text = (HextoDec(r5)) / 100

 CloseCom
Form1.Show
Form2.Hide
 Open "e:\proj97.doc" For Append As #1
g = Date & " " & Time
Print #1, g
g = Form1.Text1.Text & " " & Form1.Text2.Text & " " & Form1.text4.Text
Print #1, g
g = "Total cost " & Form2.Text3.Text & "Bath"
Print #1, g
g = "Subject list"
```

```
Print #1, g
g = Form1.List1.List(0)
Print #1, g
g = Form1.List1.List(1)
Print #1, g
g = Form1.List1.List(2)
Print #1, g
g = Form1.List1.List(3)
Print #1, g
g = Form1.List1.List(4)
Print #1, g
g = Form1.List1.List(5)
Print #1, g
g = Form1.List1.List(6)
Print #1, g
g = Form1.List1.List(7)
Print #1, g
g = "_____"
Print #1, g
Close
   End If

End If
Form4.Hide

End Sub
```

```
Private Sub Command2_Click()
Form4.Hide

End Sub
```

form5

```
Private Sub Form_Activate()
   Picture1.Visible = True
   Picture2.Visible = False
End Sub


Private Sub Command5_Click()
Form8.Show
Form5.Hide
End Sub


Private Sub Command6_Click()
 Dim  q As Integer
 Dim aa,  dm, x,  result As String
   Picture1.Visible = True
   Picture2.Visible = False
   q =  openport(2, "9600,N,8,1")
   q =  PowerOn( result)
  q =  InsertCard( result, "Insert card into reader.")
   dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
   q =  SendCommand(1, UCase( dm),  result)
```

```
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0005008")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
    result = AsciiToChar( result)
   aa = Left( result, 1)
   If aa = "1" Then
   Picture2.Visible = True
   Picture1.Visible = False
   Else
   MsgBox "Your Card not permit to access"
   Picture1.Visible = True
   Picture2.Visible = False
   End If
End Sub


Private Sub Command7_Click()
Dim  q As Integer
 Dim aa,  dm, x,  result As String
   Picture1.Visible = True
   Picture2.Visible = False
    q =  openport(2, "9600,N,8,1")
    q =  PowerOn( result)
    q =  InsertCard( result, "Insert card into reader.")
    dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
    q =  SendCommand(1, UCase( dm),  result)
```

```
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0005008")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
    result = AsciiToChar( result)
   aa = Left( result, 2)
   aa = Right(aa, 1)
   If aa = "1" Then
   Picture2.Visible = True
   Picture1.Visible = False
   Else
   MsgBox "Your Card not permit to access"
   Picture1.Visible = True
   Picture2.Visible = False
   End If
End Sub


Private Sub Command8_Click()
Dim  q As Integer
 Dim aa,  dm, x,  result As String
   Picture1.Visible = True
   Picture2.Visible = False
    q =  openport(2, "9600,N,8,1")
    q =  PowerOn( result)
    q =  InsertCard( result, "Insert card into reader.")
    dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
    q =  SendCommand(1, UCase( dm),  result)
```

```
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0005008")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
    result = AsciiToChar( result)
   aa = Left( result, 3)
   aa = Right(aa, 1)
   If aa = "1" Then
   Picture2.Visible = True
   Picture1.Visible = False
   Else
   MsgBox "Your Card not permit to access"
   Picture1.Visible = True
   Picture2.Visible = False
   End If
End Sub


Private Sub Command9_Click()
Dim  q As Integer
 Dim aa,  dm, x,  result As String
   Picture1.Visible = True
   Picture2.Visible = False
    q =  openport(2, "9600,N,8,1")
    q =  PowerOn( result)
    q =  InsertCard( result, "Insert card into reader.")
    dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
    q =  SendCommand(1, UCase( dm),  result)
```

```
    dm =  MakeCmdStr( COMMAND_OUTCMD, "fab0005008")
    q =  SendCommand(1, UCase( dm),  result)
   x = Len( result) - 6
   Mid( result, 1) = "00000"
   Mid( result, x) = "000000"
    result = AsciiToChar( result)
   aa = Left( result, 4)
   aa = Right(aa, 1)
   If aa = "1" Then
   Picture2.Visible = True
   Picture1.Visible = False
   Else
   MsgBox "Your Card not permit to access"
   Picture1.Visible = True
   Picture2.Visible = False
   End If
End Sub


Private Sub Image2_Click()
   Form5.Hide
   form3.Show
End Sub
```

form6

```
Private Sub Command1_Click()
```

```
  Form7.Show
End Sub


Private Sub Image2_Click()
   Form6.Hide
   form3.Show
End Sub
```

form7

```
Private Sub Form_Activate()
 Dim  result As String
 Dim  q As Integer

 cost = InputBox("Enter amount to debit:", "Debit")


 If cost <> "" Then

 Text1.Text = ""
 Else: Form7.Hide
 End If
 q =  openport(2, "9600,N,8,1")
      q =  PowerOn( result)
End Sub
```

```
Private Sub Command1_Click()
Dim  q, i, j, l As Integer,  result As String
  Dim r5 As String
 Dim r1, r2, r3, r4 As String
  Dim PIN As String * 16
 Dim aa,  dm, x, a, a1, a2, a3, a4, a5, b1 As String


  Dim b(9) As String * 8


  Dim u1 As String
  Dim u2, unit As Double
  i = cost
  cost = cost * 100
  cost = hex(cost)
  For l = Len(cost) To 5
     cost = "0" & cost
  Next l
  If cost <> "00000" Then
  PIN = Text1.Text
   dm =  MakeCmdStr( COMMAND_INCMD, "fa20000008" & PIN)
   q =  SendCommand(1, UCase( dm),  result)


  End If
   result = Right( result, 7)
   result = Left( result, 4)
   result =  result Xor 9000
  If  result = 0 Then
  Y = datee
   dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
```

```
 q =  SendCommand(1, UCase( dm),  result)
 dm =  MakeCmdStr( COMMAND_INCMD, "fa30000008000000" & Y & cost)
 q =  SendCommand(1, UCase( dm),  result)

End If
 result = Right( result, 7)
 result = Left( result, 4)
 result =  result Xor 9000
   If  result = 0 Then
 dm =  MakeCmdStr( COMMAND_INCMD, "faa4000002ef03")
 q =  SendCommand(1, UCase( dm),  result)
'ShowText  result

 dm =  MakeCmdStr( COMMAND_OUTCMD, "fab200040f")
 q =  SendCommand(1, UCase( dm),  result)
x = Len( result) - 6
Mid( result, 1) = "00000"
Mid( result, x) = "000000"
 r1 = Left( result, 22)
r5 = Right(r1, 6)
Form6.Text3.Text = (HextoDec(r5)) / 100

 CloseCom

Form7.Hide
 Open "e:\debit.doc" For Append As #1
g = Date & " " & Time
Print #1, g
g = Form6.Text1 & " " & Form6.Text2.Text
Print #1, g
```

```
    g = "Debit " & i & "Bath"
    Print #1, g
    g = "New balance" & " " & Form6.Text3.Text & " " & "Bath"
    Print #1, g

    g = "_____"
    Print #1, g
    Close
      End If
    q =  openport(2, "9600,N,8,1")
    q =  PowerOff( result)
End Sub


Private Sub Command2_Click()
    Form7.Hide

End Sub
```

form8

```
Private Sub Command1_Click()
Dim  q As Integer
Dim  result As String
    Key = Text1.Text
    If Key = "key" Then
     q =  openport(2, "9600,N,8,1")
     q = PowerOn( result)
     q = InsertCard( result, "Insert card into reader.")
    If Check1.Value = 1 Then
     a = "31"
```

```
Else: a = "30"
End If
If Check2.Value = 1 Then
b = "31"
Else: b = "30"
End If
If Check3.Value = 1 Then
c = "31"
Else: c = "30"
End If
If Check4.Value = 1 Then
d = "31"
Else: d = "30"
End If
e = "30303030"
e = a & b & c & d & e

 dm =  MakeCmdStr( COMMAND_INCMD, "faa40000029902")
 q =  SendCommand(1, UCase( dm),  result)
 dm =  MakeCmdStr( COMMAND_INCMD, "fa200000899999999999999999")
 q =  SendCommand(1, UCase( dm),  result)
 dm =  MakeCmdStr( COMMAND_INCMD, "fad6005008" & e)
 q =  SendCommand(1, UCase( dm),  result)
Else
MsgBox "Incorrect Permit Key:"
End If
Form8.Hide
Form5.Show
End Sub
```

com.bas

```
Sub CloseCom()

   On Error GoTo ErrorCloseCom

   main!Comm1.PortOpen = False   'Com1
   Exit Sub

ErrorCloseCom:

End Sub


Function  openport(ByVal CPort As Integer, ByVal Portset As String) As Integer '(v_chaine As
String) As Integer
   'Open communication port that is selected by CurrentCom variable
   'input:  CurrentReader (0=Daughther, 1=Lot, 2=Key)
   'output: return open com port status (true=can open/false=cannot open)

   On Error GoTo Error openport
   'open comport
   If main!Comm1.PortOpen = True Then main!Comm1.PortOpen = False
   main!Comm1.CommPort = CPort
   If Portset <> "" Then main!Comm1.Settings = Portset
   main!Comm1.InputLen = 0
   main!Comm1.PortOpen = True 'open port
   If main!Comm1.PortOpen <> True Then GoTo Error openport
    openport = True
   Exit Function
```

```
Error openport:
    openport = False

End Function



Function ReadCom(s As String) As Integer
   'Read from communication port which selected by CurrentCom variable
   'input:  none (s is the return string for read result)
   'output: return read status (true=can read/false=cannot read)

   On Error GoTo ErrorReadCom

   'check com port status
   If main!Comm1.PortOpen = False Then
      GoTo ErrorReadCom
   End If

   'read from communication port
   s = main!Comm1.Input
   ReadCom = True
   Exit Function

ErrorReadCom:
   ReadCom = False
   'Resume Next

End Function
```

```
Function WriteCom(ByVal s As String) As Integer
   'Write "s" to communication port that is currently selected by CurrentCom variable
   'input:  s string tp send to reader
   'output: return true if write sucessfully

   On Error GoTo ErrorWriteCom

   'test the com port status
   If main!Comm1.PortOpen = False Then
      GoTo ErrorWriteCom
   End If

   'read from card
   main!Comm1.Output = s

   WriteCom = True
   Exit Function

ErrorWriteCom:
   WriteCom = False
   'Exit Function

End Function
```

function

```
Public Function AsciiToChar(ascii As String) As String
 Dim a As String, i As Integer, b As String, c As Integer
 Dim d, e, f As String
  Dim z(1 To 45) As Integer

 ' a = Text2.Text
  a = ascii
  'Text1.Text = ""
  For i = 1 To Len(a) Step 2
    d = Mid(a, i, 2)
    If d <> "00" Then
    f = f & d
    End If
  Next i
  e = ""
  For i = 1 To Len(f) Step 2
    b = Mid(f, i, 2)
    c = Val("&h" & b)
    e = e & Chr(c)
    'Text1.Text = Text1.Text & Chr(c)
    'Text1.Text = e
    AsciiToChar = e
  Next i
End Function


Public Function CutCode(iblock As String) As String
  Dim code As String * 8
  CutCode = Left(iblock, 8)
End Function
```

```
Public Function CutUnit(iblock As String) As String
 Dim unit As String * 1
   unit = Right(iblock, 2)
   CutUnit = unit
End Function


Public Function datee() As String
   a = Date
   dd = Day(a)
   mm = Month(a)
   yy = Year(a)
   X1 = (mm And (&H8)) / 8
   X2 = ((yy - 1980) And (&H7F)) * 2
   Y1 = hex(X1 Xor X2)
   Y2 = hex(dd Xor ((mm And (&H7)) * 32))
   datee = Y1 & Y2
End Function
```

tlp224.bas

```
Function  SendCommand(LRC As Integer,  dm As String,  result As String) As Integer
   Dim  q As Integer
   Dim s As String, Start As Single, rs As String

    SendCommand = False
   s =  dm
   Select Case LRC
```

```
    Case 0  'do not put LRC at the end of command
    Case 1  'put LRC at the end of command
      s = s &  CalLRC(s)
    Case Else
       SendCommand = False
      Exit Function
  End Select
  s = s & Chr(3)
  'flush remain input buffer
  ' result = Main!Comm1.Input
   result = ""
  'send command to reader
   q = WriteCom(s)
  If  q = False Then Exit Function
  'do a delay loop here (1 sec)
  Start = Timer
   result = ""
  Do
     'DoEvents
      q = ReadCom(rs)
      result =  result & rs
     If  q = False Then Exit Function
  Loop Until (Timer - Start) > 5 Or Right( result, 1) = Chr(3)
  'receive answer from reader
  ' result = Main!Comm1.Input
  ' q = ReadCom( result)
  'If  q = False Then Exit Function
   SendCommand = True
End Function
```

```
Public Function  MakeCmdStr(ctype As Integer, s As String) As String
   Dim t As String, length As String
   length = AsciiToHexStr(Chr(1 + Len(s) / 2))
   t = "60"
   'CType is command type according to TLP-224 protocol
   Select Case ctype
      Case  COMMAND_ICC_ON      'ICC power on (if already on then off)
         t = t & "046E" & s & "0000" 'where s is the delay time (from "00" to "FF")
      Case  COMMAND_ICC_OFF     'ICC power off
         t = t & "044D" & s & "0000" '--may be wrong
      Case  COMMAND_INCMD       'incomming (to card)
         t = t & length & "DA" & s
      Case  COMMAND_OUTCMD      'outgoing (from card)
         t = t & length & "DB" & s
      Case  COMMAND_VERSION     'version report
      Case  COMMAND_WARM_RESET  'warm reset
      Case  COMMAND_LED         'read statis
   End Select
   'add LRC and chr(3)
   't = t &  CalLRC(t) & Chr(3)
    MakeCmdStr = t
End Function


Function  InsertCard( result As String, msg As String)
   Dim  q As Integer
```

```
    InsertCard = False
    Do
        q =  PowerOn( result)
        'ShowText " result=" &  result
        'If Left( result, 4) <> "6010" Then  'error, no card
        'If Left( result, 14) = "6004FB280300B4" Then
        If Left( result, 4) = "6004" Then
            q = MsgBox(msg, vbCritical + vbRetryCancel, "Error")
            If  q = vbRetry Then
                q = False
            Else
                q = False
                Exit Function
            End If
        End If
    Loop While  q = False
    InsertCard =  q
End Function


Function  PowerOff( result As String) As Integer
    PowerOff =  SendCommand(1, "60044D010000",  result)
End Function


Function  PowerOn( result As String) As Integer
    PowerOn =  SendCommand(1, "60046E010000",  result)
End Function
```

GENERAL

The TLP 224 NV is fully compatible with the previous generation TLP 224 except for its fixed speed of 9 600 bauds.

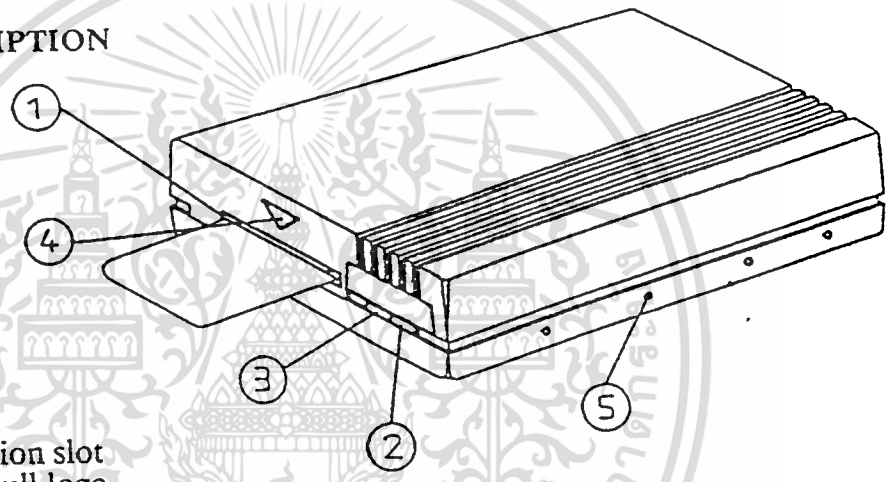
It has been developed:

- to revitalise the offer
- to reduce prices
- to expand existing markets and open up new ones.

It accepts all cards complying with ISO 7816 and the French Télécarte phonecard.

PHYSICAL DESCRIPTION

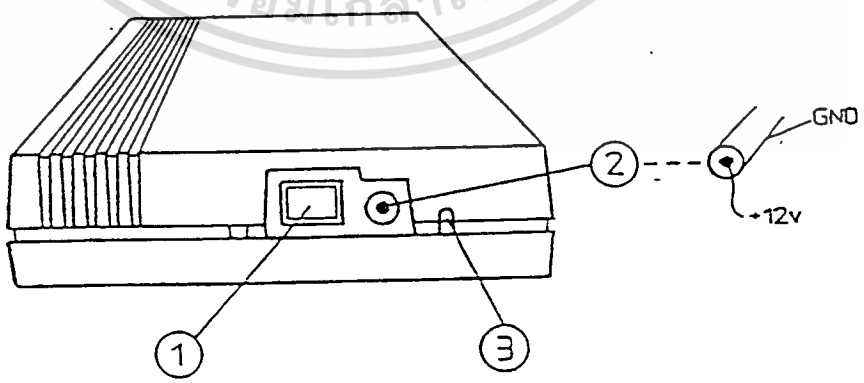
FRONT VIEW



. Front panel:
1 Horizontal card insertion slot
. Two LEDs under the Bull logo
2 Green : power on
3 Yellow : card powered up
4 "Insert card here" symbol
5 Fixing holes for incorporation in a PC

. Dimensions : 12 cm x 10 cm x 2.5 cm
. Weight : 200 g

REAR VIEW

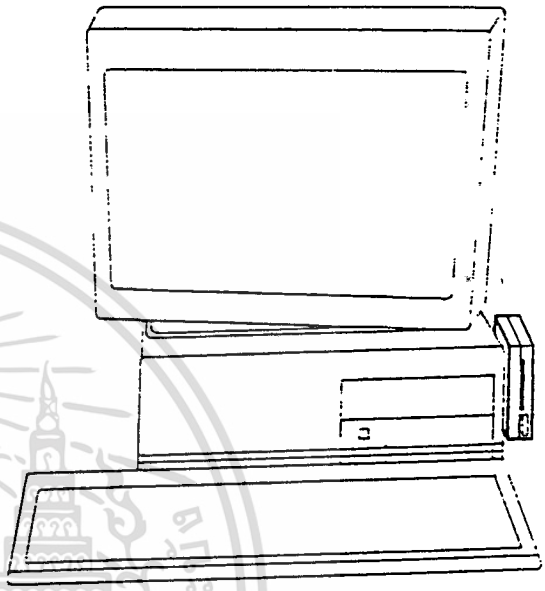


1 8-way modular jack (for CBL 2842 cable)
2 +12 V DC input jack
3 Reset button accessible with a pointed object such as the tip of a pencil

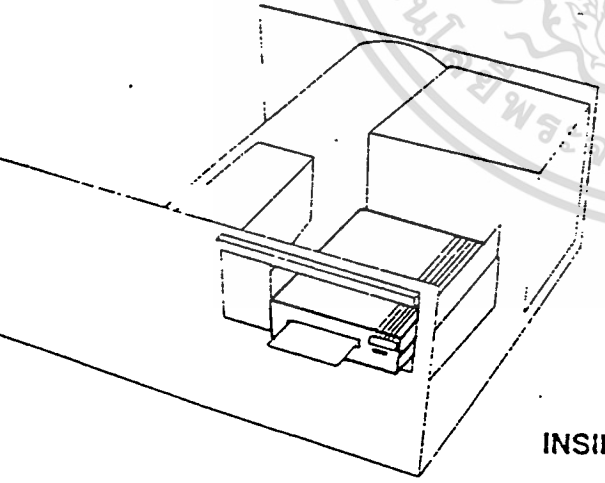
The reader/encoder can be installed:

- near the PC,
- on the PC,
- inside the PC.

NEAR THE PC: Apply the four stick-on feet

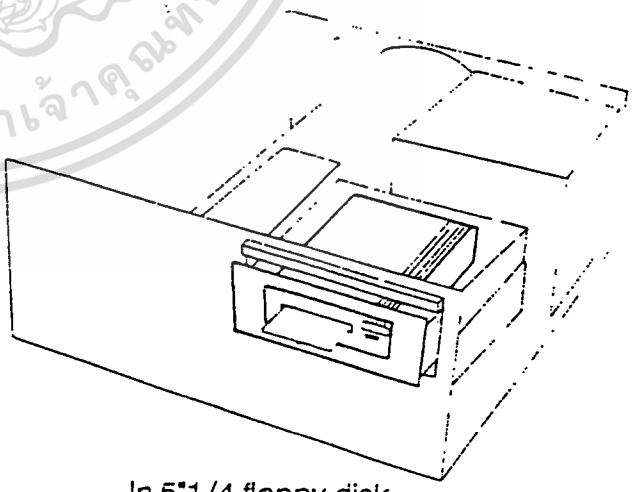


ON THE PC: Use the four Velcro strips to secure the reader/writer to the PC



In 3"1/2 floppy disk drive bay

INSIDE THE PC



In 5"1/4 floppy disk drive bay

The integration kit supplied includes the necessary mechanical fastenings.

OPERATION AND USE

The TLP 224 NV manual insertion smart card reader/writer is connected to a workstation via an RS 232 C asynchronous link operating at 9 600 bauds.

It accepts the following types of smart card:

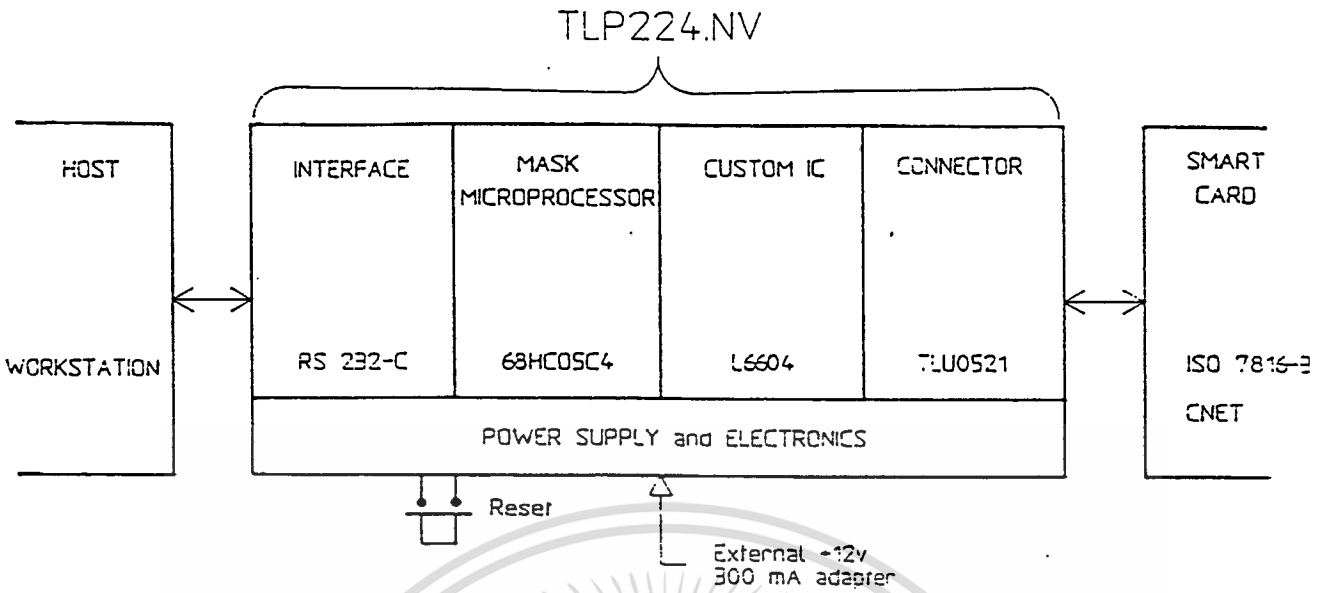
- CNET standard phonecards (Télécarte); CNET document: ST/PAA/TPA/PRI/1069 - Edition 3.

- Cards fitted with the STM TS 1200 or compatible component.

- Microcomputer cards to the ISO 7816-3 international standard, subject to the following restrictions:

  - card initial frequency : Fo = 3.579545 MHz,
  - card working frequency : F = 1 (fs = fo),
  - asynchronous dialogue mode only,
  - low active reset only,
  - 9 600 bauds dialogue only,
  - Icc (card) limited to 110 mA,
  - Ipp (card) limited to 50 mA.

The TLP 224 NV is a transparent card interface in the sense that it executes instructions and generates responses but does not process the data. Cards are accessed using a set of six commands, including two specifically for CNET standard phonecards.



. RS 232 C INTERFACE

Four signals constitute the dialogue between the TLP 224 NV reader/writer and the host (workstation or PC):

ED Data transmitted from TLP 224 NV.

RD Data received by TLP 224 NV.

RTS Request to send from TLP 224 NV.

DTR (Data terminal ready) TLP 224 NV powered up.

. MASK MICROPROCESSOR

This microprocessors manages the strobe (timing) signals and the dialogue with the host and with the card.

. CUSTOM IC

This IC manages the card power supply, programming voltage and clock signals.

It also protects the card interface against short-circuits.

. CONNECTOR

The connector has two sets of 8 clamp ("touchdown") action contacts plus a "card present" contact.

It accepts all smart cards, embossed or otherwise.

It accepts cards with the microchip in the following locations:

- top/recto/front
- bottom/recto/front
- bottom/verso/back

} according to the ISO 7816-2 standard

Insertion and extraction force < 7,5N

Service life > 100 000 operations

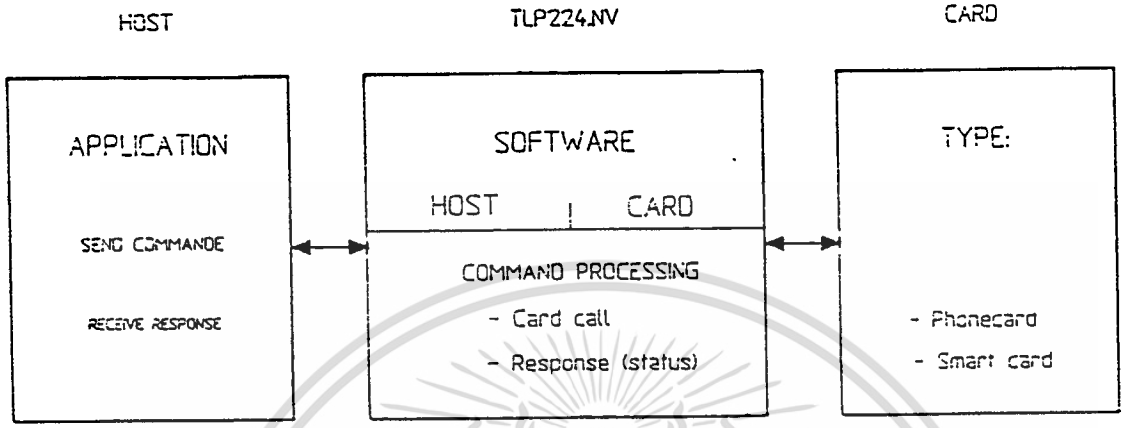
. POWER SUPPLY AND ELECTRONICS

The external mains adapter provides the +12 V, 300 mA supply.

The electronics on the circuit board regulates the very low voltage and handles microprocessor Reset generation and monitoring functions.

The microprocessor can be reset manually by pressing the Reset button at the rear of the TLP 224 NV.

COMMUNICATION PROCEDURE



. Each exchange has three phases:

- A command is sent from the Host to the TLP 224 NV.
- The TLP 224 NV processes the command.
- The Host receives from the TLP 224 NV the response relating to execution of the command.

. Exchanges are timed as follows:

The TLP 224 NV switches to receive mode:

- on power up (DTR signal),
- after sending its response.

The TLP 224 NV switches to transmit mode:

- when ready to send its response to the Host (RTS signal).

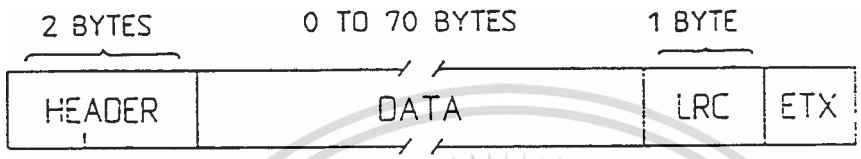
. Dialogue takes place at 9 600 bauds.

. Character format:

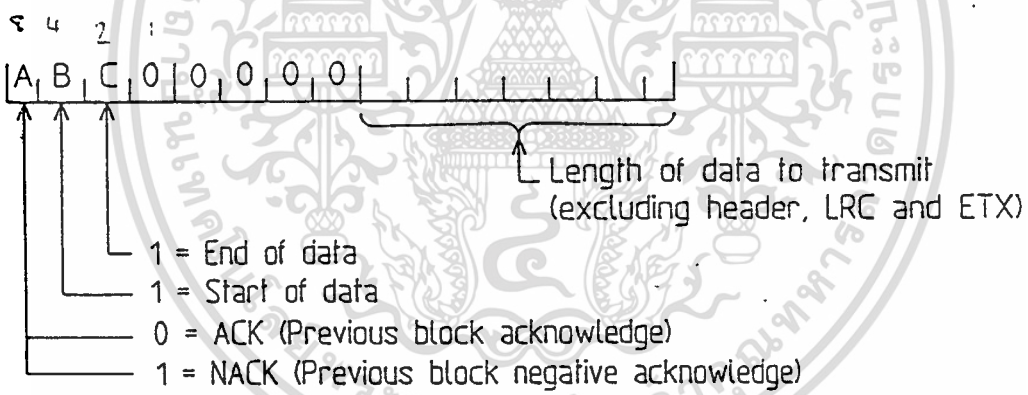
- 1 start bit,
- 8 data bits,
- no parity bit,
- 1 stop bit.

. Data is exchanged between the host and the TLP 224 NV in blocks, each made up of binary characters on one byte:

- 2 header characters,
- 0 to 70 data characters,
- 1 LRC character,
- 1 ETX character.



The two header bytes comprise:



Note:

The maximum block length compatible with the communication protocol is 70 bytes excluding header, LRC and ETX characters.

. The LRC (Longitudinal Redundancy Check) byte is the result of applying a series of exclusive-OR operations to the header and data bytes.

. During transmission on the ED or RD lines (TLP 2244 NV –> Host or Host –> interface controller), all characters of the block to be transmitted (except ETX) are converted into displayable ASCII characters and therefore split into two bytes.

. The commands are sent by the application included in the Host software.

They are processed by the TLP 224 NV software.

. Command format

Instr and N bytes	- Parameters. - Input data.

. Response format

Status and N bytes	- Parameters. - Output data.

. List of instructions

INSTR	MEANING/REMARKS
6E	Power up (and/or reset) smart card.
4D	Power down smart card.
DA	"Transparent" incoming instruction. On receiving this instruction the TLP 224 NV sends to the smart card the bytes constituting the incoming instruction and then manages the exchanges according to ISO standard 7816-3.
DB	"Transparent" outgoing instruction. On receiving this instruction the TLP 224 NV sends to the smart card the bytes constituting the outgoing instruction and then manages the exchanges according to ISO standard 7816-3.
72	Read Phonecard.
BO	Write Phonecard.

	HOST/TLP 224 NV INTERFACE
STATUS	MEANING/REMARKS
00	Command executed correctly.
03	Byte receive error (with NACK message).
04	PDS 39 TLP 224 NV, unknown instruction.
05	LRC error. If error persists, check application and/or connections (with NACK message).
08	Length (2nd header byte) does not match received length. If error persists, check application (with NACK message).
	TLP 224 NV/CARD INTERFACE
A0	Card not supported by TLP 224 NV (with power up instruction).
A2	"Dumb" card (with power up instruction).
A3	Transferred byte parity error, card communication error (power up instruction).
E2	"Dumb" card (with instructions other than power up).
E3	As A3, check card powered up (with instructions other than power up).
E4	Inconsistent acknowledge byte from card (with incoming or outgoing instruction).
E5	Card has interrupted exchanges with TLP 224 NV, incorrect access attempt, check instruction sequence, write address.
E7 WARNING	TLP 224 NV tests ME1 and ME2 bytes sent by card. - If ME1 = 90 or ME2 = 0, TLP 224 NV sends this status to host.
F7	Card "snatched". This status is sent once only.
FB	Card absent.

PocketBook V3.1

SMART CARD

USER'S MANUAL

APPLICATION DATE : 10/17/1994

INTERNAL DEVELOPMENT : P949
DOCUMENT REFERENCE : 941001

REDACTED BY

Service	Name	Function	Date	Visa
Development	Didier MARTIN	Engineer	10/17/1994	

APPROVED BY

Service	Name	Function	Date	Visa
Development	Olivier DEBELLEIX	Project manager	10/18/94	

1. INDEX :

2. CHARACTERISTICS :

=> Asynchronous card with ISO 7816-3 and 7816-4 compliance

=> Multi-file with 2-byte file identifiers and electronic purse management

=> Different file types: transparent (binary), record and electronic purses (2 types)

=> Each file stores 2 keys in its header, for secure access, with DES algorithm

=> Access conditions are separately defined, for each file, and for each access type (read, write, update)

=> Exchanges may be done in plain mode or ciphered mode, using a cryptogram or not

=> Card becomes mute only after detection of EEPROM failure, after having sent a particular error message

=> 2-byte checksum protects dynamically the EEPROM content (system and user areas)

=> Mirror zone (with checksum protection) prevents from any incomplete operation (accidental electrical problem, ...)

=> Data can be read from the card up to 255 bytes at once, when, due to internal limitations (mirror area length), more than 16 bytes cannot be sent to the card simultaneously

3. GENERAL DESCRIPTION :

3.1. Memory organization overview :

PocketBook is a micro controlled smart card . with 953 (Hex. 3B9)-user free EEPROM bytes. for files storing.
On delivery, one mandatory file exists in the card, called "embedder file". It is 40 bytes long, and its identifier is always 2F 00 (hex).
Here's an example of internal organisation :

embed = fix firmly

File :	Identifier :	Size :
Embedder file	2F 00	40
User file n° 1	EF 01	80
User file n° 2	EF 02	264
User file n° 3	EF 2A	120
User file n° 4	EF 2B	80
User file n° 5	43 56	48
User file n° 6	FD CC	123
Free remaining area for new files		240

Table 1 : File organisation example

3.2. File creation :

The PocketBook card allows to create as many files as wanted. with the only limit of space available for such files.
Creation of a new file is protected by IK key (Issuer Key).

Four different file types exist :

- Transparent file : the user accesses to data by byte address in the unique field of the file.
- Record file : the data are stored in fixed length fields. access is obtained by specifying a record number.
- EP3 file : electronic purse (3-byte amounts. ceilings & balances). This type only exists for compatibility with older versions of PocketBook. and should not be used any more.
- EP4 file : electronic purse (4-byte amounts. ceilings & balances).

Record and purse files contain records. 255 is the maximum enabled number of records for such files.
The file creation is processed by a single instruction operation. data field(s) are blank (0) after creation. For more information. please see chapter describing the "CREATE FILE" command (page 30).
The card operating system controls that each file has an unique identifier.

3.3. PocketBook command set :

PocketBook recognizes 18 different instructions, giving methods for direct and secured readings, writings and updatings to data (to be) stored in it.

Commands in alphabetical order

Command	Op code
ASK RANDOM	84h
CHANGE KEY	24h
CREATE FILE	E0h
DECREASE	30h
EXTERNAL AUTHENTICATION	82h
GET RESPONSE	C0h
GIVE RANDOM	86h
INCREASE	32h
INVALIDATE	04h
READ BINARY	B0h
READ RECORD	B2h
SELECT FILE	A4h
UPDATE BINARY	D6h
UPDATE CEILING	D6h
UPDATE RECORD	DCh
VERIFY PIN	20h
WRITE BINARY	D0h
WRITE RECORD	D2h

Table 2

Commands in numerical order

Op code	Command
04h	INVALIDATE
20h	VERIFY PIN
24h	CHANGE KEY
30h	DECREASE
32h	INCREASE
82h	EXTERNAL AUTHENTICATION
84h	ASK RANDOM
86h	GIVE RANDOM
A4h	SELECT FILE
B0h	READ BINARY
B2h	READ RECORD
C0h	GET RESPONSE
D0h	WRITE BINARY
D2h	WRITE RECORD
D6h	UPDATE BINARY
D6h	UPDATE CEILING
DCh	UPDATE RECORD
E0h	CREATE FILE

Table 3

3.4. Answer-To-Reset (ATR) sequence :

When powered on, PocketBook sends "Answer-To-Reset" sequence, consisting in 9 bytes described below :

3Bh 26h 00h H1 H2 H3 H4 H5 H6

This sequence always begins with 3Bh 26h 00h.

- Byte H1 contains MSB chip code : on V3.1, H1=06h for SGS-THOMSON ST16601
- Byte H2 contains LSB chip code : on V3.1, H2=01h
- Byte H3 is the version number (high nibble) and the mask numbers (low nibble) : H3=31h
- Byte H4 is a user definable byte, but it must be written during prepersonalization phase of card life, and cannot be further modified.
- Bytes H5-H6 are status bytes :

  If an EEPROM default has been detected, these bytes are 65h 01h, and card becomes mute, else :

  H5 is always 90h, or 91h if life counter has reach its end of life value (more than 100.000 writings into EEPROM) : life counter has no other effect on card use.

  H6 stores in its high nibble the number of registered bad PIN presentations.

  Then possible combinations of H5-H6 are :

65h 01h :	EEPROM defects, card is mute.
90h 00h :	No bad PIN presentation, normal life period.
90h 10h :	1 bad PIN presentation registered, normal life period.
90h 20h :	2 bad PIN presentations registered, normal life period.
90h 30h :	3 bad PIN presentations registered, normal life period, PIN is locked.
91h 00h :	No bad PIN presentation, end of life period.
91h 10h :	1 bad PIN presentation registered, end of life period.
91h 20h :	2 bad PIN presentations registered, end of life period.
91h 30h :	3 bad PIN presentations registered, end of life period, PIN is locked.

3.5. EEPROM protection by checksum :

The whole EEPROM contents is protected by calculation of a checksum after every writing in the EEPROM.

The checksum is made of a 16-bit addition with loop back of every outgoing (16th) carry bit to bit 0.

This value is checked in the very first steps of power on. When wrong, "Answer-To-Reset" message is modified (ending with 6501), and card becomes mute.

3.6. EEPROM protection by mirroring mechanism :

Every command that results in a change in EEPROM uses the mirroring mechanism : it consists, prior to the writing of new data, in saving old values in a special EEPROM area, and setting an EEPROM flag to know that a programming is in progress. Then the new values are written, and, if everything is correct, the flag is reset.
This mechanism has the disadvantage to slow down the card operation, but gives it a high level of security, thus it prevents from any partial writing or updating, in all conditions. On power on, the card operating system tests this flag, and, if set, restores old values in EEPROM.

For more protection, EEPROM areas used by mirroring mechanism are protected by a checksum (8-bit addition, with loop back of outgoing (8th) carry bit to bit 0).
This checksum is verified before restoring data if mirror flag is set, in other cases, it is not checked.
Checking is processed as follows: if stored checksum doesn't correspond to the one calculated with current checksum area, the card sends status word 65 01 at the end of Answer-To-Reset message, then becomes mute.

3.7. EEPROM life counter :

A 24 bit counter is implemented in system EEPROM area, which is incremented by one at every writing or updating of EEPROM. Its only purpose is to set a flag in "Answer-To-Reset" sequence (this flag is bit 0 of byte H5), as soon as a threshold value is reached.

When this flag appears in "Answer-To-Reset", it will never disappear, so the operator is informed that the card is in end of life period, thus more than 100.000 writes had occurred in some EEPROM cells.
It's up to the operator to decide what consideration he gives to this flag, no change will appear in card operation, except that EEPROM defects are subject to appear without more advice.
However, the operating system checks if each EEPROM programming has given a correct result through two different electrical level reading verifications : if a problem occurs, it sends 96-xx status word to cancel command, or it sends 65-01 and mutes the card, according to the [illegible] in which the defect appeared.

4. FILES STRUCTURE :

4.1. Transparent files :

	Hex Offset:	Fields :			
File Header 24 bytes	00h	Identifier		T/S1	S2
	04h	ACr	ACw	ACu	KUC
	08h	Secret Key #0			
	0Ch	SK0			
	10h	Secret Key #1			
	14h	SK1			
Data Area	18h	Data field			

Table 4 : Transparent file structure.

Meaning of header fields :

- **Identifier :**
  2-byte file identifier. according to ISO 7816-4. FF-FF identifier is reserved for internal use. and 2F-00 is reserved for "Embedder File" (see "Embedder file" description page 16).
- **T/S1 :**
  Two information are stored in this byte : the 3 most significant bits contain "T" which is the structure type information. and the 5 other bits are "S1" : the high part of size information. For a transparent file. T is 000. See just below description of S2 for the use of the 5 least significant bits ("S1").
- **S2 :**
  Second (low) part of the size information : S1:S2 is a 13-bit number that gives. in bytes. the size of EEPROM area reserved for this file (header+body lengths.
- **ACr :**
  Access conditions for reading. (p. 23)
- **ACw :**
  Access conditions for writing. (p. 23)
- **ACu :**
  Access conditions for updating. (p. 23)
- **KUC :**
  Keys Use Conditions. Tells which key is to be used for each access type. (p. 23)
- **SK0 :**
  Secret Key number 0. See SK1 for use description type.
- **SK1 :**
  Secret Key number 1. Secret Keys are used to protect access to file type data. according to ACx and KUC fields.

Available commands on transparent files are: "READ BINARY". "WRITE BINARY" and "UPDATE BINARY".

4.2. Record files :

	Hex Offset :	Fields :			
File Header 24 bytes	00h	Identifier		T/S	NB
	04h	ACr	ACw	ACu	KUC
	08h	Secret Key #0			
	0Ch	SK0			
	10h	Secret Key #1			
	14h	SK1			
Data Area	18h	Data records			

Table 5 : Record file structure.

Meaning of header fields :

- Identifier :
  2-byte file identifier, according to ISO 7816-4. FF-FF identifier is reserved for internal use, and 2F-00 is reserved for "Embedder File" (see "Embedder file" description page 16).
- T/S :
  Two information are stored in this byte : the 3 most significant bits contain "T" which is the structure type information, and the 5 other bits are "S" : the size of each record ; S must be lower or equal to 16, but not zero. For a record file, T is 001.
- NB :
  Number of records in this file : $0 < NB \leq 255$.
- ACr :
  Access conditions for reading. (p. 23)
- ACw :
  Access conditions for writing. (p. 23)
- ACu :
  Access conditions for updating. (p. 23)
- KUC :
  Keys Use Conditions. Tells which key is to be used for each access type. (p. 23)
- SK0 :
  Secret Key number 0. See SK1 for use description.
- SK1 :
  Secret Key number 1. Secret Keys are used to protect access to file data, according to ACx and KUC fields.

Available commands on record files are: "READ RECORD", "WRITE RECORD" and "UPDATE RECORD".

4.3. Electronic purse files (EP4) :

4.3.1. Header description :

	Hex Offset :	Fields :			
File Header 32 bytes	00h	Identifier		T/S	NB
	04h	ACr	ACi	ACd	KUC
	08h	Secret Key #0			
	0Ch	SK0			
	10h	Secret Key #1			
	14h	SK1			
	18h	Ceiling			
	1Ch	CP#1	CP#2	RFU	CHK
Data Area	20h	See below : "Records description"			

Table 6 : Electronic purse (EP4) file structure.

Meaning of header fields :

- **Identifier :**
  2-byte file identifier, according to ISO 7816-4. FF-FF identifier is reserved for internal use, and 2F-00 is reserved for "Embedder File" (see "Embedder file" description page 16).
- **T/S :**
  Two information are stored in this byte : the 3 most significant bits contain "T" which is the structure type information, and the 5 other bits are "S" : the size of each record : $8 \le S \le 16$. For a purse file, T is 111 (EP4 type).
- **NB :**
  Number of records in this file : $2 \le NB \le 255$.
- **ACr :**
  Access conditions for reading. (p. 23)
- **ACi :**
  Access conditions for increasing. (p. 23)
- **ACd :**
  Access conditions for decreasing. (p. 23)
- **KUC :**
  Keys Use Conditions. Tells which key is to be used for each access type. (p. 23)
- **SK0 :**
  Secret Key number 0. See SK1 for use description.
- **SK1 :**
  Secret Key number 1. Secret Keys are used to protect access to file data, according to ACx and KUC fields.

4.4. Electronic purse files (old type : EP3) :

4.4.1. Header description :

	Hex Offset:	Fields :			
File Header 32 bytes	00h	Identifier		S/T	NB
	04h	ACr	ACi	ACd	KUC
	08h	Secret Key #0			
	0Ch	SK0			
	10h	Secret Key #1			
	14h	SK1			
	18h	Ceiling			CHK
	1Ch	CP#1	CP#2	RFU	RFU
Data Area	20h	See below : "Records description"			

Table 8 : Electronic purse (EP3) file structure.

Meaning of header fields :

- **Identifier :**
  2-byte file identifier according to ISO 7816-4. FF-FF identifier is reserved for internal use, and 2F-00 is reserved for "Embedder File" (see "Embedder file" description page 16).
- **T/S :**
  Two information are stored in this byte : the 3 most significant bits contain "T" which is structure type information, and the 5 other bits are "S" : size of each record. **For a purse file, T is 011 (EP3 type).**
- **NB :**
  Number of records in this file : $2 \leq NB \leq 255$.
- **ACr :**
  Access conditions for read (p. 23)
- **ACi :**
  Access conditions for increasing. (p. 23)
- **ACd :**
  Access conditions for decreasing. (p. 23)
- **KUC :**
  Keys Use Conditions. Tells which key is to be used for each access type. (p. 23)
- **SK0 :**
  Secret Key number 0. See SK1 for use description.
- **SK1 :**
  Secret Key number 1. Secret Keys are used to protect access to file data, according to ACx and KUC fields.
- **Ceiling :**
  3-byte ceiling value for transactions in purse.

- CHK :
  Checksum of ceiling value. This checksum should be calculated and checked by the application ; the card doesn't take any care of this value.
- CP#1 :
  Current pointer to last operation on purse.
- CP#2 :
  Redundant current pointer to last operation on purse, for safe operation. Redundancy's checked by operating system : if result is bad, it sends 65 01 and mutes the card.
- RFU :
  2 bytes reserved for future use.

Available commands on purse files are: "READ RECORD", "INCREASE" and "DECREASE".

4.4.2. Record description :

	Hex Add	Fields :	
Fixed Area	0	Transaction number	MSB date
	4	LSB date	Balance
Optional Area	8	0 to 8 optional byte(s) for application data	
	C		

Table 9 : Electronic purse (EP3) record structure.

Meaning of record fields :

- Transaction number :
  3-byte transaction counter, incremented on every credit or debit operation made on the purse. The application can only read this value. There's no check made by the card on this counter, so it may roll-over to 0, if enough transactions were performed on the purse (2^{24} = 16Mega-operations !! that is highly improbable).
- Date (MSB & LSB) :
  These data are given by application when increasing or decreasing purse, they have no meaning for the card. So, any data can be stored here and may have no link with a date !
- Balance :
  3-byte purse current value managed by the card operating system.
- Optional bytes :
  On file creation, an optional area can be specified for every record. Optional data will be automatically updated while crediting or debiting purse, if sufficient data (more than 8 bytes) are are given to the "INCREASE" or the "DECREASE" command ; if there's not enough data in the "INCREASE" or "DECREASE" command data field, the remaining bytes are automatically blanked to zero.

4.5. Embedder file :

This file is created during the prepersonalization phase of life, by the embedder. Its purpose is to cancel cheat risks after card manufacturing step. It is identified by 2F 00, and structured as a transparent file. Therefore, its header size is 24 bytes long, and data area is 16 bytes long.

After its creation and the writing of data, this file is invalidated, so data will never more be modified. However, data may be read without restriction.

Data written in this file give information about card manufacturing and embedding.

Data organisation is described here :

Hex Offset:	Embedder file body :			
0	Manufacturer code		CHK	SKCM/SNP
4	Customer code	Embedder code	Serial number (MSW)	
8	Serial number (LSW)		Manufacturing day	Manufacturing month
C	Manufacturing year	Customer bytes		

Table 10 : Embedder file mapping.

Meaning of these bytes :

- **Manufacturer code :**
  2-byte manufacturer code is always FFh FBh.

- **CHK :**
  The checksum is the number of reset bits in bytes at offsets 3 to F.

- **SKCM/SNP :**
  SKCM means "System Keys Change Mode", and is stored in bits 6 & 7.
  SNP means "Serial Number Presence", and is stored in bit 0.
  Bits 1 to 5 are reserved for future use, and set to 0.

  SNP: this bit is set to 1 if a valid serial number is stored in bytes 6 to 9, else it is 0.

  SKCM: Bits 7 (named "SKCM1") & 6 (named "SKCM2") control mode (ciphering, certifying) used to change the system keys. (see "System and data keys change" chapter, page 24)

- **Customer code :**
  This byte, chosen by SOLAIC SMART CARD INDUSTRIES, can be set to any value between 00h and FFh, according to the customer commercial file.

- **Embedder code :**
  The SOLAIC SMART CARD INDUSTRIES' issuer code is A0h.

- **Serial number :**
  The serial number is written using a hexadecimal or BCD (customer choice) format, it's incremented for every card. SOLAIC SMART CARD INDUSTRIES guarantees that each number is unique (only if SNP is set and for a given customer number).

- **Manufacturing date :**
  The date written here is the prepersonalization date, coded in BCD (i.e. Feb. 12th 94 is coded 12h 02h 94h).

- **Customer bytes :**
  These bytes are customer-definable. They must be specified to be written during the prepersonalization step, because the file is write-protected on card delivery.

The only available command on embedder file is: "READ BINARY", the commands "WRITE BINARY" and "UPDATE BINARY" are forbidden since file is invalidated.

5. ELECTRONIC PURSE MANAGEMENT :

Two types of electronic purses are implemented in PocketBook smart card V3.1. We'll discuss here only the EP4 type, described above, because old EP3 type differs only on minimum area of record structure: it's got only one counter, without blocking value, and has an "application date" 2 byte area.

Purse files are derivated from record files, with additionnal internal management : except for records reading, an electronic purse file is automatically managed as a cyclic file, using the CP ("Current pointer") byte (duplicated to safe operations) in the file header. Only readings can (and must) be record-addressed. When crediting or debiting a purse, the operating system updates the record following the current record, then increments the CP, or, if the current record is the last one, it updates the first one, then sets the CP to 1.

A purse file must have at least 2 records, and could hold up to 255 records, if available EEPROM would permit it (on V3.1, if only one purse file is created, and no other file except the embedder file, available area for records is 925 bytes : if no optional area is required, max number of records becomes 115 ...).

The issuer can create in the PocketBook card as many electronic purse files as the EEPROM size can store. Transactions are stored one by record, and a "Current Pointer" always points to the last transaction record, allowing to read directly the current purse balance.

Two special commands are implemented for purse management, and replace standard record files commands "UPDATE RECORD" and "WRITE RECORD", which are forbidden for these purse files. These commands are "INCREASE", to credit the purse, and "DECREASE", to debit it.

The card operating system automatically rejects the "INCREASE" commands that would result in a balance greater than the ceiling value stored in the file header, and the "DECREASE" commands that would result in a negative balance.

A special formulation of the "UPDATE BINARY" command, named "UPDATE CEILING" (p. 43), allows to change a purse file ceiling.

The transactions are counted by the card, with separate 2-byte counters for credit and debit operations. So the maximum value for these counters is 65535. Thus, before executing a debit or a credit command, the card checks the counter, if it is already equal to the maximum, the command is rejected with error 94h 20h (see "Status words" chapter, page 27).

Record organisation is described here :

Hex Offset:	Electronic Purse (EP4) record	
0	Credit counter	Debit counter
4	Balance	

Table 11 : Electronic purse (EP4) record mapping.

This figure only describes the 8 first bytes of a purse file record, which are mandatory. Optional bytes can be added after main part of the record, from 0 to 8 ; their use is free. They can be updated with "INCREASE" and "DECREASE" command ; if these commands access a record with less data than the record length, data will be first padded with zeroes to record length. In fact, data field of a transaction command is composed of from 8 to record length bytes.

6. SECURITY MANAGEMENT :

6.1. Data exchanges :

6.1.1. Global overview :

Data exchanges between the terminal and the card may be done in 4 different ways : data may be plain, ciphered, certified or certified and ciphered.
Some commands permit the use of only one way (e.g. "VERIFY PIN" waits for text data), whereas other keep the choice between these modes, according to particular information (e.g. "READ BINARY" determines the mode to be used with the current file ACr byte).

In order to use ciphered and/or certified mode, the data must be organized in 8-byte blocks, because the cryptographic algorithm used is the DES ("Data Encryption Standard") which works on 8-byte blocks.

6.1.2. Data ciphering :

When data are sent to the PocketBook card in ciphered mode, the card operating system first deciphers data with DES^{-1} algorithm, using the key specified by the instruction and the current file header ACx byte (except for "CHANGE KEY" -p. 29- and "EXTERNAL AUTHENTICATION" -p. 33-), then operates data as needed.

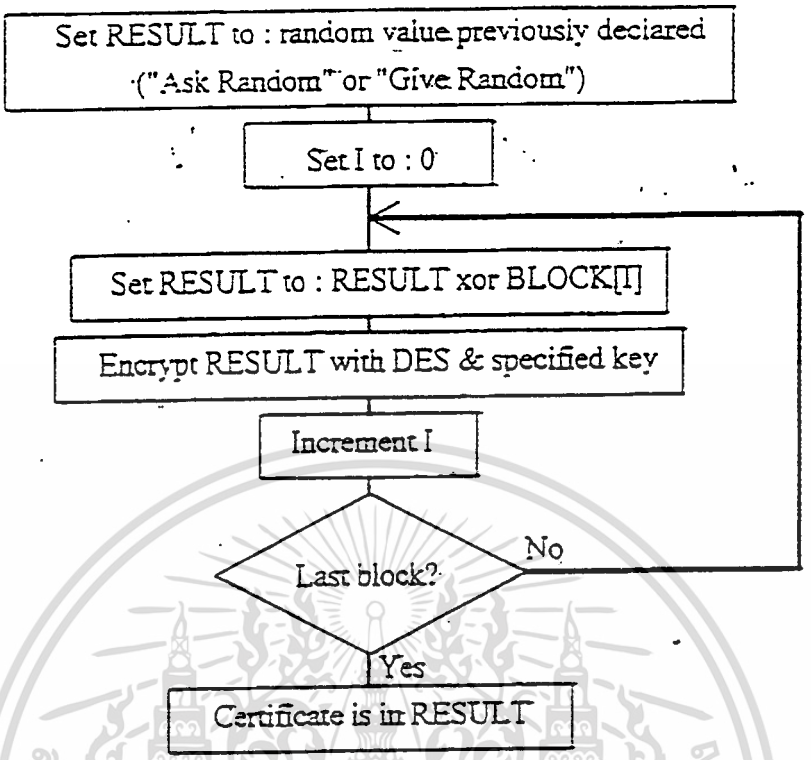
When data are to be sent by the PocketBook card in ciphered mode, they are first ciphered with DES algorithm using the key specified by the instruction and the current file header Acr byte.

6.1.3. Data certifying :

Before sending data to the PocketBook card in certified mode, the external application has to ask the card for a random 8-byte value ("ASK RANDOM" -p. 28-), then to calculate the certificate, with the principle of certifying described below. When the card operating system receives the certified data, it first calculates the same certificate, with the specified key and sent random, then compares the two certificates (the internal calculated and the received one); if equal, the data are then processed as needed, else a status word is returned, and no operation is performed.

Before expecting from the PocketBook card data in certified mode, a random value must be sent to the card with "GIVE RANDOM" command (p. 35). Then the card operating system calculates the certificate with the key specified by the current file ACr and KUC bytes, and sends the data, followed by the certificate.
Upon reception, the application should calculate the data certificate in the same way, and compare it with the card one.

A certificate is calculated with the following algorithm :



By "specified key", we mean the key selected with ACr, ACw or ACi, ACu or ACd and KUC bytes of current file. See "Access conditions" chapter (page 22) for details about these bytes.

The random value used as an initialization block for certifying data is sent to the card by a "GIVE RANDOM" command (page 35) when reading data, and given by an "ASK RANDOM" command (page 28) in other cases.

When certifying data, an header block called block 0, is treated before the true data. This block contains information relative to the current command and the current file, in order to prevent any certificate falsification.

Block 0 is built as follows (and represents the "Standard Block 0" appellation in the commands descriptions) :

- in case of a "READ RECORD" command :

CLA	INS	A1	A2	LNG	MSB-ID	LSB-ID	00h

A1 & A2 represent the logical address of the record in the file :

A1:A2=(record number-1) x (record size)[1]

- in case of all other commands :

CLA	INS	P1	P2	LNG	MSB-ID	LSB-ID	00h

P1 & P2 come from ISO-command header bytes.

CLA & INS come from ISO-command header bytes, and MSB-ID:LSB-ID is the current file identifier.
LNG is the useful data length, and is at most equal to P3-10h. In the case of an ingoing certified command, LNG may be a non-modulo 8 value, the last data block being 0-padded.

6.1.4. Data certifying and ciphering :

When data are to be exchanged in certified and ciphered way, certificate is calculated prior to the data ciphering. The ciphering is processed on all blocks except the certificate one.

6.1.5. Key presentation :

Other security level(s) can be used for files data access, by requiring one (or more) previous key(s) verification (passive or active for the PIN, and active for the other keys). The next chapter gives complete information about how to specify access requirements for data exchange.
Be careful that the keys are numbered a different way between the "CHANGE KEY" (p. 29) and the "EXTERNAL AUTHENTICATION" (p. 33) commands.

[1] 1st record is called record number 1

6.2. Access conditions :

Access conditions are controlled by 4 bytes stored in the file header, therefore they are different for every file.
These control bytes are named ACr (read), ACi (increase), ACd (decrease) and KUC for electronic purse files, and ACr, ACw (write), ACu (update) and KUC for other types of files.

They are located at the same positions in all the files header, as described previously, i.e. ACr is at offset 4, ACi or ACw are at offset 5, ACd or ACu at offset 6 and KUC at offset 7.

6.2.1. "ACx" bytes :

All ACx bytes are bit-coded as follows :

PIN	DK	CER	CIPH	DIS	CHK	CHK	CHK

Meaning of these bytes, when set :

- PIN : PIN must have been successfully presented (with "VERIFY PIN" or EXTERNAL AUTHENTICATION" commands) prior to the data access.
- DK : A data key (DK0 or DK1 according to KUC byte ; see next chapter) must have been successfully presented (with EXTERNAL AUTHENTICATION" commands) prior to the data access. A file's got two data keys, the number of the data key here requested is stored in the KUC byte, see below.
- CER : Data must be certified.
- CIPH : Data must be ciphered.
- DIS : Access is disabled.
- CHK : Checksum of the 5 bits described here above, which should be set and verified by application (the PocketBook card operating system doesn't manage these 3 bits). We suggest to store in theses bits the number of zeroed bits in the 5 significant bits, that will be performed by any "INVALIDATE" command (p. 37).

When a file is invalidated ("INVALIDATE" command, p. 37), ACw and ACu (respectively ACi and ACd for a purse file) are set to 0Ch, i.e. DIS is set, other (four) bits are zeroed, so CHK is 4.

Examples of ACx coding can be found in the "PocketBook samples" chapter, p. 52.

6.2.2. "KUC" byte :

KUC means Key Use Conditions.

There's one KUC byte per file header, and it's mainly used to choose which between the two data keys DK1 and DK2 must be taken into account for each data access type (reading, writing/crediting or updating/debiting). It also allows to specify a security mode for changing the data keys.

KUC bytes are bit-coded as follows :

DKCM1	DKCM2	RFU	ACr-Kn	RFU	ACw/i-Kn	RFU	ACu/d-Kn

Meaning of these bytes :

- RFU : RFU bits are "Reserved for Future Use".
- DKCM1 : When set, bit DKCM2 has no meaning, and keys can be changed in plain text mode. If zero, key change mode is specified by DKCM2 byte.
- DKCM2 : Useful only if DKCM1 is reset. In this case, when set, key change must be done in certified and ciphered mode, when reset, key change must be done in ciphered but *not* certified mode.
- ACr-Kn : This bit gives data key number (0 or 1) for DK condition in ACr.
- ACw/i-Kn : This bit gives data key number (0 or 1) for DK condition in ACw (or ACi if purse file).
- ACu/d-Kn : This bit gives data key number (0 or 1) for DK condition in ACu (or ACd if purse file).

Examples of KUC coding can be found in the "PocketBook samples" chapter, p. 52.

6.2.3. System and data keys change :

System keys (IK, UK, CCK and PIN) and data keys (DK0 and DK1 in every file header) can be changed, according to the conditions written in the "SKCMi" bits in embedder file (p. 16) for system keys, and in the "DKCMi" bits in "KUC" file header byte (p. 23) for data keys.

Before a key can be changed, it must have been successfully presented, except for the data keys which require IK presentation.

Notice that these bits have the same names and the same positions (bits 6 & 7) in their respective byte.

The table below shows the required mode to change a key.

SKCM1/DKCM1	SKCM2/DKCM2	Key change mode
0	0	ciphered
0	1	ciphered and certified
1	x	plain

Table 12 : System & data keys change mode.

6.3. Keys description :

Keys exist in PocketBook at two different levels. There are, first, the system keys, which are stored in EEPROM system area. The system keys are :

- PIN : Personal Identification Number
- CCK : Change Ceiling Key
- UK : Unlocking Key
- IK : Issuer Key.

The other key level is the file level : every file gets two local keys called DK0 and DK1 : "Data Key number 0" & "Data Key number 1".

All these keys can be presented to the card with the "EXTERNAL AUTHENTICATION" command (p. 33). The PIN can also be verified in plain mode with the "VERIFY PIN" command (p. 46).

The PIN and the DKi can be used as conditions to access to the files data, according to the files header ACx and KUC bytes (p. 22).

The CCK must be correctly presented before performing a modification of the ceiling in a purse file. operation done with a special case of "UPDATE BINARY" command, called "UPDATE CEILING" - see this command to get more information about ceiling change on p. 43-.

The UK has the only use of unlocking the PIN bad presentations counter when locked by 3 or more wrong active or passive PIN presentations. To unlock a locked PIN, the UK must be correctly presented, and this must be followed by a good active or passive presentation of the PIN.

The IK must have been presented, prior to any file creation.

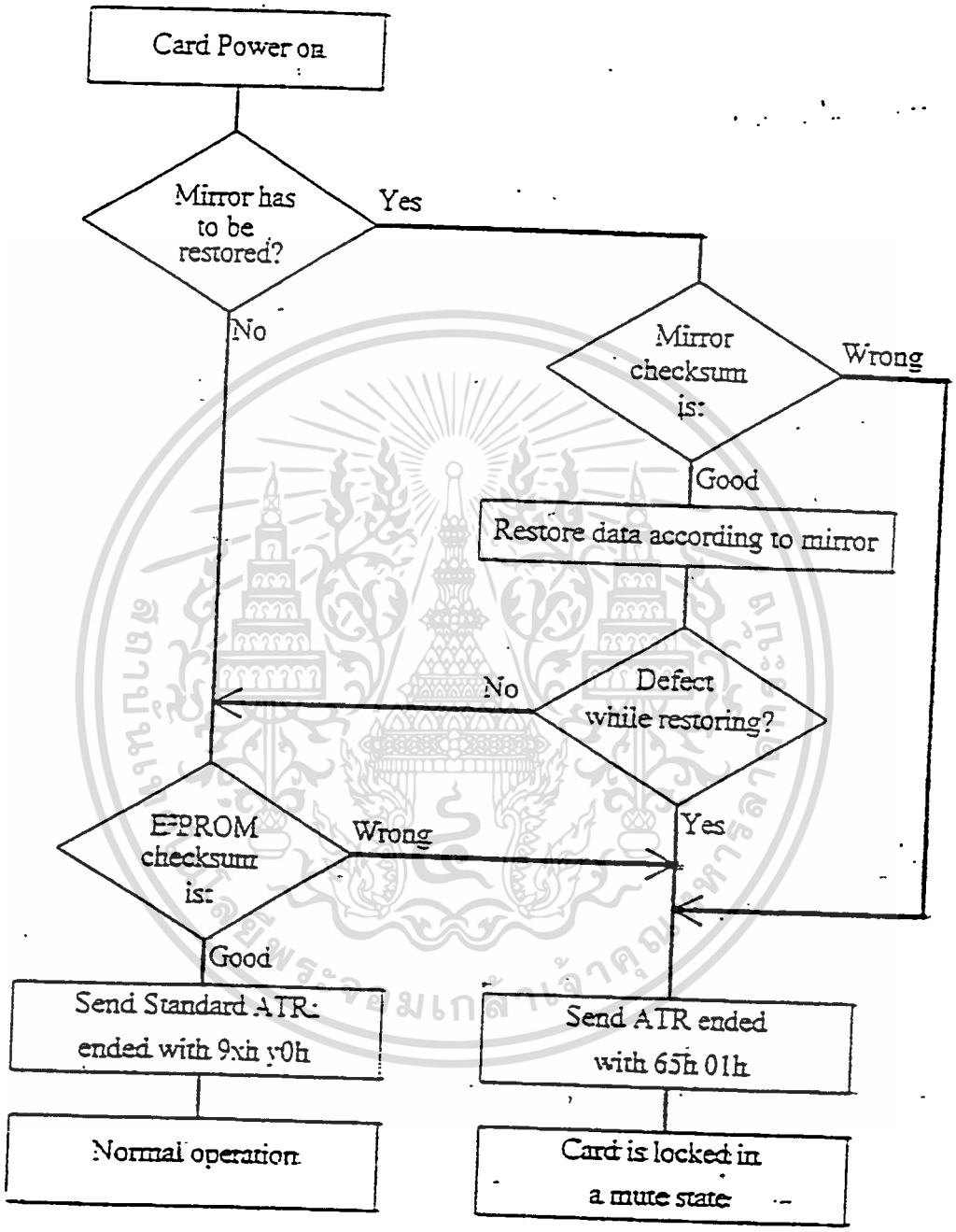
The rights of a key presentation are :

- for PIN, CCK and IK, acquired for the whole session ;
- for UK, acquired for the whole session, except in one case : the rights of UK are lost after the next PIN presentation, on a card in locked-PIN state;
- for DK0 and DK1, valid until the next "SELECT FILE" command, or the end of the session

Examples of key values can be found in the "PocketBook samples" chapter, p. 52.

6.4. EEPROM protection mechanism :

On power-on, the PocketBook card checks for EEPROM defects, before completion of "Answer-To-Reset" sequence sending. These checks follow the next scheme :



7. STATUS WORDS :

Status word :	Meaning :
65h 01h	Critical problem in EEPROM ; card is mute.
67h 00h	Bad P3 in ISO-command header.
6Bh 00h	Bad P1 and/or P2 in ISO-command header.
6Dh 00h	Unknown command (bad INS in ISO-command header).
6Eh 00h	Bad command class (bad CLA in ISO-command header).
90h 00h	No error, execution is complete.
94h 10h	INCREASE command results in balance exceeding ceiling, or DECREASE command results in negative balance.
94h 20h	INCREASE command results in credit counter overflow, or DECREASE command results in debit counter overflow.
96h 10h	Physical problem during EEPROM writing (type 1).
96h 20h	Physical problem during EEPROM writing (type 2).
96h 30h	Physical problem during EEPROM writing (type 3).
98h 10h	Wrong PIN presentation, 1st try.
98h 20h	Wrong PIN presentation, 2nd try.
98h 30h	Wrong PIN presentation, last (or more) try, PIN is locked.
98h 40h	File or record size too high or too small.
98h 50h	File doesn't exists, or file already exists, or identifier is forbidden.
98h 60h	"GIVE RANDOM" or "ASK RANDOM" command not previously run.
98h 70h	Wrong certificate or, in case of system commands[2] : key not presented, or wrong key.
98h 80h	Access conditions are not fulfilled, or file is invalidated.
98h 90h	Command is not applicable to current file.

Table 13 : Status words.

[2] "CREATE FILE", "INVALIDATE", "CHANGE KEY", "EXTERNAL AUTHENTICATION" & "UPDATE CEILING"

8. COMMAND SET :

NOTA: In every command containing a P1 and/or a P2 parameter specified as "not used", we strongly recommend to set it/them to 00h, as they may become used in further versions, and ascending compatibility will, in such cases, be guaranteed by keeping previous versions characteristics with null values, where "not used" parameters previously exist.

8.1. ASK RANDOM :

Asks the card for an 8-byte random value.

Byte	Contents
Class	FAh
Command	84h
P1	not used
P2	not used
P3	08h
DATA	8-byte random number

Table 14 : "ASK RANDOM" command.

This command must be run before any active key authentication, or certified writing or updating.

A random value is valid only for the next instruction.

Possible status words for "ASK RANDOM" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 90-00 if the command's successfully completed.

8.2. CHANGE KEY :

Allows to modify or initialize any secret key of the card.

Byte	Contents :			
Class	FAh			
Command	24h			
P1	not used			
P2	Key number :	00h : UK	01h : PIN	02h : IK
		03h : DK0	04h : DK1	05h : CCK
P3	18h if certified mode, else 08h			
DATA	If certified mode : standard Block 0 (p. 22).			
	8-byte new key (plain or ciphered).			
	If certified mode, 8-byte certificate (p. 21)			

Table 15 : "CHANGE KEY" command.

Before being changed, a key must have been successfully presented, except for the data keys (DKi), which expect a previous valid IK presentation. In the case of an initialization, the key to be presented is the virgin one (00h ... 00h).
In certified and/or ciphered modes, the keys are ciphered or signed by their own old value, except for the data keys (DKi), which are ciphered or signed by IK (see "System & data keys change" p. 24).

The change mode is selected by the SKCMi bits of "SKCM/SNP" byte in Embedder file (p. 16) header for all keys, except for the data keys (DKi) for which the mode is driven by the DKCMi bits of KUC byte (p. 23) in the current file header.

Be careful that the keys are numbered a different way between this command and the "EXTERNAL AUTHENTICATION" command.

Possible status words for "CHANGE KEY" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 96-x0 if mirror writing defects.
- or 65-01 if case of a key writing defects.
- or 98-60 if no "ASK RANDOM" has been previously run (in certified mode),
- or 98-70 if something is wrong in the ciphering or in the certificate, or if the old (or virgin if initialization) key hasn't been previuosly verified.
- or 90-00 when the change has been done.

8.3. CREATE FILE :

Enables to create a new file in EEPROM. if the file total size is lower or equal to the remaining free memory space.

Byte	Contents			
Class	FAh			
Command	E0h			
P1	not used			
P2	not used			
P3	18h for transparent or record file (or 10h. for ascending compatibility), 1Ch for Electronic purse file (or 14h. for ascending compatibility).			
DATA	*Transparent file*	*Record File*	*EP3 Purse file*	*EP4 Purse file*
01h	MSB file-Id			
02h	LSB file-Id			
03h	T(000)-S1	T(001)-S	T(011)-S	T(111)-S
04h	S2	NB		
05h	ACr			
06h	ACw		ACi	
07h	ACu		ACd	
08h	KUC			
09h-10h	DK0			
11h-19h	DK1 (only if P3≥18h)			
19h-1Bh	non existent		3-byte Ceiling value	3 MSB bytes of ceiling value
1Ch	non existent		CHK	LSB of ceiling value

Table 16 : "CREATE FILE" command.

Before creating a file. the IK must have been successfully presented.

Transparent files have their size coded in the bytes T/S1 and S2. that gives a 13-bit number : the maximum theoretical size for a transparent file is then 2^{13}= 8192 bytes.
Do not forget that user free space on EEPROM (for V3.1) at delivery is only 953 bytes. The additional capability is for next generations of the **PocketBook** smart card.

For ascending compatibility the P3 byte can hold the values 10h or 14h in place of 18h or 1Ch. In these cases. DK1 is automatically filled with 0FFh's.

In the case of an EP3 purse file. the "Ceiling value" is a 3-byte value. and the fourth byte is a checksum byte. which must be handled by the terminal application.

Many files as wanted can be created. with the only limit of the EEPROM available space. But the user mustn't forget that a file cannot be deleted. once successfully created.

Possible status words for "CREATE FILE " are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header,
- or 96-x0 if mirror writing defects,
- or 65-01 if file header writing defects,
- or 98-40 if the relevant file size or its record size is too short or too long,
- or 98-50 if the relevant identifier is already allocated to a file; or if it is forbidden,
- or 98-70 if the IK has not been previously successfully presented,
- or 90-00 when the file creation has been done.

8.4. DECREASE :

Enables to debit an electronic purse file :

Byte	Contents	
Class	FAh	
Command	30h	
P1	00h (not checked in not certified modes)	
P2	04h (not checked in not certified modes)	
P3	if plain text mode : from 08h to 10h if ciphered *un*certified mode : 08h or 10h if certified (ciphered or not) mode : 18h or 20h	
DATA	*EP3*	*EP4*
	If certified mode : standard Block 0 (p. 22).	
01h-03h	00h	
04h	MSB date	00h
05h	LSB date	MSB amount
06h-08h	3-byte amount	3 low bytes of amount
09h-rec size	Free use additional byte(s) (0 or 8-byte if certified and/or ciphered mode)	
	If certified mode : 8-byte certificate (p. 21).	

Table 17 : "DECREASE" command.

To be allowed to perform a "DECREASE" command, the access conditions specified in the current file header ACd and KUC bytes must be fulfilled.

Before debiting the purse, the card checks if this will result neither in a negative balance, nor in a debit counter overflow[1] for EP4 only : no check is performed on EP3 counter!

Possible status words for "DECREASE" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 94-10 if the debit would result in a negative balance.
- or 94-20 if the debit would result in a debit counter overflow[1].
- or 96-x0 if mirror writing defects.
- or 65-01 if purse data updating defects.
- or 98-60 if no "ASK RANDOM" has been run prior to a ciphered and/or certified debit.
- or 98-70 if something is wrong in the ciphering or in the certificate.
- or 98 8[illegible] if the access conditions aren't fulfilled.
- or 98 9[illegible] if the current file is not a purse file.
- or 90-00 when the debit has been done.

[1] Reminder on "INCREASE" command, even with a zero amount, resets debit counter.

8.5. EXTERNAL AUTHENTICATION :

Enables to present any key, by ciphered way, thus without transit of key true value.

Byte	Contents			
Class	FAh			
Command	82h			
P1	not used			
P2	Key number :	0h : UK 03h : PIN	01h : IK 04h : DK1	02h : DK0 05h : CCK
P3	08h			
DATA	8-byte random given by previous "ASK RANDOM" ciphered with the secret key.			

Table 18 : "EXTERNAL AUTHENTICATION" command.

This command must have been performed on the IK key prior to run a "CREATE FILE" (p. 30) or an "INVALIDATE" (p. 37) command.

For PIN authentications, an EEPROM mapped ratification counter increments on every bad (active or passive) presentation, up to 3. When it reaches this value, the card gets in a locked-PIN state, where all the commands expecting a good PIN presentation are forbidden, till an UK presentation followed by a good PIN presentation are successfully performed.

Be careful that the keys are numbered a different way between this command and the "CHANGE KEY" command.

Possible status words for "EXTERNAL AUTHENTICATION" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 96-x0 if mirror writing defects (PIN authentication only),
- or 65-01 if counter writing defects (PIN authentication only),
- or 98-10, 98 20 or 98 30 for respectively first, second and third or more bad PIN presentation -active and passive ("VERIFY PIN") presentations merged-.
- or 98-60 if no "ASK RANDOM" has been previously run.
- or 98-70 if something is wrong in the ciphering (except for PIN, for which the result is 96 x0): bad key or bad calculation.
- or 90-00 when the key presentation has succeeded.

8.6. GET RESPONSE :

Gives information about the current file.

Byte	Contents			
Class	FAh			
Command	C0h			
P1	not used			
P2	not used			
P3	01h to 17h			
DATA	*Transparent file*	*Record File*	*EP3 Purse File*	*EP4 Purse file*
01h	85h			
02h	P3 - 02h			
03h	MSB ((S1:S2) - 18h)	MSB (S * NB)		
04h	LSB ((S1:S2) - 18h)	LSB (S * NB)		
05h	MSB Id (current file)			
06h	LSB Id (current file)			
07h	04h			
08h	ACr			
09h	ACw		ACi	
0Ah	ACu		ACd	
0Bh	KUC			
0Ch	01h			
0Dh	P3 - 0Dh			
0Eh	T (=0)	T (=1)	T (=3)	T (=7)
0Fh	00h	S1		
10h	00h		MSB ceiling	
11h	00h		Mid byte ceiling	High mid byte ceiling
12h	00h		LSB ceiling	Low mid byte ceiling
13h	00h		CHK	LSB ceiling
14h	00h		CP (=1)	
15h	00h		CP (=2)	
16h	00h		RFU (00h)	
17h	00h		RFU (00h)	

Table 10 : "GET RESPONSE" command

In case of an EP3 purse file, "CHK" is not checked by the card.
As many files as wanted can be created, with the only limit of EEPROM available space, but user must not forget that a file cannot be deleted, once successfully created.

Possible status words for "GET RESPONSE" are :

- (6x-00) if there is some wrong byte(s) in the ISO-command header.
- (9F-00) when the response has been correctly sent.

8.7. GIVE RANDOM :

Gives the card an 8-byte random value. for the next outgoing certified command.

Byte	Contents
Class	FAh
Command	86h
P1	not used
P2	not used
P3	08h
DATA	8-byte random number

Table 20 : "GIVE RANDOM" command.

This command must be run before any certified reading .

A random value is valid only for the next command.

Possible status words for "GIVE RANDOM" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 90-00 if the command's successfully completed.

8.8. INCREASE :

Allows to credit an electronic purse file :

Byte	Contents	
Class	FAh	
Command	32h	
P1	00h (not checked in not certified modes)	
P2	04h (not checked in not certified modes)	
P3	if plain mode : from 08h to 10h if ciphered *un*certified mode : 08h or 10h if certified (ciphered or not) mode : 18h or 20h	
DATA	*EP3*	*EP4*
	If certified mode : standard Block 0 (p. 22).	
01h-03h	00h	
04h	MSB date	00h
05h	LSB date	MSB amount
06h-08h	3-byte amount	3 low bytes of amount
09h-rec size	Free use additional byte(s) (0 or 8-byte if certified and/or ciphered mode)	
	If certified mode, 8-byte certificate (p. 21).	

Table 21 : "INCREASE" command.

To be allowed to perform an "INCREASE" command, access conditions specified in the current file header (CI and KUC bytes) must be fulfilled.

Before crediting the purse, the card checks if this will neither result in a balance greater than the ceiling, nor in a credit counter overflow (for EP4 only : no check is performed on EP3 counter).

Possible status words for "INCREASE" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 94-10 if the credit would result in a balance greater than the ceiling.
- or 94-20 if the credit would result in a credit counter overflow.
- or 96-x0 if mirror writing defects.
- or 65-01 if purse data updating defects.
- or 98-60 if no "ASK RANDOM" has been run prior to a certified credit.
- or 98-70 if something is wrong in the ciphering or in the certificate.
- or 98 80 if the access conditions aren't fulfilled.
- or 98 90 if the current file is not a purse file.
- or 90-00 when the credit has been done.

8.9. INVALIDATE :

Forbids definitively all the modifications in transparent and record files (writings and updatings), and in purse files (credit, debit and ceiling updating).

Byte	Contents
Class	FAh
Command	04h
P1	not used
P2	not used
P3	00h

Table 22 : "INVALIDATE" command.

Prior to this command execution, a valid presentation of the IK must have been performed. So a file invalidation may be done only under the application issuer control.

This command has as result to set ACw and ACr, or ACi and ACd to the value 0Ch, i.e. it sets the DIS bit, resets all other significants bits and set the 3-bit CHK field to 04h. No more access except readings can be further done on that file. Moreover, for a purse file, it disables the possibility to update the ceiling value.

Care must be taken, as an invalidated file can never be rehabilitated.

Possible status words for "INVALIDATE" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 96-x0 if mirror writing defects.
- or 65-01 if ACx writing defects.
- or 98-70 if the IK has not been previously successfully presented.
- or 90-00 if command's successfully completed.

8.10. READ BINARY :

Enables to read data from transparent files.

Byte	Contents
Class	FAh
Command	B0h
P1	MSB reading address inside file
P2	LSB reading address inside file
P3	01h to 0FFh if plain text mode. 08h to F8h modulo 8 if ciphered uncertified mode 18h to F8h modulo 8 if certified (ciphered or not) mode
	If certified mode, standard Block 0 (p. 22).
	Data read.
	If certified mode, 8-byte certificate (p. 21).

Table 23 : "READ BINARY" command.

Before reading a file, the access conditions specified in the current file header ACr and KUC bytes must be fulfilled.

If the reading must be done in certified mode, a "GIVE RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned

Possible status words for "READ BINARY" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- including 6B-00 if trying to read after the end of the file.
- or 98-60 if no "GIVE RANDOM" has been performed prior to a certified reading,
- or 98-80 if the access conditions are not fulfilled, or if the DIS bit in the ACr byte has been set at file creation.
- or 98-90 if the current file is not a transparent file.
- or 90-00 when the reading has been done.

8.11. READ RECORD :

Enables to read data from record and purse files.

Byte	Contents
Class	FAh
Command	B2h
P1	00h if reading of the current record (P2=04h), or XX for record number to be read (P2=04h), or no meaning if P2 byte is different from 04h
P2	04h for a direct access reading (with record number in P1), or 02h to read the next record (sequential way), or 03h to read the previous record (sequential way)
P3	01h to 5 if plain mode, 08h or 10h if ciphered *un*certified mode 18h or 20h if certified (ciphered or not) mode
DATA	If certified mode, standard Block 0 (p. 22).
	Data read.
	If certified mode, 8-byte certificate (p. 21).

Table 24 : "READ RECORD" command.

Before reading a file, the access conditions specified in the current file header ACr and KUC bytes must be fulfilled.

If the reading must be done in certified and/or ciphered mode, a "GIVE RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned.

Only one record may be read at once.

The records are numbered from 1 to NB.
When the last record has been read, reading the next record will return the record number 1. In the same way, when the first record has been read, reading the previous record will return the record number NB.

For the sequential readings, a current pointer exists in RAM. On record files, after the file selection, and before the first "READ RECORD", it points to a non existent pseudo record, located before the record number 1, and after the record number NB. So a reading of the current record (P1=0, P2=4) cannot be done until a next or previous record reading has been performed, else it results in an inexistent record access, with the status word 6B 00.

However, on purses files, this current pointer must not be mistaken with the file current pointer written in the file header, used to point to last credited or debited record ; to differentiate them, we'll temporarily call the RAM pointer RCP: "read current pointer" . After the purse file selection, and before the first "READ RECORD", RCP is set to CP value ; then RCP is managed the same way as for a classical record file, but every "INCREASE" and "DECREASE" command increments CP in the file header, then copies it back to RCP.

This means, in particular, that after each transaction (debit or credit), the reading of the current record (P1=00, P2=04) gives the last modified record contents, which contains the purse balance.

Only sequential readings, writings or updatings affect this current pointer, i.e. "READ/WRITE/UPDATE RECORD" commands with P2=2 or P2=3 and "INCREASE" and "DECREASE" commands on purse files.

Possible status words for "READ RECORD" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header,
- including 6B-00 if one wants to read an inexistant record (P1 greater than NB), or to read (record files only) the current record , without having previously read the next or the previous record,
- or 98-60 if no "GIVE RANDOM" has been performed prior to a ciphered/certified reading,
- or 98-80 if the access conditions are not fulfilled, or if The DIS bit in the ACr byte has been set at file creation,
- or 98-90 if the current file is a transparent file,
- or 90-00 when the reading has been done.

8.13. UPDATE BINARY :

Enables to update the data into a transparent file.

Byte	Contents
Class	FAh
Command	D6h
P1	MSB updating address inside file
P2	LSB updating address inside file
P3	01h to 10h if plain mode. 08h to 10h modulo 8 if ciphered *un*certified mode 18h to 20h modulo 8 if certified (ciphered or not) mode
DATA	If certified mode, standard Block 0 (p. 22).
	Data to update : 8-byte or 16-byte block if certified or ciphered mode).
	If certified mode, 8-byte certificate (p. 21).

Table 26 : "UPDATE BINARY" command.

Before updating a file, the access conditions specified in the current file header ACu and KUC bytes must be fulfilled.

Updating data first erases old data if necessary, then writes the new value(s).

Due to the card internal limitations (mirror area size), at most 16 bytes can be updated at once.

If the updating must be done in certified and/or ciphered mode, an "ASK RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned.

Possible status words for "UPDATE BINARY" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- including 6B-00 if one tries to update bytes after the end of the file.
- or 96-x0 if mirror writing defects.
- or 65-01 if data writing defects.
- or 98-60 if no "ASK RANDOM" has been run prior to a certified updating.
- or 98 70 if the signature is incorrect.
- or 98-80 if the access conditions are not fulfilled, or if the DIS bit in the ACu byte is set.
- or 98-90 if the current file is not a transparent one.
- or 90-00 when the updating has been done.

8.14. UPDATE CEILING :

Enables to update ceiling value in purse files header.

Byte	Contents	
Class	FAh	
Command	D6h	
P1	FFh	
P2	FFh	
P3	18h (ciphered and certified mode)	
DATA	*EP4 purse file*	*EP3 purse file*
01h..08h	Standard Block 0 (p. 22).	
09h..0Ch	4-byte new ceiling	3-byte new ceiling followed by CHK.
0Dh..10h	00h (padding)	
11h..18h	8-byte certificate (p. 21).	

Table 27 : "UPDATE CEILING" command.

Before updating a purse file ceiling, the CCK ("Change Ceiling Key") must have been successfully presented (with "EXTERNAL AUTHENTICATION" command).

Updating data first erases old data if necessary, then writes new value(s).

Due to the mandatory ciphered and certified mode of this command, an "ASK RANDOM" command must have been performed before it, else a 98 60 error will be returned. The CCK ("Change Ceiling Key") is the only key that can be used for ciphering and certifying with this command.

If the new ceiling value is lower than the current balance, the command will be rejected with the 94 10 status word.

Possible status words for "UPDATE CEILING" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 96-x0 if mirror writing defects.
- or 65-01 if new ceiling writing defects.
- or 94-10 if the ceiling updating would result in a ceiling lower than the balance.
- or 98-60 if no "ASK RANDOM" has been run prior to a ceiling updating (ciphered and certified mode only).
- or 98-70 if something goes wrong in the ciphering or the certificate, or if CCK has not been previously presented.
- or 98-80 if the DIS bit in the ACu byte is set.
- or 98-90 if the current file is not a purse file.
- or 90-00 when the ceiling updating has been done.

8.15. UPDATE RECORD :

Can be used to update data into the record files.

Byte	Contents
Class	FAh
Command	DCh
P1	00h if updating of the current record (P2=04h), or XX for record number to be updated (P2=04h), or no meaning if P2 is different from 04h
P2	04h for a direct access updating (with record number in P1), or 02h to update the next record (sequential way), or 03h to update the previous record (sequential way)
P3	01h to S if plain text mode, 08h or 10h if ciphered *un*certified mode 18h or 20h if certified (ciphered or not) mode
DATA	If certified mode: standard Block 0 (p. 22).
	Data to update (8-byte or 16-byte block if certified or ciphered mode).
	If certified mode: 8-byte certificate (p. 21).

Table 28 : "UPDATE RECORD" command.

Before updating a file, the access conditions specified in the current file header ACu and KUC bytes must be fulfilled.

Updating data first erases old data if necessary, then writes new value(s).

Only one record can be updated at once.

If the updating must be done in certified and/or ciphered mode, an "ASK RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned

The records are numbered from 1 to NB.

When the last record has been updated, updating the next record will modify the record number 1. In the same way, when the first record has been updated, updating the previous record will modify the record number NB.

For sequential updatings, a current pointer exists in RAM, which points initially (after file selection) to a non-existent pseudo-record, located before the record number 1, and after the record number NB. So an updating of the current record (P1=0, P2=4)just after the file selection cannot be done until a next or previous record updating has been performed, else it results in an inexistent record access, with status word 6B 00.
Only sequential readings, writings or updatings affect this current pointer, i.e. "READ/WRITE/UPDATE RECORD" commands with P2=2 or P2=3.

Possible status words for "UPDATE RECORD" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header,
- including 6B-00 if trying to update an inexistant record (P1 greater than NB),
- or 96-x0 if mirror writing defects,
- or 65-01 if data writing defects,
- or 98-60 if no "ASK RANDOM" has been run prior to a ciphered/certified updating,
- or 98-70 if something goes wrong in the ciphering or the certificate,
- or 98-80 if the access conditions are not fulfilled, or if the DIS bit in the ACu byte is set,
- or 98-90 if the current file is not a record file,
- or 90-00 when the updating has been done.

8.16. VERIFY PIN :

Enables to present any key, in plain-text way.

Byte	Contents
Class	FAh
Command	20h
P1	Not used
P2	Not used
P3	08h
DATA	8-byte PIN value.

Table 29 : "VERIFY PIN" command.

An EEPROM mapped registration counter increments on every bad (active or passive) presentation up to 3. When it's reached this value, the card gets in a locked-PIN state, where all commands expecting a good PIN presentation are forbidden, till an UK presentation followed by a good PIN presentation are successfully performed.

If the plain PIN value should not transit between the card and the terminal, the "VERIFY PIN" could advantageously be replaced by an "EXTERNAL AUTHENTICATION" command, which exploits a random value ciphered by the PIN ; to get more details, see "EXTERNAL AUTHENTICATION" description (p. 33).

Possible status words for "VERIFY PIN" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- or 96-x0 if mirror writing defects.
- or 65-01 if ratification counter writing defects.
- or 98-10, 98 20 or 98 30 for respectively first, second and third or more bad PIN presentation -active ("EXTERNAL AUTHENTICATION" on PIN) and passive presentations merged-.
- or 90-00 when the PIN verification has succeeded.

8.17. WRITE BINARY :

Enables to write data in transparent files.

Byte	Contents
Class	FAh
Command	D0h
P1	MSB writing address inside file
P2	LSB writing address inside file
P3	01h to 10h if plain mode. 08h to 10h modulo 8 if ciphered *un*certified mode. 18h to 20h modulo 8 if certified (ciphered or not) mode
DATA	If certified mode, standard Block 0 (p. 22).
	Data to write (8-byte or 16-byte block if certified or ciphered mode).
	If certified mode, 8-byte certificate (p. 21).

Table 30 : "WRITE BINARY" command

Before writing a file, the access conditions specified in the current file header ACw and KUC bytes must be fulfilled.

Writing data writes the new value(s), without previous erasing of the old data, e.g. if an 8-byte area had been set to hex 00 00 AA AA 55 55 FF FF, and if the string 0A 5F 0A 5F 0A 5F 0A 5F is written, the result string in that area is 0A 5F AA FF 5F 5F FF FF.

Due to card internal limitations (mirror area size), at most 16 bytes can be written at once.

If the writing must be done in certified and/or ciphered mode, an "ASK RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned.

Possible status words for "WRITE BINARY" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header.
- including 6B-00 if trying to write after the end of the file.
- or 96-x0 if mirror writing defects.
- or 65-01 if true data EEPROM writing defects.
- or 98-60 if no "ASK RANDOM" has been run prior to a certified writing.
- or 98-70 if something goes wrong in the ciphering or the certificate.
- or 98-80 if the access conditions are not fulfilled, or if the DIS bit in the ACu byte is set.
- or 98-90 if the current file is not a transparent file.
- or 90-00 when the writing has been done.

8.18. WRITE RECORD :

Enables to write data in record files.

Byte	Contents
Class	FAh
Command	D2h
P1	00h if writing of the current record (P2=04h). or XX for record number to be written (P2=04h). or no meaning if P2 is different from 04h
P2	04h for a direct access writing (with record number in P1). or 02h to write the next record (sequential way). or 03h to write the previous record (sequential way)
P3	01h to 5 if plain mode. 08h or 10h if ciphered *un*certified mode 18h or 20h if certified (ciphered or not) mode
DATA	If certified mode, standard Block 0 (p. 22).
	Data to write (8-byte or 16-byte block if certified or ciphered mode).
	If certified mode, 8-byte certificate (p. 21).

Table 31 : "WRITE RECORD" command.

Before writing a file, the access conditions specified in the current file header ACw and KUC bytes must be fulfilled.

Writing data writes new value(s), without previous erasing of old data, e.g. : an 8-byte record had been set to hex 00 00 AA AA 55 55 FF FF, and if the string 0A 5F 0A 5F 0A 5F 0A 5F is written, the result string in that record is 0A 5F AA FF 5F 5F FF FF.

Only one record can be written at once.

If the writing must be done in certified mode, an "ASK RANDOM" command must have been performed prior to this command, else a 98 60 error will be returned

The records are numbered from 1 to NB.

For sequential writings, a current pointer exists in RAM which points initially (after file selection) to a non-existent pseudo record, located before the record number 1, and after the record number NB. So a writing of the current record (P1=0, P2=4) just after the file selection cannot be done until a next or previous record writing has been performed, else it results in an inexistent record access, with status word 6B 00.
Only sequential readings, writings or updatings affect this current pointer, i.e. "READ/WRITE/UPDATE RECORD" commands with P2=2 or P2=3 .

When the last record has been written, updating the next record will modify the record number 1. In the same way, when the first record has been updated, updating the previous record will modify the record number NB.

Possible status words for "WRITE RECORD" are :

- 6x-00 if there is some wrong byte(s) in the ISO-command header,
- including 6B-00 if trying to write an inexistant record (P1 greater than NB),
- or 96-x0 if mirror writing defects,
- or 65-01 if true data EEPROM writing defects,
- or 98-60 if no "ASK RANDOM" has been run prior to a ciphered/certified writing,
- or 98-70 if something goes wrong in the ciphering or the certificate,
- or 98-80 if the access conditions are not fulfilled, or if the DIS bit in the ACu byte is set,
- or 98-90 if the current file is not a record file,
- or 90-00 when the writing has been done.

9. PERFORMANCES :

Here's a global overview of execution times for main commands, in many cases (different data lengths, different modes).

Times are in milliseconds, and include the transmission delays between cpad and card reader at 9600 bauds.

These times are for global appreciation only, they cannot be guaranteed, as they depend on the EEPROM file structure.

Due to the 16-byte page EEPROM erasing/writing internal algorithm, a more than one byte access to EEPROM can has important execution time variations, according to its address : if stored in only one page, it'll be faster than if in two pages. For example, samples obtained for 16-byte plain-text "WRITE BINARY" command vary from 67.28ms to 80.46 ms (414 samples, average value is 76.18, and σ is 2.85 (σ : sample standard deviation)).

Another phenomenon can influence a bit more the variation of these results : it is due to erasing automatic suppression when the new data do no imply an old data erasure. This last characteristic is increased by the use of an EEPROM mirroring mechanism.

The times have been evaluated from mean values, with from 50 (some data lengths for "READ BINARY") to more than 10.000 samples ("ASK RANDOM" has been sampled more than 284.000 times), according to type of command, with a samples mean number near to 750. Sample standard deviations vary from 0.005 ("ASK RANDOM") up to 13. So high values apply for commands erasing/writing EEPROM, when the paging mechanism appears (see the paragraph above).

Command	Data length	Plain text	Ciphered	Certified	Ciph. Certif.
ASK RANDOM	8	21.47	dna[1]	dna	dna
CHANGE KEY	8	70.41	87.00	dna	165.44
CREATE FILE	16 / 24	89.90 / 100.32	dna	dna	dna
CREATE FILE	20 / 28	97.91 / 108.35	dna	dna	dna
DECREASE(EP3)	8 / 7<n<16	102.78/87.007+n*1.966	122.43 / dna	162.89 / dna	199.26 / dna
DECREASE(EP3)	16	118.46	155.39	195.83	250.80
DECREASE(EP4)	8 / 7<n<16	102.99/86.480+n*2.058	122.71 / dna	163.12 / dna	199.48 / dna
DECREASE(EP4)	16	119.40	155.49	195.82	250.96
EXTERNAL AUTH	8	dna	39.63	dna	dna
GET RESPONSE	0<n<24	10.771+n*1.362	dna	dna	dna
GIVE RANDOM	8	20.99	dna	dna	dna
INCREASE(EP3)	8 / 7<n<16	99.62/84.116+n*1.867	118.98 / dna	159.32 / dna	195.51 / dna
INCREASE(EP3)	16	114.00	151.19	192.08	246.52
INCREASE(EP4)	8 / 7<n<16	100.74/86.442+n*1.699	118.96 / dna	159.07 / dna	195.53 / dna
INCREASE(EP4)	16	113.62	151.10	191.31	246.51
INVALIDATE	0	51.76	dna	dna	dna
READ BINARY	n[2]	10.683+n*1.362	10.775+n*3.673	51.262+n*3.677	69.909+n*5.990
READ RECORD	0<n<17	10.720+n*1.362	dna	dna	dna
READ RECORD	8	21.62	40.20	80.72	117.94
READ RECORD	16	32.52	69.58	110.14	165.84
SELECT FILE	0	9.33	dna	dna	dna
SELECT FILE	12	13.28	dna	dna	dna
UPDATE BINARY	0<n<9	48.494+n*1.911	dna	115.763+n*0.905	dna
UPDATE BINARY	8<n<17	48.494+n*1.911	dna	146.569+n*0.733	dna
UPDATE BINARY	8	64.22	87.12	123.15	160.66
UPDATE BINARY	16	76.35	120.00	158.32	214.66
UPDATE RECORD	0<n<17	48.760+n*1.986	dna	dna	dna
UPDATE RECORD	8	65.02	83.21	126.09	156.91
UPDATE RECORD	16	78.02	119.42	153.69	209.28
VERIFY PIN	8	97.48	dna	dna	dna
WRITE BINARY	0<n<9	46.470+n*1.875	dna	115.373+n*0.620	dna
WRITE BINARY	8<n<17	46.470+n*1.875	dna	144.134+n*0.641	dna
WRITE BINARY	8	61.75	79.96	127.5	164.78
WRITE BINARY	16	76.18	114.57	157.79	213.55
WRITE RECORD	0<n<17	49.173+n …397	dna	dna	dna
WRITE RECORD	8	60.38	79.99	124.61	155.66
WRITE RECORD	16	69.60	111.80	151.91	207.13

Table 32 : Global overview of average execution times (in ms).

[1] dna: does not apply.

[2] modulo 8 if not plain-text

10. POCKETBOOK CARD SAMPLES :

To avoid the realisation of existing PocketBook applications clones, samples smart cards that are delivered contain 5 files to fill completely the EEPROM area.

There is no way to create new files on such sample cards.

These 5 files are the following :

- The embedder file, with identifier 2Fh 00h as described in its chapter (p. 16) ; the system keys can be changed in plain mode, and there's no serial number ; this file is invalidated (ACr=05h : ACw=ACu=0Ch ; KUC=80h) . [24÷16 bytes].

Hex Offset:	Sample card embedder file body :			
0	FFh FBh		CHK	80h
4	FFh	A0h	FFh FFh	
8	FFh FFh		Manufacturing day	Manufacturing month
C	Manufacturing year	FFh FFh FFh		

Table 33 : Sample cards embedder file contents.

- An electronic purse file (EP4 type), called EFh 10h, with 11 records, and an 8-byte optional area per record. Access conditions are : certified reading (ACr=24h), certified and ciphered increase under DK0 control (ACi=72h), decrease under PIN control (ACd=84h). It is initialized with a ceiling of 100.000.000 units, and a balance of 29.710.000 units (decimal values), and its data keys can be changed in ciphered *and* certified mode (KUC=40h). [32÷176 bytes].

- Another electronic purse file (EP3 type), called EFh 01h, with 17 records, no optionnal area. Access conditions are set to : plain-text reading under PIN and DK1 control (ACr=C3h), certified increase under DK0 control (ACi=63h), and DK0-ciphered decrease under PIN control (ACd=93h). It is initialized with a ceiling of 1.000.000 units, and a balance of 297.100 units (decimal values), and its data keys can be changed in ciphered *uncertified* mode (KUC=10h). [32÷136 bytes].

- A record file with 9 15-byte records, called 80h 80h, reading under PIN control (ACr=84h), writing under DK0 control (ACw=44h) and updating under PIN and DK1 control (ACu=C3h), all in plain-text mode. Records are all zeroed, except first and last ones, and the data keys can be changed in ciphered *and* certified mode (KUC=41h). [24÷135 bytes].

- A transparent file that fills almost the whole remaining area, that is to say a 360-byte data area, called EFh 00h. Access conditions are DK1-ciphered reading under PIN control (ACr=93h), certified writing under PIN and DK0 control (ACw=D2h) and ciphered certified updating under PIN and DK1 control (ACu=E1h). Data field is zeroed, except 16 first and 16 last bytes, and the data keys can be changed in plain mode (KUC=91h). [24÷392 bytes].

- 2 bytes remain free, not usable at all since the minimum size to create a file (transparent or record without body) is 24 bytes.

This 5-file structure will allow to test almost all PocketBook capabilities and performances, with or without DES use ("EXTERNAL AUTHENTICATION", ciphering, certifying).

Keys are set to these values :

- PIN = 00 00 00 00 00 00 00 00
- UK = 00 00 00 00 00 00 00 00
- IK = 00 00 00 00 00 00 00 00
- DK0 = 11 22 33 44 55 66 77 88 (all files)
- DK1 = 01 23 45 67 89 AB CD EF (purse files)
- DK1 = FF 00 AA 55 F0 0F A5 5A (other files)

Purses counters, as well as life counter, contain non-zero values, due to the operations performed to create and test this structure.

11. Index :

กิตติกรรมประกาศ

ปริญญานิพนธ์ชิ้นนี้สามารถสำเร็จลุล่วงไปได้ ด้วยความช่วยเหลือจากหลายๆฝ่ายดังนี้

- คำปรึกษาและคำแนะนำจาก รศ.ดร. มนัส สังวรศิลป์ และอาจารย์ทุกท่าน
- ความช่วยเหลือด้านคำแนะนำและอุปกรณ์จากบริษัทINNOVATECH
- กำลังใจจากพ่อแม่ และเพื่อนๆทุกคน