NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF MAGNETIC STRIP/SMART CARD TECHNOLOGIES AND THEIR APPLICATIONS AT NPS

by

Rodolfo G. Dollete

September 1994

Thesis Co-Advisors: Timothy Shimeall
Roger Stemp

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

4. TITLE AND SUBTITLE
IMPLEMENTATION OF MAGNETIC STRIP/SMART CARD TECHNOLOGIES AND THEIR APPLICATIONS AT NPS(U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Dollete, Rodolfo Giron

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*

Various student management activities at the Naval Postgraduate School, Monterey, California are still using manual, labor-intensive, and inefficient methods. One such activity is the tracking of students attending the weekly Superintendent's Guest Lecture (SGL) series which requires a representative from each curriculum office to collect attendance cards for approximately 15 minutes and at least another three hours to sort out all the collected cards and generate the necessary reports. Is it possible to enhance student management activities, such as this, through implementation of Card Technologies?

In order to answer this and other related questions, two card technologies were investigated. These were the magnetic stripe and smart card technologies. Included in this thesis is an in-depth look at both of these technologies with emphasis on general technical characteristics, their strengths and weaknesses, and their current applications. In addition, three simple applications have been developed for NPS as "proofs of concept" of the potential benefits that the school might gain from utilizing these technologies.

We found that magnetic stripe/smart card technologies can enhance various student management functions at NPS. The result of an experimental application developed to keep track of students attending the Superintendent's Guest Lecture showed that an absentee report, an important management document that is non-existent under the current system, can be generated rapidly with significant cost savings in terms of manpower requirements. Other applications utilizing smart card technology such as military ID, credit/debit card, and student check in/check out have been developed and tested successfully in the computer laboratory.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Magnetic stripe, smart card, magnetic stripe application, smart card application.	163
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

IMPLEMENTATION OF MAGNETIC STRIP/SMART CARD TECHNOLOGIES AND THEIR APPLICATIONS AT NPS

by

Rodolfo G. Dollete
Lieutenant, United States Navy
B.S., Old Dominion University, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1994

Author: Rodolfo G. Dollete
Rodolfo G. Dollete

Approved By: Timothy J. Shimeall
Timothy Shimeall, Co-Advisor

R Stemp
Roger Stemp, Co-Advisor

Ted Lewis
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Various student management functions at the Naval Postgraduate School (NPS), Monterey, California are still using manual, labor-intensive, and inefficient methods. One such activity is the tracking of students attending the weekly Superintendent's Guest Lecture (SGL) series which requires a representative from each curriculum office to collect attendance cards for approximately 15 minutes and at least another three hours to sort out all the collected cards and generate the necessary reports. Is it possible to enhance student management functions, such as this, through implementation of Card Technologies?

In order to answer this and other related questions, two card technologies were investigated. These were the magnetic stripe and smart card technologies. Included in this thesis is an in-depth look at both of these technologies with emphasis on general technical characteristics, their strengths and weaknesses, and their current applications. In addition, three simple applications have been developed for NPS as "proofs of concept" of the potential benefits that the school might gain from utilizing these technologies.

We found that magnetic stripe/smart card technologies can enhance various student management functions at NPS. The result of an experimental application developed to keep track of students attending the Superintendent's Guest Lecture showed that an absentee report, an important management document that is non-existent under the current system, can be generated rapidly with significant cost savings in terms of manpower requirements. Other applications utilizing smart card technology such as military ID, credit/debit card, and student check in/check out have been developed and tested successfully in the computer laboratory.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The Naval Postgraduate School (NPS) at Monterey, California, like other colleges and universities in the country, has to deal with many student management activities such as vehicle registration, checking in and out with various departments, issuance of door access codes to restricted areas, and tracking of student attendance to the weekly Superintendent's Guest Lecture (SGL) series. These management activities involve many personnel, both civilian and military, which can be easily translated into man-hours and dollar costs that the school is ultimately responsible to pay. With the current personnel draw-down and decreasing fiscal budgets, there is an increasing need for all military installations to find ways and means to cut costs by eliminating unnecessary expenditures and making the necessary ones as efficient and effective as possible.

At NPS, the author proposes to cut costs by enhancing two different student management activities through the implementation of Magnetic Stripe and Smart Card Technologies. The first student management activity involves tracking of student attendance for the weekly SGL series. Under the current system, managing the student attendance for the SGL is a manual chore. Each student is issued a lined 3" x 5" card on which his/her name and curriculum number are handwritten to identify the card's owner. This card serves two purposes.

First, it is used for keeping records of dates of all lectures that the card owner has attended. This is accomplished by recording the date of every lecture attended using a mechanical date stamp. Second, the card is also used to track students failing to attend lectures without proper authorization. This is accomplished by distributing the cards to their rightful owners in the morning of the day of the scheduled lecture, and then the cards are individually collected by a representative from each curriculum office at the entrance to the King Hall Auditorium. The collected cards represent those who attended the lecture on that specific day. The cards are then individually date stamped and compared against a

master list in order to produce a list of absentees. The Curricular Officer is then notified which students were not in attendance.

The whole manual process takes up a total of approximately 3.5 hours to complete. At $20.00 per hour, this can be translated to an approximate expenditure of $33,600.00 a year ($20 x 3.5 hrs x 4 weeks x 10 curric. offices x 12 months) for the entire school. Furthermore, another factor that is difficult to attach a dollar amount to but nonetheless a disadvantage of the current system is the fact that since the whole process is usually interrupted by other more important activities during the course of the day, the reports are normally not ready until after a day or more after the lecture.

The second student management activity is the Check-in/out process. This process, which normally takes approximately 5 working days to complete, requires each student reporting to NPS to provide redundant basic information such as name, rank, social security number, etc. to various departments listed on the check-in sheet. A typical scenario involves a student filling out one or more forms before the department representative puts his/her initials on the check-in sheet. The data on the forms are then either filed away or entered manually into some computer filing system such as a database. Since each department has its own unique forms to fill out, the possibility of errors, or recording of inconsistent information can occur when the student fills out the forms or during the manual transfer of information from the forms into the computer. It is obvious that the current system has three major deficiencies. First, the current system takes too much time to complete. Second, the probability of having inconsistent student data held by different departments is high. Third, the Admin Officer has to rely on the scribbled initials on the check-in sheet as proof of check-in/out completion, because he has no way of verifying whether or not a student has, in fact, appeared in person with the various departments listed on the check-in sheet without calling each one of them.

B. OBJECTIVE OF THESIS

The objective of this thesis is to develop a potential solution to the current inefficient, and labor-intensive processes involved in tracking students attending the

weekly Superintendent's Guest Lecture series at NPS using the magnetic stripe technology. In addition, a potential solution to enhance the check-in process will be developed using the smart card technology. The smart card solution will also include an electronic purse application to show that, with smart card technology, different applications can be integrated into one smart card chip.

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

The scope of this thesis is limited to developing and testing magnetic stripe and smart card applications using computer hardwares and softwares, manufactured by Datacard Corporation, that are currently available at NPS computer laboratory. It is assumed that the same applications may be developed using similar products manufactured by other vendors when available. Furthermore, the data to be used for the magnetic stripe application program will only comprise of student information from Code 32 (approximately 300 students). It is assumed that if the application program can successfully handle 300 students, it can successfully handle the entire student body at NPS.

D. ORGANIZATION

This thesis is organized into six chapters. This chapter provides the introduction and scope of work to be performed. Chapter II covers an in-depth technical review of the Magnetic Stripe and Smart Card card technologies, and their advantages and disadvantages. Chapter III introduces the current and potential applications of Smart Card technology. Chapter IV and V cover the specific implementations of the magnetic stripe and smart card technologies at NPS. Finally, Chapter VI provides the conclusion of the thesis and suggestions for future work.

II. TYPES OF CARD TECHNOLOGIES

There are two major types of card technologies that are currently available in the market today. These are the **magnetic stripe** and the **smart card** technologies. The history, physical properties, differences, and applications for these technologies are the topics of discussion in this chapter.

A. MAGNETIC STRIPE CARDS

Magnetic Stripe Cards are the plastic cards that just about everyone nowadays carries around in his or her wallet. They represent the least expensive, least data storage, most mature, and most widely used card technology with an approximate shipment of 1.1 billion cards between 1980 and 1990 [Svig87]. Although these cards are used for different functions such as calling cards, identification cards, automatic teller machine cards, or credit cards, they all have one thing in common; that is, each is manufactured using the same standards of layout and measurements. These standards are promulgated by the International Organization for Standards (ISO). The following is a brief history of how these plastic cards came about and how their use gained popular acceptance by the society.

The development of the magnetic stripe industry came about as a response to a growing need for a machine-readable, reliable, and rewrite-capable media. Its first use was seen on Commuter Rail Tickets in the early 1960's for automation of passenger processing and revenue control of mass transit systems. The magnetic stripe was first implemented on paper tickets before it evolved to a plasticized form. The first few user were the London Transit Authority, Illinois Central Railroad, and San Francisco Bay Area Rapid Transit (BART). The magnetic stripes of those days were mainly used to record the declining balance on the users' tickets. For example, a user may start with a $50.00 ticket which allows him/her to use the transit system. For each time that his/her ticket is inserted or

swiped to a magnetic card reader, the current balance on the ticket is reduced and recorded. This system was so effective that it became popular and quickly gained wide public acceptance [McGe92].

The first plastic cards that we are all too familiar with were first introduced by the Franklin National Bank in 1951 and then followed by other banks towards the end of the 1950's. At that time, there were no magnetic stripes yet on the cards. Instead, raised letters and numbers created by embossing machines were used to imprint the information from the card on a set of paper forms. These cards, also known as "credit cards" were issued by banks to eligible customers with the idea of providing convenience through shopping on credit by charging purchases they would pay for the following month, and a way to avoid embarrassment at being short of cash at eating establishments. This arrangement proved so successful that in 1958, American Express, which was best known for its traveler's checks, introduced the "American Express Card" to the market place. Other banks such as the Hilton Corporation joined the card business by its introduction of the "Carte Blanche" bank card in 1959. The bank's motivation was, and still is today, to gain additional revenue by charging a set percentage out of the customers' outstanding balance. As far as the banks were concerned, the larger the balance, the better it was for the business. [McGe92]

The appearance and application of the card remained basically unchanged until the 1970's when there was a need for a reliable and machine-readable media to support ease of operation of Automatic Teller Machines. The medium of choice was, of course, the magnetic stripe. So it was during this same period that the plastic cards were enhanced by adding magnetic stripes on the back of the cards. The motivation behind this enhancement was to allow a magnetic card reader inside the automatic teller machine to capture information about the customer from the magnetic stripe and preclude the manual entry of a 19-digit customer identification number to operate the ATM; an activity that was not only

cumbersome but also vulnerable to human errors. All the user had to do now is to enter a maximum of 5-digit personal identification number (PIN) to authenticate himself to the automatic teller machine that he is in fact the true owner of the card. Currently, most cards are still usable by both the magnetic card readers and the embossed-characters imprinting devices. [Svig87]

1. Physical Characteristics of Magnetic Stripe Cards

The physical characteristics of magnetic stripe cards are set by the International Standards Organization (ISO). These specific measurements which include length, width, thickness, location of magnetic stripe, and embossing requirements are illustrated in Figure 1 below.

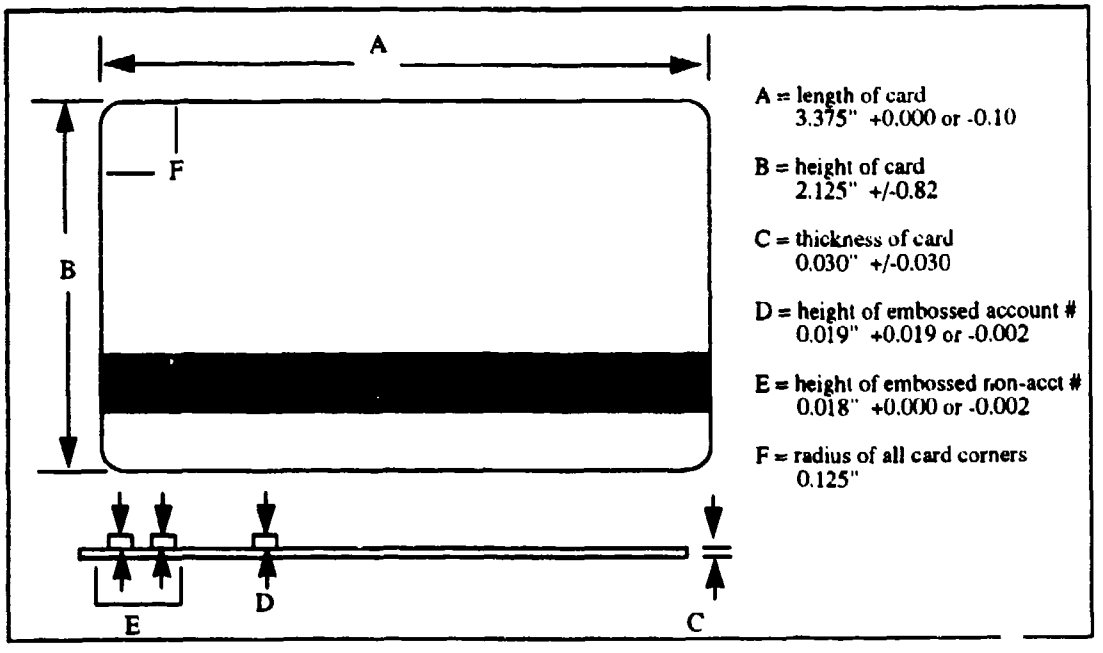

Figure 1. Card Dimensions in accordance with the ISO.

The magnetic stripe on each card has three tracks which are specifically reserved for different applications. These tracks occupy three physical locations on the magnetic stripe. Each track has a width of 110 thousandths of an inch on which data is to be encoded. The ISO standard numbers the tracks according to their proximity to the card's top edge, track

1 being the track nearest to the top. The three tracks with their specific functions are described below:

a. Track 1

Used for automating airline ticketing or other transactions where a database is accessed. This track is encoded with a density of 210 bits per inch in ALPHA format and may contain up to 79 alphanumeric characters, including the primary account number which could be up to 18 digits, name which could be up to 26 alphanumeric characters and additional data such as expiration data, restriction or type and discretionary data. Encoding specifications, information content and sequence for this track were developed by the International Air Transportation Association (IATA). [Svig87]

b. Track 2

Used for automating financial transactions. This track is usually referred to as the banking industry track. It is encoded at 75 bits per inch in BCD format and it may contain up to 40 alphanumeric characters, including the primary account number which could be up to 19 digits, and additional data such as expiration data, restriction type, and discretionary data. The primary account number identifies the issuing industry, the issuer, and the user identification number. Also included as an optional entry into track 2 is an offset or a PIN parameter which could be up to 5 digits. Track 2 is written before the cardholder receives the card. When the card is used with an on-line terminal it is interrogated and the information concerning the cardholder's account and identification are transferred to the card issuer's central computer for verification. [Svig87]

c. Track 3

Used for financial transactions. This track is encoded at 210 bits per inch in BCD format and it may contain up to 107 numeric characters, including primary account number of up to 19 digits; user and security data, such as country code, currency code, amount authorized per cycle and service restrictions; and additional data, including

subsidiary account data, and cryptographic check digits. This track is intended for use in applications where the magnetic data is changed and modified after most transactions. The third track was developed after the others and it is the only one that can be rewritten. It usually contains an encoded version of the chardholder's PIN. When the user enters his PIN, it is compared with the PIN encoded in the magnetic stripe without reference to a central computer. This eliminates the need to transmit data to a distant computer for confirmation before the transaction can proceed. [Svig87]

2. Encoding of Magnetic Stripes

In order to understand encoding and reading information on a magnetic stripe, one needs to have a basic knowledge of how a permanent bar magnet works. This section provides a review of the underlying principles of magnetism which is the foundation of all magnetic stripe recording. Figure 2 is an illustration of the two kinds of magnets, a permanent bar magnet and an electromagnet which is also sometimes referred to as a solenoid.

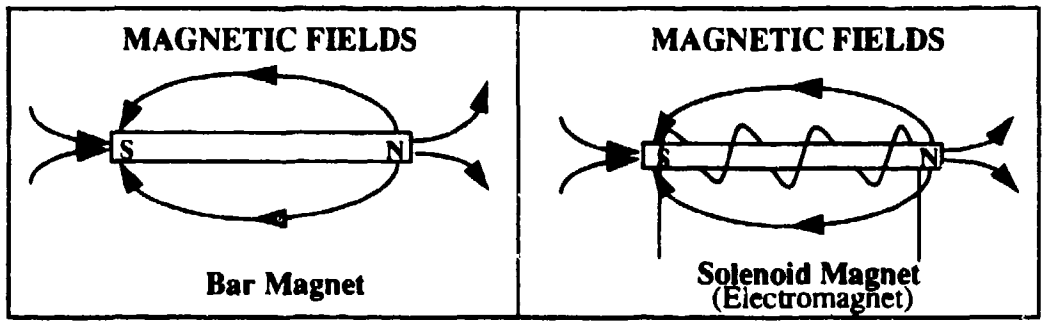

Figure 2. Two kinds of magnets: a bar magnet and an electromagnet.

Each of these magnets has a north pole and a south pole. The lines with arrows originating from the north pole to the south pole represent the magnetic fields. The magnetic field can also be interpreted as the magnetic lines of attraction or magnetic flux between the two poles. What is important to note in this illustration is the presence of a

higher concentration of magnetic fields lines at the poles than anywhere else, especially right along the middle section of the magnet. The two magnets shown are the same in all respects except for the requirement of electric current to maintain the magnetic fields of the electromagnet. In addition, the polarity of the electromagnet can be reversed, i.e. the south pole turning into a north pole and the north pole turning into a south pole, by simply changing the direction of the electric current. It is this special property that makes the solenoid or electromagnet more practical and convenient to use in producing tiny permanent bar magnets in the magnetic stripe. But how can we produce tiny bar magnets in the magnetic stripe?

Let's refer to the permanent bar magnet again in Figure 2. If this bar magnet is cut into two or more pieces, each piece will have its own north pole and south pole. And since opposite poles attract each other, these pieces can be stacked end to end with the north pole of one attached to the south pole of the other as illustrated in Figure 3 below.

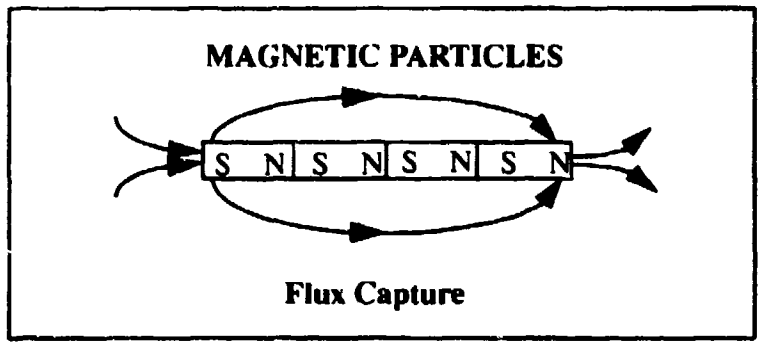

Figure 3. Tiny magnetic bars stacked end-to-end.

It is clear from the above illustration that a bar magnet is actually made up of a series of tiny magnets. When tiny bar magnets are stacked together as in Figure 3, all of the magnetic flux coming out of the north pole of one tiny magnet is captured by the adjacent south pole of the next tiny bar magnet. This **flux capture** results in a situation similar to Figure 2 where the only flux lines escaping the bar magnet are those coming from the end

poles. The construction of a magnetic stripe is similar to what has just been described; that is, stacking tiny bar magnets end-to-end. There are, however, two major differences. First, the tiny bar magnets on a magnetic stripe are much smaller -- 20 millionths of an inch long to be exact. These magnetic particles, sometimes called domains, are held together by a rigid binder to make them immovable. The second major difference is in the polarity arrangement of the particles. On the magnetic stripe, the magnetic particles are stacked and packaged together and then magnetized in such a way that the north pole of one particle is adjacent to the north pole of the next, and the south pole of the next is adjacent to the south pole of the particle following it. This arrangement is exactly the opposite of what we saw in Figure 3. The purpose of placing two similar poles adjacent to each other is to create a flux reversal which is obtained when two poles repel each other. As illustrated in Figure 4 below, the new arrangement results in a high concentration of magnetic flux lines at the points where two similar poles are stacked together. The magnetic flux reversal lines are generated in the magnetic stripe when it passes through the magnetic encoding head. [Byce65]

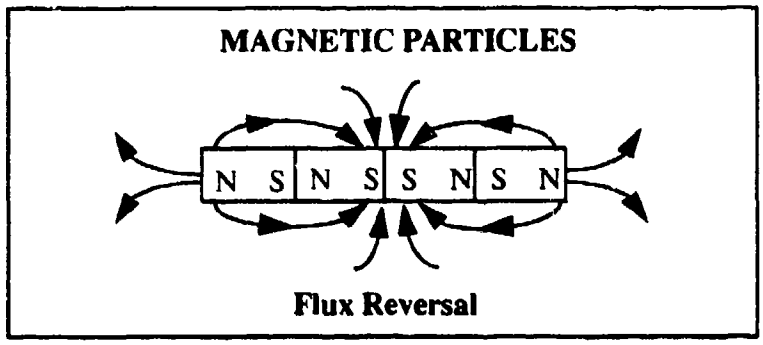

Figure 4. Magnetic Particles on a magnetic stripe.

The magnetic encoding head is nothing more than a solenoid magnet discussed earlier with its north and south poles facing each other to form a gap of about 1/2

thousandth of an inch wide. With this small gap, the magnetic field can be made strong enough so that when the magnetic stripe passes through it, as in Figure 5, it will magnetize the magnetic particles embedded in the stripe to form a series of tiny permanent bar magnets with the south pole on the left and the north pole on the right. The polarity of the magnetic particles is opposite that of the gap. To create the flux reversal necessary in encoding information in the magnetic stripe, all we had to do was change the direction of the current flow through the solenoid. The change in current flow will reverse the polarity of the magnetic gap resulting in a corresponding change in the polarity of the stripe's magnetic particles under the gap. [Byce65]

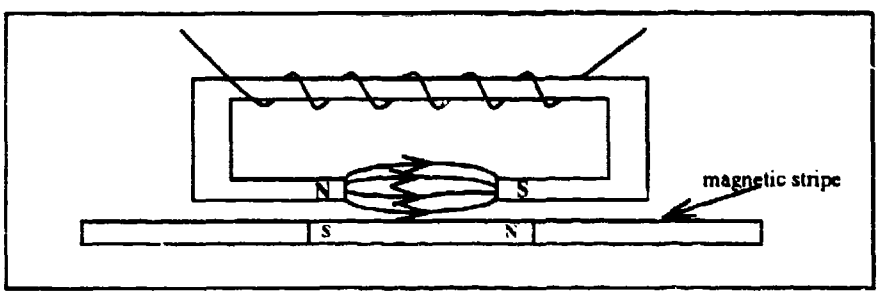

Figure 5. Encoding of Magnetic Stripe.

A snapshot of this technique taken right after the change in the direction of current flow is graphically shown below in Figure 6. This illustration shows the polarity of the magnetic particle to the left of the gap before the current was reversed, and the polarity of the trailing particle under the gap right after the change in the direction of current flow. Note that the particle to the left of the gap has a north-south polarity while the particle under the gap has a south-north polarity. Our interest, however, is not on the polarity of individual particles but on the resulting magnetic flux reversal created by two adjacent poles with the same polarity. In Figure 6, we have created a south-south flux reversal with one change in the direction of the current flowing through the solenoid. If we continue to change the

direction of current as the magnetic stripe moves to the left away from the gap, a north-north flux reversal will be created followed by another south-south flux reversal and so on. These magnetic flux reversals will play a major role in the process of decoding the encoded information in the magnetic stripe which is the topic of the next section.

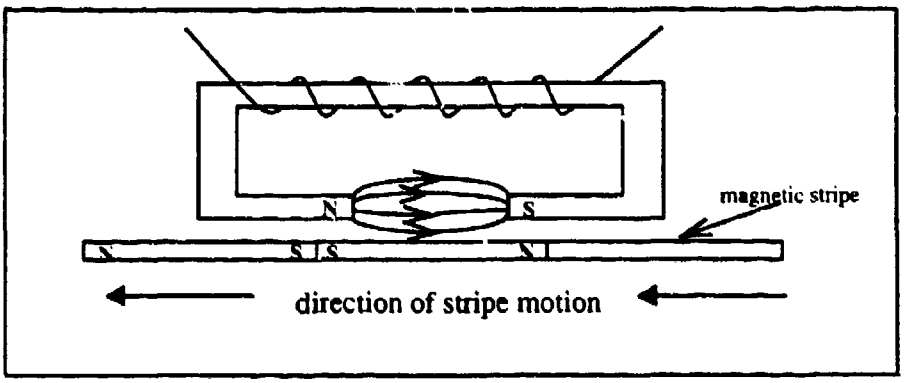

Figure 6. Encoding of Magnetic Stripe.

3. Decoding of Magnetic Stripe

Decoding or reading of information from a magnetic stripe is accomplished by passing the stripe under the decode head. The decode head is the same as the encode head discussed earlier without the current flow through the solenoid coils. As the magnetic stripe passes underneath the gap, the high concentration of magnetic flux lines radiating from the magnetic reversal points in the card will result in a rapid build up of magnetic field in the gap and produce current to flow through the solenoid coil. The magnetic field build up and current flow through the coil will reach their peak when the magnetic reversal points on the stripe are directly underneath the decoding head. However, both will drop suddenly as the reversal points slide away from the gap. This rapid fluctuation of magnetic field and current flow through the solenoid coil can be detected or recorded as a change in voltage across the coil. This voltage, which comes in the form of pulses, will be positive for reading north-north reversals, and negative for south-south reversals. To guarantee a build up of magnetic

field at the magnetic gap, it is important to generate a relative motion between the magnetic stripe and the magnetic gap. No motion will result in no change in magnetic field and therefore no current flow. The speed of relative motion must be no slower than 1/2 inch per second. Any speed below 1/2 inch per second will be interpreted as no motion at all. [Pear67] Figure 7 is an illustration of this technique.

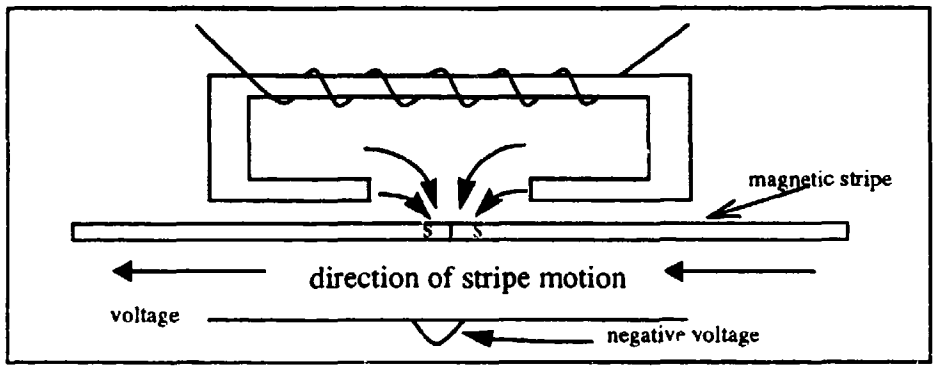

Figure 7. Decoding of Magnetic Stripe.

4. Encoding Technique

Now that we know how to create magnetic flux reversals in the magnetic stripe using the encode head, and how to detect those flux reversals with a decode head, the next step would be to show a technique on how to find the exact locations of the flux reversals so that they can be used to represent data. The technique is called Biphase Encoding. This technique takes advantage of the fact mostly known to everyone that any information can be represented by a string of zeros and ones. This allows us to pick a fixed spacing of flux reversals, called zero-bit spacing, the size of which depends on how many data bits we want per inch of stripe length. The zero-bit spacing is also called the bit-cell. The Biphase Encoding rule states that if the bit-cell does not have a flux reversal at its midpoint, then it represents a zero-bit; otherwise, it is a one-bit representation. It is now easy to see how we can encode any information we desire on the magnetic stripe using strings of zeros and ones. Figure 8 below is a graphical illustration of the biphase encoding technique. Note that the one-bit frequency is always twice that of the zero bit. This characteristic makes the

biphase technique self-clocking which means that decoding is independent of read speed. Finally, biphase is universal, since decoding the zeros and ones is completely independent of the data format used or the data content encoded on the magnetic stripe. In other words, knowledge of the data format used in encoding information in magnetic stripe is not required to decode the card. [Pear67]

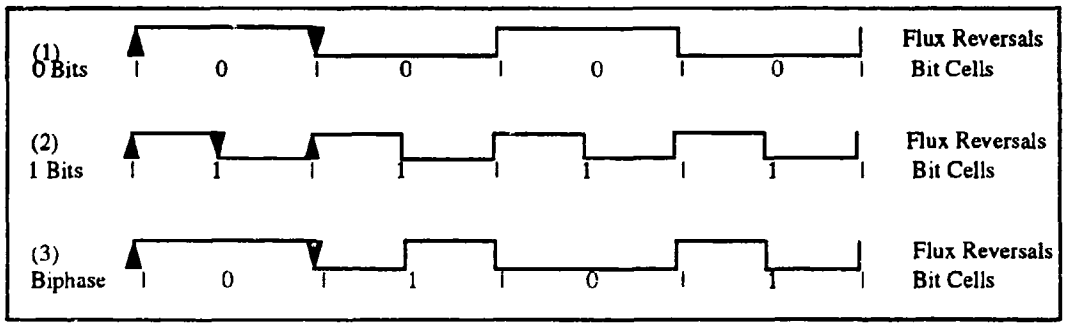

Figure 8. Biphase Encoding Technique.

5. ISO BCD Data Format

Decoding means to translate the biphase flux reversal on the magnetic stripe into a string of zeros and ones. If that's all we do, then the output of our reader is a string of data bits with a clocking strobe -- called Bit & Strobe or Data & Clock output. However, if this bit-pattern represents data, then the person who encoded the stripe had to follow a rule to determine the flux reversal pattern, therefore, we must know the rule to translate the bit-pattern back into the data.

The rule is called a data format, and the process of translating data into bits is called formatting, and translating bits into data is called reformatting. While the number of possible data formats is infinite, our discussion will focus on the ISO BCD format which is used on track #2 of all credit cards.

The ISO format has 16 characters in the set, including the digits 0 through 9 and some punctuation marks. The fundamental requirement of a data format is that every character in the set has a unique bit-pattern. For a16 character set, this requires 4 bits for

each character, labeled data bits. For example, the semicolon will be represented as 1-1-0-1 and the digit 9 will be represented as 1-0-0-1. The ISO BCD format adds a fifth bit for each character, called an odd parity bit, to check for read errors in the four data bits. This means that every character encoded in the BCD format is a string of 5 bits of zeros and ones on the magnetic stripe. [Pear67]

A second requirement for a data format is that you designate one character as the Start Sentinel and another character as the End Sentinel. The ISO format uses the semicolon and the question mark as its Start Sentinel and End Sentinel, respectively. The encoding of magnetic stripe requires that the Start Sentinel immediately precedes the data characters and the End Sentinel immediately follows the data characters. The ISO data format is illustrated in Table 1 below.

Data Bits				Parity	Character	Function
bit-1	bit-2	bit-3	bit-4	bit-5		
1	1	0	1	0	;	Start Sentinel (SS)
0	0	0	0	0	0	Data
1	0	0	1	1	9	Data
1	1	1	1	1	?	End Sentinel (ES)

Table 1: ANSI/ISO BCD Data Format.

The ISO also defines another data format, called the ALPHA format, which is a 7-bit set containing the alphabet as well as the numerals. It is used on track #1 of all credit cards, and follows the same configuration as the BCD format.

6. ISO Encoding Protocol

Now that we have a data format, and have used it to format our data into a string of zeros and ones, we need a protocol for where to put it on the magnetic stripe. Figure 9 below graphically illustrates these requirements.

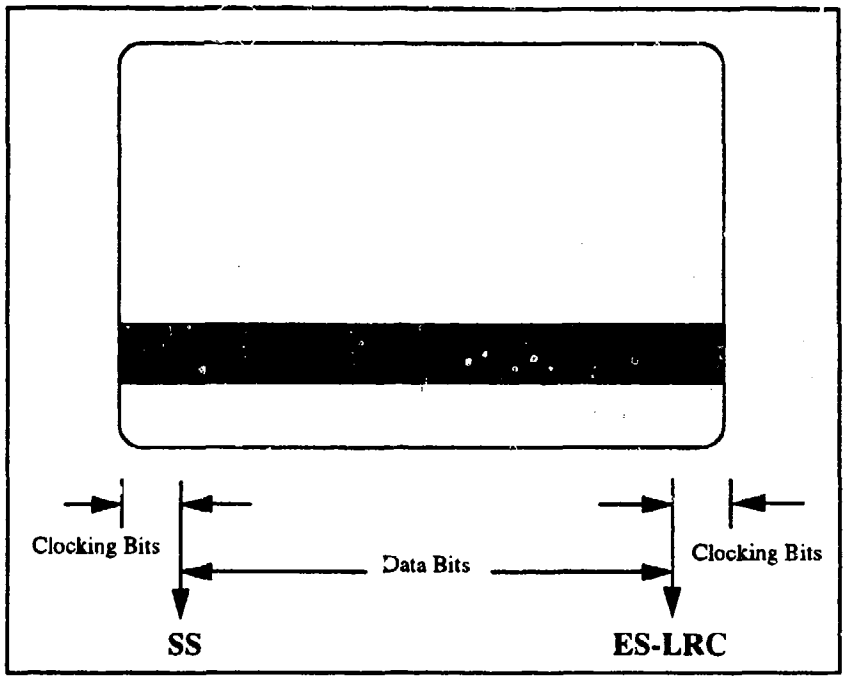

Figure 9. ANSI/ISO Encoding Protocol.

Notice that we must begin and end the encoding on the stripe with a series of clocking bits -- these are zero bits and not zero characters. This allows the reader decode circuit to synchronize on bit-cells that it knows are all zero-bits before it has to start checking for mid-cell reversals to decode one-bits. This is how the self-clocking feature of biphase lets you read your magnetic stripe cards at widely different speeds. You will also notice that the ISO protocol requires another additional character after the End Sentinel. This character is called Longitudinal Redundancy Check, or LRC character which acts as another parity check character to catch any read errors in the data. [Pear67]

7. Advantages of Magnetic Stripe Card Technology

The following is a list of the advantages of magnetic stripe card technology.

- **Machine-readable.** Data can be easily stored and read by machines that support magnetic stripe technology.

- **Reliability.** Although read reliability is only at 80 -95%, most problems associated with magnetic stripe card are due to magnetic damage and operator error in inserting the card into the card reader slot. Both problems can be easily corrected by using HiCo cards and employee training, respectively.
- **Re-write Capability.** Before smart card technology, magnetic stripe has a high data capacity and its re-write capability allows modification/update of the stored information.
- **Low Cost.** The magnetic stripes wide range of alternative substrate and media sizes allows the issuer to choose, based on the application, the most cost effective option.

B. SMART CARDS

Smart Cards are predicted to appear in mass quantities throughout the world and will affect many areas of the society. To anyone who is used to carrying a credit card or phone card, a smart card may appear no different. Like the popular credit card, this new card may even have embossed characters on the front and a magnetic stripe on the back. Its appearance, however, is quite deceptive for the smart card is certainly not just another financial card but a new type of portable computer. The main difference between the smart card and a magnetic stripe card is the presence of a microcomputer chip embedded in the card which provides it greater memory capacity and intelligence. In short, smart cards are devices that have both processing power and memory, and are capable of being packaged in the format laid down by the ISO.

The microcomputer chip used on smart cards is a product of gradual technological improvements of the first general purpose computer (ENIAC) developed in 1945 by John Mauchly and J. Presper Eckert at the University of Pennsylvania. The ENIAC occupied 1,800 square feet of floor space, had about 18,000 vacuum tubes, 70,000 resistors, 10,000 capacitors, 6,000 switches, and a power requirement of 180,000 watts. Furthermore, it weighed over 30 tons. [Acad94] To think that the smart card chip originated from this monstrous machine may be quite incredible to many people. However, a simple review of history will reveal many significant technological breakthroughs that brought about better

ideas in designs and development of smaller but more efficient computers. The miniaturization process can be seen in the production of mainframe computers, mini-computers, personal computers, and portable computers (lap tops). These machines have not only decreased in size and weight but the price have also become affordable for personal use. By the end of 1976, the computer industry had developed the microcomputer chip to a point where it was possible to have it embedded into a credit card-size plastic packaging. The technological advances that had a profound effect toward the development of the microcomputer chip are listed chronologically in the following table.

Date	Technological Breakthroughs
1945	The Electronic Numerical Integrator and Computer was developed by John Mauchly and J. Eckert.
1948	Invention of transistors by Walter H. Brattain and John Bardeen. Transistors are solid state devices made of semiconducting materials that can be used to control or amplify electrical current.
1950's	Invention of the magnetic-core memory.
1960's	Introduction of the Busicon Calculator using 3 chips.
1970	Dr. Kunitaka Arimura of Japan patented the idea for a plastic integrated circuit card. His patent, however, covered Japan only.
1971	A microprocessor, 4004, containing four chips was developed and distributed by Intel.
1974	A French journalist, Roland Moreno, patented an idea of putting a microcomputer chip inside a conventional plastic credit card.
1976	The first 8-bit single chip microcomputer was developed by Intel. The microprocessor, called 8048, consisted of a CPU, ROM, RAM, and Input/Output.

Table 2: Historical Evolution Of The Smart Card Chip.

1. Components of the Smart Card

A smart card has four fundamental components - a power source, memory, input/ output, and a microprocessor. Each of these components are discussed below.

a. Power Source

There are three methods currently in use in providing the necessary power to smart cards: The first method includes a thin battery embedded within the smart card package. Cards with this type of power source are usually thicker than the physical dimensions laid down by ISO standards. This type of power source is usually found in smart cards with RAM technology as the only memory medium. The continuous power supply provided by the built-in battery is required to maintain the contents of the RAM when the card is not physically in contact with a read/write unit.

The second method utilizes inductive coupling which allows both data and power to be transmitted from the read/write unit through the air waves or some non-metallic surface to the card. This power source is used primarily in providing power to contactless smart cards.

The third method is applicable to cards containing chips with metallic contacts on the surface. Power is transmitted to the designated power rails when the cards are inserted or swiped through a read/write unit. The exact dimension and location of the chip contacts are laid down by ISO 7816-2.

b. Memory

Memory is made up of an array of cells. The cells have two electrical charge states: either a charge is present or no charge at all. These electrical states represent the binary numbers 0 and 1. The instruction sets and data in memory are formed using the two binary digits. Typically, smart cards have memories ranging from 1K to 8K.

Smart card memories can be classified into three types: Read Only Memory (ROM), Random Access Memory (RAM), and Programmable Read Only Memory (EPROM). ROM is a non-volatile memory which means that the data is retained in memory

even when the power source is removed. The contents of ROM are embedded in the chip during the manufacturing stage and cannot be altered.

RAM, on the other hand, is a volatile memory which means that all data stored in memory will be lost upon removal of the power supply. The main advantage of RAM is that data can be stored to it, deleted from it, modified in any way, and read from it. In addition, data in RAM can be accessed in random order which makes it an excellent temporary storage area for smart cards.

The third memory classification, PROM, is a combination of non-volatile and reprogrammable memory. There are two types of PROM memory: electrically programmable read only memory (EPROM), and electrically erasable programmable read only memory (EEPROM). Memory cells in EPROM, which are initially set in the 1 state during manufacture, can be changed to 0 state through programming in order to suit the user's application requirements. Once programmed, however, the memory cells can no longer be programmed back to the 1 state without erasing the entire memory by exposing it to ultraviolet light. Despite this shortcoming, EPROM is still used in smart card applications that do not require memory locations to be altered more than once. The EEPROM is quite similar to EPROM with the exception of its added characteristic that allows it to be reprogrammed using electrical means. Erasing the entire memory contents by ultraviolet light is no longer necessary.

(1) A Typical Memory Organization. The four types of memories discussed earlier, ROM, RAM, EPROM, and EEPROM, can be organized in different ways within the smart card chip. For example, the 2K microprocessor smart card chip manufactured by Datacard Corporation is configured into three main areas.

The first area, the ROM area, holds the operating system for the smart card chip. The size of this area is 6K bytes. The second area, the RAM area, consists 160 bytes of memory space and is used by the operating system to process various smart card commands. As mentioned earlier, this area is cleared of its contents when power is

turned off. The third area, the EEPROM area, is where the data for the smart card application is stored. It consists of 2048 bytes.

The EEPROM area is further divided into two areas: a security area and a user area. The security area is used by the Smart Card Operating System (SCOS) to store keys and keep track of card status. This area takes up 128 bytes of the 2048-byte EEPROM space, and is not accessible to the user. The remaining EEPROM space (1920 bytes) is called the user area. [Data92] The memory organization of the 2K microprocessor smart card chip by DataCard Corporation is illustrated in Figure 10 below.

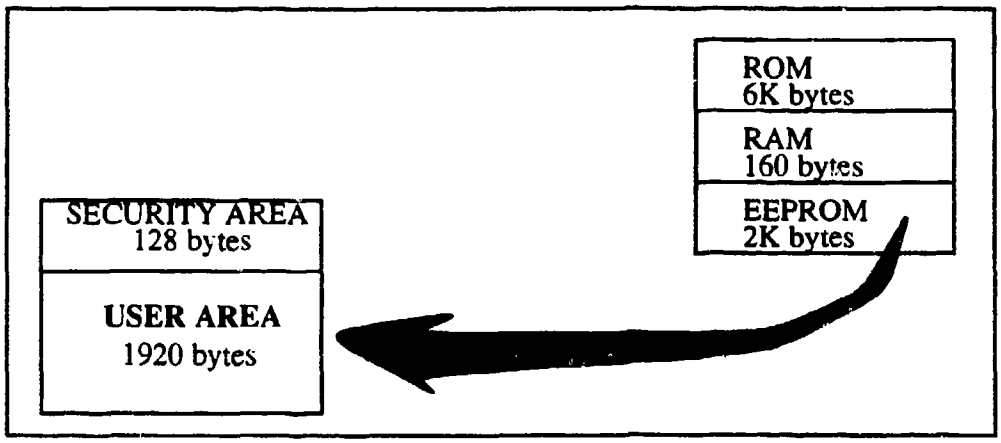

Figure 10. 2K Smart Card Memory Organization.

The user area is organized in a manner that is closely similar to that of a formatted floppy disk. It has a directory area, called the file definition table, and an area for the actual file data. As illustrated in Figure 11, each file has a file definition that is 16 bytes long occupying the lowest available logical memory location, and the actual file data occupying the highest available logical memory location. For example, the file definition of the first file will occupy the first 16 bytes of the user area (0 - 15). The actual data for this file definition, assuming the data is 60 bytes long, will occupy the last 60 bytes of the user area (1860 - 1919). [Data92] The next file definition will occupy memory location 16 through 31, and its data will start in memory location 1800 through 1859, assuming it is

also 60 bytes long. In general, the beginning address of the file data can be determined by the following formula:

$$BA = (HAM - FS) + 1$$

where

BA - Beginning Address
HAM - Highest Available Memory
FS - File Size

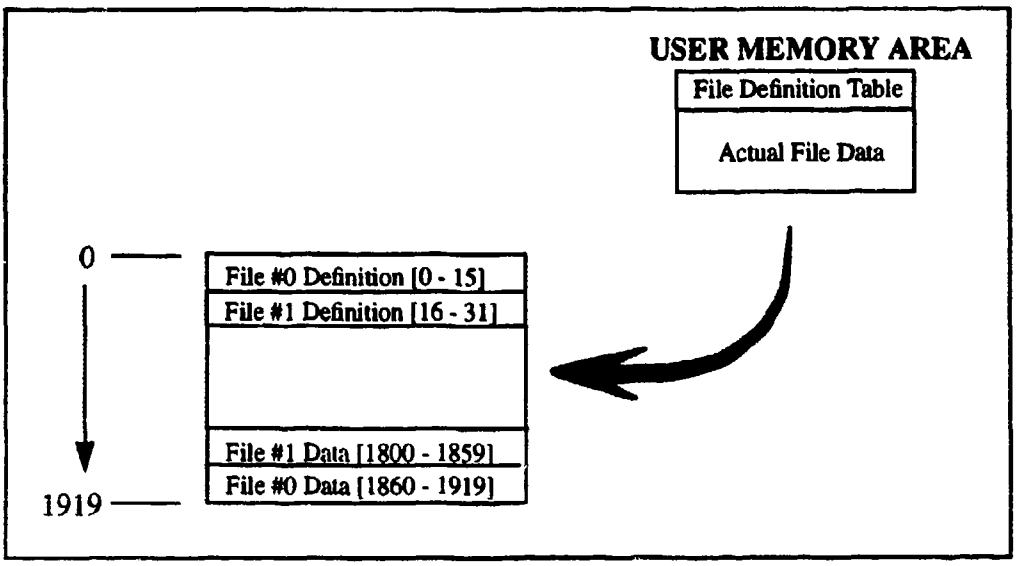

Figure 11. Organization of User Memory Area.

c. Input/Output

To be useful, the smart card must be able to communicate with the real world. This requires a means of receiving and transferring data from its memory locations to a read/write unit. Various input/output methods exist for different types of card. For contact smart cards, the metallic contact terminals on the surface of the cards are the link by which data is transferred and received between the smart cards and the read/write unit. For contactless smart cards, communication between the cards and the read/write unit is accomplished by a process called frequency modulation or frequency shift keying. This process involves two coils of wire, one acting as primary coil and the other as the secondary coil. When an alternating current is passed through the primary coil, a magnetic field will

be created. When the secondary coil is brought in close proximity to the primary coil the alternating magnetic field induces a flow of current in the secondary coil. Data is communicated by using alternating current with two different frequencies, representing the binary digits 0 and 1. [Pemb94]

d. Microprocessor

The microprocessor is the intelligent element of the smart card. Its two basic functions are manipulation and interpretation of data. These functions are carried out by executing the instruction set stored in the card's memory. The classification of microprocessors are based on the number of bits which a microprocessor is capable of manipulating simultaneously. For example, a four-bit microprocessor is capable of performing calculations on binary numbers with four bits, an eight-bit microprocessor is capable of performing calculations on binary numbers with eight bits, and so on. Most smart cards have eight-bit microprocessors.

2. Classification of Smart Cards

Different manufacturers usually produce different smart cards. They can vary in terms of the microprocessor used, the type of memory, and the way they communicate with the read/write unit. Despite these differences, however, smart cards can be broadly divided into three categories: contact smart cards, contactless smart cards, and super smart cards. These categories are discussed in detail below.

a. Contact Smart Cards

Smart cards with surface contacts were the initial industry standards. These smart cards have the microelectronics embedded in the card with connections to metallic contact pads on the surface of the card. The contacts are used to provide electrical power to the smart card chip and the means by which the read/write unit and the card's microcomputer communicate. The International Organization for Standards dictates that there should be 8 contacts on the front or back of the card. The issuer has the option of

selecting either side. As illustrated in Figure 12 below, two of the contacts are reserved for supply voltage and ground, one for reset and one for the clock signal which provides timing information for the microprocessor to carry out its operational sequences. The remaining contacts provide for serial data input and output and power supply necessary to program the memory. Two contacts are reserved for future designation. Since the contacts are exposed, it is not unusual for these cards to have problems of contamination during distribution or normal use, resulting in bad connection and sometimes complete card failure. [Svig87]

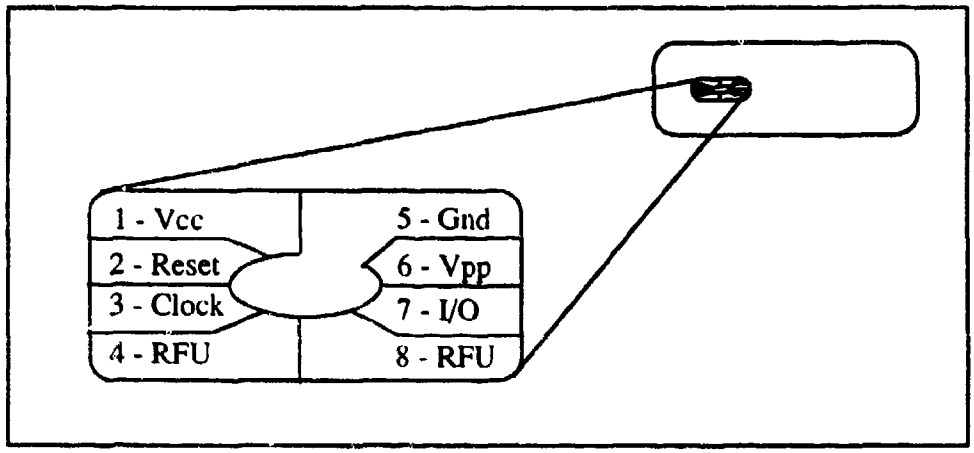

Figure 12. Smart Card Chip Contacts Based On ISO 7816-2.

b. Contactless Smart Cards

This type of smart card can perform all of the functions provided by a contact smart card. The only differences are the method of data communication and provision of power supply. The contactless smart card does not have external contacts similar to a contact smart card discussed earlier. Both data and power are inductively transferred using a medium frequency which corresponds to the CPU clock frequency [Stan93]. These differences are the major factors which make the contactless smart card more attractive than the contact smart card for many applications.

(1) Advantages of Contactless Smart Cards. The following are some of the advantages of contactless smart cards over contact smart cards:

- **Reliability.** In any electrical system, contacts are almost invariably the points where failures occur. A contactless interface makes the card intrinsically more reliable than cards with surface contacts which are susceptible to damage, contamination and wear.
- **Longer life.** For the reason mentioned above, a contactless smart card has a potentially longer life expectancy compared with contact smart cards.
- **Facility.** In use, the contactless card can be placed in any orientation on the read/write unit, even upside down. It is easier and faster to use than a contact smart card which has to be placed in a slot.
- **Convenience.** The read/write unit can be mounted under or behind any non-metallic working surface. This is very useful in banks and at retailing check-outs, keeping counters free of equipment.
- **Minimal Maintenance.** Unlike many conventional card reading mechanisms the read/write units have no moving mechanical parts.
- **Vandalism-proof.** In most cases the read/write units have no slots that in conventional readers are often subject to vandalism; for example, by putting glue or chewing gum in the slot.
- **Robustness.** Read/write units can withstand harsh environments. They can be made available as fully-sealed electronic units. The contactless card and read/write unit are most suitable for working in industrial or other harsh environments where either may become covered with oil, water, grease, or dirt. These conditions could cause major problems for cards with exposed surface contacts and read/write units with slots. [Hono94]

(2) Disadvantages of Contactless Smart Cards. Despite the many advantages of contactless smart cards, progress and market acceptability is restrained by a lack of standards for communication. In fact, with public domain projects being implemented with different standards, such as highway tolls, many smart card manufacturers have been forced in a "waiting" state as they do not know which types of circuits to design for the market. [Watt92]

c. Super Smart Cards

The super smart card is the third category of smart card. It differs from the other smart cards quite radically insofar as it incorporates a keyboard and liquid crystal display (LCD). It can function more like a stand-alone terminal without the need of a card read/write unit. Although it can work independently, there are instances when it needs to

go on-line to computer. To accommodate this function, these cards also have surface contacts. To maintain downward compatibility with magnetic stripe readers, the card is able to electronically emulate a magnetic stripe being read. As with other cards, a logo and an embossed cardholder's name and number can be incorporated. In the case of the super smart card, on the side of the card, opposite the display and keyboard.

Some of the disadvantages of super smart card include its higher price compared with other smart cards, difficulty in meeting ISO measurement standards, and the restricted size of the keyboard which makes it slow and difficult to use.

3. Advantages of Smart Cards

Smart cards have many advantages over the traditional magnetic stripe cards. The following are just a few of these advantages:

a. Greater Storage Capacity

The capability of smart cards to store data is substantially greater than magnetic stripe cards. A two kilobyte microprocessor smart card, for instance, can store up to 1,920 characters of data. The amount of data storage on a magnetic card with three tracks, on the other hand, can only store approximately 198 characters. The type of application and cost are usually the two most important considerations in making decisions on the selection of memory size. The obvious advantage of a larger memory is its flexibility to accommodate expansion of functionality at a later date. [Data92]

b. Multiple Applications

Due to its greater capacity to hold more data, it is possible to consolidate several card-related applications in one card without losing control and management of each application. For example, the applications could be as diverse as health care information, electronic benefits, credit card accounts, school records, security access, pre-paid cards, and many others. Another advantage of the smart card technology is its capability to be modified for addition or deletion of various applications. The number of

different applications that can be implemented using one card is only limited by the size of the smart card's on-board memory. Currently most smart cards memory ranges from 2k bytes to 8k bytes. [Data92]

c. Anti-Fraud Capability

In a credit or debit card application, the smart card technology can prevent overspending by its owners because, unlike magnetic stripe cards, the current balance and credit limit are stored on the card. The card is immediately updated during each transaction to reflect the new balance which forces the card holder to spend within his/her limit at all times. [Data92] The implementation of Electronic Purse in Chapter V covers this capability in more detail.

d. Off-Line Transaction Processing

One of the weaknesses of the magnetic stripe card technology is the long delay for verification and authentication of the validity of the user's card. The delay, which can be significant during peak hours of operation, can be attributed to the necessary connection to a central computer that holds the information necessary for verification and authentication. The smart card technology, however, carries all the verification and authentication data right on the chip. For example, an off-line smart card reader can access the current balance of the customer's account, subtract the cost of current transaction, and record the new balance in the card. This capability results in the customer's account balance being accurate all the time whenever and wherever he/she conducts business using the smart card. All transactions are captured and recorded by the smart card readers for periodic update of the central master files in the central computer during non-peak operational hours. [Data92]

e. Reconfigurable

New cards need not be re-issued whenever there is an upgrade to the application or enhancement of the user's privileges. These changes can simply be uploaded

to the card the next time it is used with access to the host processor. This flexibility provides the customers with prompt, inexpensive and uninterrupted service. [Data92]

f. Redundant Audit Trail

Off-line transactions using magnetic stripe technology does not provide any kind of audit trail at all. Due to its limited space, only the current balance is recorded on the card with no record of past transactions. This presents great difficulty in recovering lost or stolen cards since the current balance is not recorded anywhere else. With smart cards, however, complaints from customers are easily handled with the help of redundant records of all transactions. These records are available in three different places: in the smart card's memory, in the smart card reader, and finally in the central computer. These redundancies are believed to have contributed to the substantial decline in the attempt of malicious users to alter the information in the card or try to use the card in a fraudulent manner. [Data92] An example of audit trail implementation using smart card is covered in Chapter V of this thesis.

III. APPLICATIONS OF SMART CARDS

Smart cards are now being used in every field of social, educational, and business life. Both users and service providers have something to gain in using smart cards. To the users, the cards are convenient to use, practical, and secure. The service providers find that the card's security and built-in logic generate direct savings from phone calls to check authenticity, and indirect savings from losses due to fraud. Some of the major applications of smart cards are discussed below.

A. FINANCIAL APPLICATIONS

One of the major applications of smart card is in the world of finance, which includes banking, wholesale, insurance, and retail businesses. Although some of these applications can be combined in one card, each is discussed separately in this section.

1. Credit and Debit Cards

The security features of the smart card make it ideal for use as a credit or debit card because of the protection it gives against fraud. It is estimated that fraud and bad debt losses on conventional magnetic stripe credit cards were around $2 billion a year for one major card issuer alone. Fraud losses are usually due to stolen cards, altering and cloning cards, and customers not paying their debts. [Svig87] With smart cards, the losses may be reduced by including a built in credit limit into the card's memory. This technique will prevent the cardholder from exceeding the limit and run up large bills which he is later unable to pay. The credit limit could be changed and reset at a bank when the bill is paid.

As part of the card's security feature, the following information can be held in the smart card's memory when used in a financial application: identification of the card issuer, cardholder's identification, PIN, account balance, transaction limit, and a log of transactions.

2. Electronic Checks

The smart card can be used in Electronic Fund Transfer at the Point-of-Sale (EFTPOS). At the retailer's check-out counter, the card can be placed on or in the reader unit where it automatically goes through authentication sequences. To authorize payment, the customer must type in his PIN which the card then matches with the one held in its memory. If the number corresponds, the card will authorize the terminal to transfer funds from the cardholder's bank account to the retailer's bank account. This can be done instantaneously, or at a pre-set time, or the transaction can be stored in the point-of-sale terminal until the terminal is interrogated by the bank's computer. A record of the transaction will be stored in the card at the same time.

The card cannot give the cardholder information about the balance in his bank account, because of other transactions such as direct debits, but it will allow transactions up to a certain limit set by the bank, perhaps on a monthly basis. This use of the smart card will bring about a significant decrease in the number of checks and vouchers handled by financial institutions and should, therefore, reduce the costs of paper processing. On-line communications costs will also be reduced as the card's built-in authorization means that referral to the bank's main computer is not required for every transaction. They can be batched up for transfer in the middle of the night. The ability of the smart card to contain a pre-set credit limit will reduce losses due to bad debt and will enable the financial institutions to issue cards to a greater number of people. This will, in turn, benefit the retailer as there will be more customers with credit.

3. Electronic Tokens

The smart card has considerable potential as an electronic token. The idea here is that a prepaid area is set aside to store electronic units of time or electronic tickets, etc., for a specific service or item. Magnetic stripe cards are currently being used with public telephones, parking meters, and vending machines for this type of application. These

tokens have declining balance and are discarded when empty. Using a smart card, however, could result in longer life since they are rechargeable and could hold several token areas.

As an example, the smart card could be used to pay for gas and parking as a replacement for cash and coins. Consumers could pay for units at a vending machine at a gas station or at the parking lot management office, which would credit their cards. The cards would then be used to operate the gas pumps and parking meters. The advantages of this system would be that collections would no longer have to be made and there would be no more incentive for people to rob meters or collectors.

4. Electronic Purse

The electronic purse is a smart card containing electronic cash which can be used as a substitute for coins and bank notes. This scheme solves the problems associated with the use of cash such as handling costs and security risks. The smart card is loaded at ATM's and can be used for low-value purchases in shops and vending machines for products such as bread, newspapers, groceries, car parking and others, without necessarily requiring the authorization of a PIN. When the card is credited with the cash, the cardholder's account is debited in the normal way, except that the amount debited is then held by the bank until reclaimed by a retailer. When the card is used to purchase goods or services, the value of the transaction is deducted from the card and held securely at the retailer's terminal. The retailer then presents this information to the bank so that his account can be credited. In the intervening time the bank can invest the money. For the retailer and his customers, the electronic purse is a secure, convenient, and fast method of payment.

Electronic purse transactions do not usually require the use of a PIN which speeds up the transaction but makes the purse as vulnerable to theft as conventional cash. The amounts involved, however, are usually only small. Wide spread adoption of electronic purse will reduce the costs to banks and retailers of handling large quantities of cash. An example of a smart card Electronic Purse implementation is covered in Chapter V.

5. Government Benefits

The smart card would be ideal as a means for payment of government benefits such as welfare checks, social security allowances, and pensions. Using smart cards for these applications would alleviate much of the paper processing required under the current system. Currently, several states have been given approval by the Federal Government to develop and implement payment of welfare benefits using the Electronic Benefit Transfer (EBT) system. The importance of this topic is evidenced by Vice President Gore's plan to have an EBT system in place or under development in every state by 1996, and a legislation introduced in Congress calling for the elimination of paper Food Stamps by 1997. [Mill94] In Wyoming, for example, the smart card technology is currently being tested to handle food stamp payments to 2,100 households and 1,700 Women, Infants, and Children (WIC) participants. The smart card's multiple application feature is being tested in this pilot project since some of the WIC participants are also entitled to food stamps, and both benefits are contained in one card. [Will94] Wyoming's pilot project is being closely watched as it is the first state to test the concept of combining several welfare benefit programs on a single smart card which is the key factor in justifying the extra cost of smart card technology.

B. MEDICAL APPLICATIONS

Of all human activities, modern health care system is perhaps the most decentralized, specialized, and information-intensive. This means that the availability of better information at the right place and time translates to better health care and lower costs. The smart card technology has all the necessary attributes to meet the need for efficient, portable, and secure information. Listed below are just a few of the many potential medical applications of smart card.

1. General Medical Card

The smart card's large memory capacity (compared with magnetic stripe), could be used to record the following information about the cardholder:

- Holder's address, date of birth, and next of kin
- Name and address of doctor
- Recent medical history
- Serious complaints
- Allergies
- Drugs being taken
- Donor wishes

The card could be carried by the individual and in the event of an emergency, such as the holder collapsing in the street, it could provide immediate information to an ambulance crew or doctor. The speed of the availability of vital information could save many lives. The cost to a health organization of implementing such a scheme could be relatively small as many people would see the card as a form of insurance and be prepared to pay for it themselves.

The smart card could hold information concerning the patient's recent medical history, say, for the previous six months. In many cases, this would be all the information a doctor would require. If the patient carried the card to his appointment it would not be necessary for the doctor to have the patient's file in front of him. At the end of the consultation the doctor's notes could be recorded on the card and a copy printed out for the file. At present, on occasions, a great amount of effort has to be put into locating a patient's records and getting them to the right place at the right time. If each patient was issued with a smart card the paper records could be kept for reference only and for use if a card is lost.

A smart card medical record could prove useful to doctors and home health care agencies caring for patients discharged early from hospitals. At present, it can be several days before a doctor or nurse receives the patient's medical records and this can cause delay in giving the patient correct medical treatment. If the patient's medical records could be accessed by health care professionals, from a smart card through a portable card reader, administrative procedures could be improved.

2. Drug Issue Card

This smart card could be most useful to patients receiving regular medication. Some patients, for instance, take several different drugs at a time. At present the patient has to fill in forms and get the doctor's authorization for each repeat prescription. This is time consuming and wasteful. The drug issue card could contain the amounts to be prescribed and the frequency with which they should be dispensed. It would facilitate accurate control over the drugs issued to an individual, ensuring that no more than a certain amount was prescribed in a given period. In particular, the card could be useful in controlling the issue of dangerous drugs.

A drug issue card could replace written prescriptions if doctors were required to record the prescriptions in the card instead of on paper. This would have the advantage of eliminating the risk of the pharmacist misinterpreting the doctor's handwriting and could also save time for the pharmacist if the care were able to initiate automatic label printing.

3. Staff Identifier

The smart card is an ideal means by which access to designated areas in a hospital can be restricted to authorized staff personnel only. For example, the smart card could be used as a key to access areas where controlled drugs, medical records, and dangerous machines are kept. In addition, the smart card can be used as a time card for all hospital employees.

C. SECURITY APPLICATIONS

Smart cards can serve as electronic keys to high security areas such military instaliations, and computer installations of major banks. Smart cards are excellent for this type of application because of its capability to carry multiple security levels. In addition, smart cards can carry biometric information such as digitized fingerprints, photographs, signatures, and retinal scans. All this information can be embedded in the smart card securely which makes it very difficult to duplicate or modify. The following are some of the popular security applications of smart cards.

1. Access Control Card

An important concern in all types of establishment is access control. The smart card can provide secure access control to buildings, computer rooms, and other restricted areas. Authorized personnel will be issued with cards specifically programmed to allow them access to areas necessary for the performance of their jobs. As an added feature, the card can be programmed to maintain a log of audit trail which includes all the cardholder's movements through the restricted areas. The log will include the date and time of both successful and failed attempts to gain entry.

2. Personal Identification

Linking an identification card to its rightful owner is a major security concern. Identification of people used to be an easy task; but the complexities brought about by modern society have made this process complicated and expensive. We all have read front-page news such as: the welfare recipient signing up for benefits under six identities, the child being released to a stranger from a day care center, the hacker accessing sensitive databases, the counterfeiter making duplicates of bank cards, etc. These are just a few examples of stories resulting from improper identification. The demand for a better security system has never been greater.

With the magnetic stripe card, the usual method of identifying the rightful owner is the PIN. First, the identity number stored on the card is read by the system and the PIN is generated from it using an algorithm. This PIN is compared with the PIN provided by the cardholder. If both PINs are the same, it is assumed that the person presenting the card is the rightful owner of the card.

With a smart card, the process is more complex than that of the magstripe but more secure. An identification system using smart cards offer the following advantages:

• The computing power of the smart card allows it to carry out the PIN generation and comparison without giving any identity number to the system. This capability avoids the security weakness of the magstripe technology.

• Encryption and decryption of PIN can be handled by the cards microprocessor. This prevents compromise of identification numbers through electronic tapping or eavesdropping while the card is on-line.

• The cardholder can change his PIN any number of times to one that he can easily remember. The smart card may be programmed to refuse to accept PINs that are too obvious and easy to break.

The use of PINs in conjunction with either magstripe or smart card, however, only proves that someone knows the key to the system. It does not necessarily mean that the person using the card is the authorized cardholder. An alternative, using biometrics stored in the smart card's memory, is the most secure form of identification that is currently being introduced in the market. This method involves measurement of a unique personal characteristic of the cardholder, digitization of the measured characteristic, and recording of the digitized image in the card's memory. Some of the characteristics being considered are discussed below.

a. Fingerprint

The fingerprint is well established for its stability and uniqueness. It is estimated, based on a century of examination, that the chance of two people, including twins, having the same print is less than one in a billion [Mill94]. This unique characteristic makes fingerprint an excellent addition to the security features of a smart card. The whole fingerprint image, which includes arches, loops and whorl, and the tiny points called "minutiae", can be scanned and stored in the card's memory. Verification of cardholder's identity will include reading of the user's fingerprint by a scanning device, and comparing the result with the fingerprint image stored in the card.

b. Eye Patterns

The blood vessel pattern on the retina of the human eye provides a unique physical characteristic that can be used for positive identification. The process involves scanning the retina by directing a low-intensity infra-red light through the pupil to the back

of the eye. The system forms a profile of the subject for comparison by locating and recording the nodes and branches of the blood vessel patterns. Currently, Eyedentify, Inc., is the only company producing retinal scan products.

Another company, called IriScan, is currently working on a device that examines the pattern of flecks on the human iris. The advantage of this new technique over the retinal scan is its ability to capture the iris image from several feet away [Mill94].

c. Hand Scans

Like the fingerprints and eye patterns, the human hand also has some unique features such as the lengths of fingers and the radius of the curvature of fingertips. This technique of identification is sometimes referred to as the granddaddy of biometrics due to its 20-year history of live applications. The process involves positioning the hand on a photo-electric scanning device which takes the measurements and compares them with previously stored data. [Mill94] As in previously discussed biometrics, the smart card could be used to store the cardholder's hand scan characteristics.

d. Signature Dynamics

Verification of signature as a means of identifying the rightful owner of a card is usually done by visual inspection at the point of sale. There are occasions, however, when the signature is not checked at all and, even if it is, the examination is usually only cursory. With automatic verification using signature dynamics, the whole process becomes more consistent, thorough, and accurate. The key in signature dynamics is to differentiate between the parts of the signature that are habitual and those that vary with almost every signing. The verification process involves the user signing his signature with a wired pen on a sensitive writing surface. The interaction between the pen and the writing surface, which includes pressure and drag, is recorded and then compared with previously stored data. [Mill94]

3. Voice Verification

Voice verification is one of the most popular biometric approaches in the market. Its application can be seen in Sprint's Voice Activated Foncard television commercial. This technique of identification involves programming a computer to compare measured features of the user's voice with a previously recorded original. The original recording is the result of having the user speak various words through a microphone, and the necessary features for identification are then analyzed, digitized, and recorded. To prove positive identification using this method, the system will ask the user to speak a word or words in order to obtain the necessary sample for comparison with the original voice. [Mill94]

4. Communication

Another security feature offered by the use of smart card relates to communications. With the exception of smart cards used solely for identification, all smart cards must be able to communicate with another device. The method of communication may be through a read/write unit connected to a point-of-sale terminal, or a computer connected through telephone lines. The device that the smart card communicates with is really not important. The main concern is the security of the data being transferred to and from the smart card.

As with other secure means of communication, there are four basic requirements that must be attained in order to ensure security of messages that are transmitted between a smart card and another device. These requirements are discussed below.

a. Integrity

Integrity means that the smart card and another device must be able to detect whether the message that has been sent between them has been modified deliberately or accidentally. The integrity of transmitted data can be guarded by using a checksum method. This method involves adding up all the bits that make up the whole message and the sum is appended at the end of the message being transmitted. The receiving device can carry out the same addition of message bits and compare the result with the transmitted checksum.

If the computed checksum matches the transmitted value, it is very likely that the message has not been modified, accidentally or otherwise. [Pfle89]

b. Validity

Validity refers to unauthorized recording of genuine messages that are played back at a later time or date for the purpose of committing fraudulent transactions. For example, if a genuine message crediting a person's account with $100 is intercepted and recorded by an impostor, it could be continually replayed with the effect of adding $100 to that person's account each time.

There are various ways of providing validity, but each method ensure that, somehow, each message sent is made unique. This means that any subsequent identical message can be regarded to be fraudulent. The uniqueness can be obtained by using date and time stamps, or a message number which increments with each message sent. [Pfle89]

c. Authenticity

Authenticity refers to the ability of the card and another device to determine that messages are being received from a genuine sender and sent to a genuine receiver. To ensure authenticity of messages, the smart card and the receiving device must have an identical authentication algorithm and secret keys. The card first generates a random number, R_1, and sends it to the receiving device. Both the receiver and the smart card process the random number using the authentication algorithm and secret key. The receiving device will transmit the result of its computation, [A(K, R1), R2], along with a new random number, R_2, back to the smart card which compares it with the result of its own computation. If the two match, the smart card knows that the receiving device is authentic. To prove its authenticity to the receiving device, the smart card will generate a response using the random number, R2, the secret key, and the authentication algorithm. The smart card will transmit the response, [A(K, R2)], back to the receiving device which compares it with its own computation. If the two match, the smart card is also authentic. [Pfle89] This simple challenge and reply scenario is illustrated in Figure 13 below.

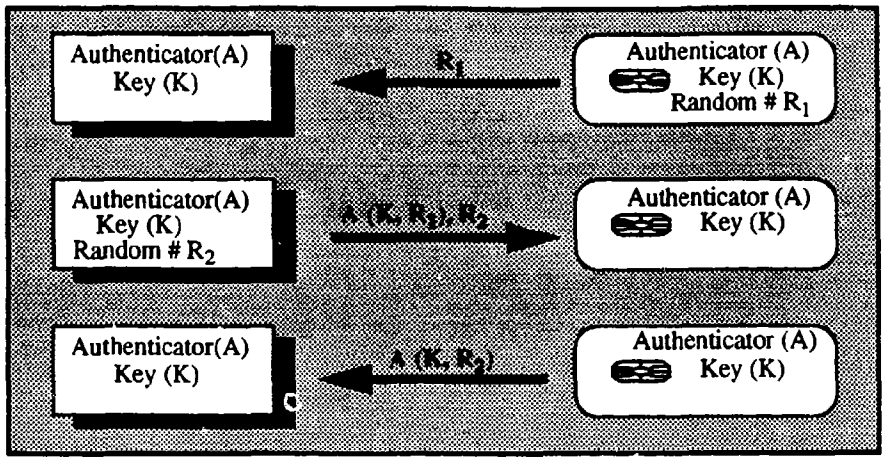

Figure 13. Authenticity Determination Between A Smart Card And An External Device.

d. Privacy

Privacy refers to the ability to prevent unauthorized persons from reading the messages that are being transmitted. Privacy of message transmission may be attained through encryption. Encryption is defined as a process of encoding a message so that the meaning of the message is not obvious to unauthorized receivers. Various encryption algorithms exist today, but the three most popular of them are: Merkle-Hellman knapsack, Rivest-Shamir-Adelman (RSA), and Data Encryption Standard (DES). Details on these topics can be found in [Pfle89].

5. Military Applications

In the military, where security is vital, various functions using the smart card can be implemented. The following are just a few of these applications:

a. Personnel Access Control

The use of smart cards can contribute to the military's requirement for controlling access to restricted and classified areas, and definitely in controlling access to classified information. Currently, the most common devices used to control access of

personnel to restricted areas where sensitive or classified data is held are keys, combination to cipher locks, badges, and magnetic cards. All these have the disadvantage of being duplicated, and if stolen or passed on, they can allow entry by an unauthorized personnel.

As discussed in the previous section, a smart card overcomes the weaknesses mentioned above by its ability to store digitized personal characteristics which are difficult, if not impossible, to duplicate. The card can be programmed so that the cardholder is allowed access only to specific areas based on a "need to know" policy and the level of his security clearance. In addition, the smart card can provide an audit trail of all the cardholder's movements through the security system.

b. Portable Data File

The military is currently testing a new Identification card for all service personnel. This new ID is called MARC which stands for Multi-Technology Automated Reader Card. It is a smart card that carries a bar code, magnetic stripe, embossed data, printed information, and an Integrated Circuit (IC) chip. The combination of several media on one credit card-sized device gives the MARC ID card its versatility: it can interface with a variety of technologies and systems from the old embossing machines to smart card readers/writers attached to a computer system. Some of the applications that are currently being tested using the MARC card include: Composite Health Care System (CHCS) Patient Reception, Field Medical Documentation, Food Service Head Count, Readiness Processing, Accountability, and Manifesting. [Ayre94] With this technology, a service member could carry in his wallet information concerning his service records, financial records, medical records, privileges, and others. The obvious advantage of this scheme is the availability of up-to-date information wherever the service member goes.

c. Equipment Maintenance Record

Military equipment of high cost value such as forklifts, cranes, generators, boats, etc., require service documentations which must be regularly updated to reflect

current certifications and maintenance actions. The smart card's memory could be used to hold all these information and can be attached to its associated equipment for easy access.

IV. IMPLEMENTATION OF MAGNETIC STRIPE TECHNOLOGY

This chapter covers the implementation of magnetic stripe technology at NPS to enhance the tracking of attendance of students required to attend the weekly Superintendent's Guest Lecture. The chapter will include discussions of the current system's weaknesses, proposed solution using magnetic stripe technology, and implementation details.

A. WEAKNESSES OF THE CURRENT SYSTEM

The current SGL attendance tracking system discussed in Chapter I has been in place for more than 30 years and its performance, in general, has been considered adequate. However, the system has the following major weaknesses:

- Generating a list of students failing to attend the lecture involves manual comparison of all collected cards against a master list which requires many hours to prepare. At least three hours for Code 32 alone.
- Individually stamping the date on every card is not only labor-intensive but serves little purpose since the card does not have any information on what lectures were attended.
- Collecting student cards requires one person per Curricular Office Code.
- System is obsolete in terms of today's available technology.

B. MAGNETIC STRIPE CARD SOLUTION

With magnetic stripe card technology, every student required to attend the SGL can be issued a magnetic stripe card which will contain all the necessary information about the cardholder such as name, rank, social security number, curriculum, etc. The magnetic stripe card will replace the 3" x 5" card mentioned earlier. To implement the new attendance system, all that is required is a magnetic stripe card reader that will capture the information from the magnetic stripe, a computer, and the software program to manipulate the captured data and generate all the necessary reports. Since all data manipulations are handled by the computer, all required reports can be made available to managers within seconds after the

last student swipes his card. This new system will require one person whose job is only to ensure the system is set up properly and each student swipes only his/her own card. The total man-hours required for the new system can be computed by adding the set-up time (approximately 15 minutes) and the time necessary for data capture (approximately 20 minutes). If this man-hour requirement is translated into dollar amount as was done earlier for the current system, the magnetic stripe technology solution will only cost $5,600 (.58 hr. x $20 x 4 weeks x 10 curric offices x 12 months) a year plus the added advantage of having all the necessary reports within seconds after the last student swipes his/her card.

C. IMPLEMENTATION DETAILS

This section covers all the necessary steps in implementing the magnetic stripe solution. It includes a listing and description of hardware and software requirements, encoding of student information on the magnetic stripe, programming of the application, and finally, a short discussion of the problems and difficulties encountered during the implementation phase.

1. Hardware

As with any computer-based application, implementation of magnetic stripe solution requires some computer hardware and its peripheral equipment. Listed and described below are the minimum hardware requirements for the implementation of this project.

a. IBM compatible Lap Top

The AST PowerExec lap top computer was used for creating and testing the program for encoding the student ID cards, and the application program for the demonstration. In addition, the lap top computer was a major part of the system set up for capturing and manipulating of data during and after the demonstration. Although, other computer systems with the capability to run a C++ compiler and Microsoft DOS version 3.3 or higher may have accomplished the same results, the lap top model was highly preferred mainly because of its portability.

b. Datacard 160 Controller

The Datacard 160 Controller, manufactured by Datacard Corporation, is a small embossing and encoding system that can be used for any application requiring an embossed or magnetically encoded card. The controller is capable of receiving inputs from a host computer, PIN pad, magstripe reader, or a keyboard. For this demonstration, the keyboard was utilized as the means of inputting the required student data into the controller.

c. Datacard 110 Card Reader

The Datacard 110 Card Reader system, manufactured by Datacard Corporation, is a combination of a keyboard interface decode unit and a magnetic stripe reader. The system can be programmed to read either tracks 1, 2 or a combination of both tracks. With the aid of a pen light, which can be attached to the rear panel of the interface unit, the system can be configured to include keyboard function codes that will exclude input delimiters and capture only the desired data from the magnetic stripe. For the magnetic stripe technology demonstration, the Datacard 110 card reader system was used to capture the necessary data from the students' ID cards.

d. Printer

The magnetic stripe application developed for this technology demonstration has the capability to generate and display reports either on the screen or via a printer. Any printer that is capable of receiving data from the computer's printer port can be used with this application.

2. Software

The following software products were used to develop the application programs for the magnetic stripe technology demonstration.

a. Datacard 160 Configurator

The Datacard 160 Configurator is proprietary software distributed by Datacard Corporation as an integral part of the Datacard 160 System. This software allows

the user of the Datacard 160 system to develop and compile programs or routines, on an IBM-compatible computer, that can be tested for accuracy through simulation and later downloaded to the controller for actual implementation. The 160 configurator was utilized during development of the magnetic stripe technology demonstration to configure the datacard 160 encoder to record the desired student's information on track one only.

b. Borland C++ Compiler Version 4.0

The Borland C++ compiler was utilized to create the magnetic stripe application program used in the demonstration. Other program compilers currently available, such as Ada, C, Pascal, etc., although not tested during the development of this project, may be used to develop various magnetic stripe applications.

3. Encoding Of Student Data

The first step in implementing a card-based application is to decide what data are required to be encoded on the card's magnetic stripe. For the magnetic stripe demonstration application program, the following student data were encoded:

- Social Security Number
- Full Name
- Rank
- Curriculum Number
- Curriculum Office Code

Encoding the above information on a magnetic stripe was accomplished by programming the Datacard 160 Controller using the Datacard 160 Configurator. As mentioned earlier, the Configurator allows the application developer to control and configure the data to be encoded on the magnetic stripe. The Configurator is somewhat similar to other Compilers available in the market in that, it also has an editor environment, various compiler commands, and a simulator which allows the developer to test his/her program without having any link between the PC and the Datacard 160 system.

The data configuration for the student ID card has been designed to be encoded only on track 1 of the magnetic stripe. The data has a maximum length of 40 alpha-numeric characters including two characters for start and end sentinels. Figure 13 below illustrates a sample data format encoded on the Student's ID card.

%%547475556/DOLLETE, RODOLFO G/03/368/32?

start sentinel:	**%%**
end sentinel:	**?**
social security no:	**547475556**
full name:	**DOLLETE, RODOLFO G**
rank:	**03**
curriculum:	**368**
curriculum code:	**32**

Figure 13. Encoded Student Data.

The following is the program segment used to implement the encoding of student data shown in Figure 13 on the magnetic stripe. The complete program source listing is included in Appendix A.

```
Create STUDENT_DATA. Alphanumeric. Variable. Length: 40. " "
Create IATA. Alphanumeric. Variable. Length: 76. " "
Label START_ENCODE
Prompt "SSN/LNAME, FNAME MI./RANK/CURR/OFFCODE?" (on the
        Display). Line number: 1
Prompt " Enter Track 1 Data Using Format Above" (on the Display). Line
        number: 2
Pause for 3 seconds
Clear line 2
Keyin STUDENT_DATA starting at 1 of length 40. Row 2. Col 1.
    Enable/disable <ESC>: <ESC> disabled. Case conversion: To upper.
Copy from STUDENT_DATA at 1 to IATA at 1 of length 40
Clear line 1
Prompt "Insert card to be encoded into the 160" (on the Display). Line
        number: 1

Encode Track Field
        1 IATA
Keep or release card: Release
On error goto ENCODE_ERROR
Prompt "     Card Encoding Successful!" (on the Display). Line
        number: 2
Pause for 2 seconds
Goto MAIN
```

4. Application Development

Most application programs have some kind of input and output data. The magnetic stripe demonstration program is no exception. It requires two input data in the form of text files, processes them, and produces two output data files in the form of reports which can be viewed by the user on the screen or on hard copy via a printer. This section will include discussions of input/output data, and program segments used for data manipulation. An overview of the application design is illustrated in Figure 14 below.

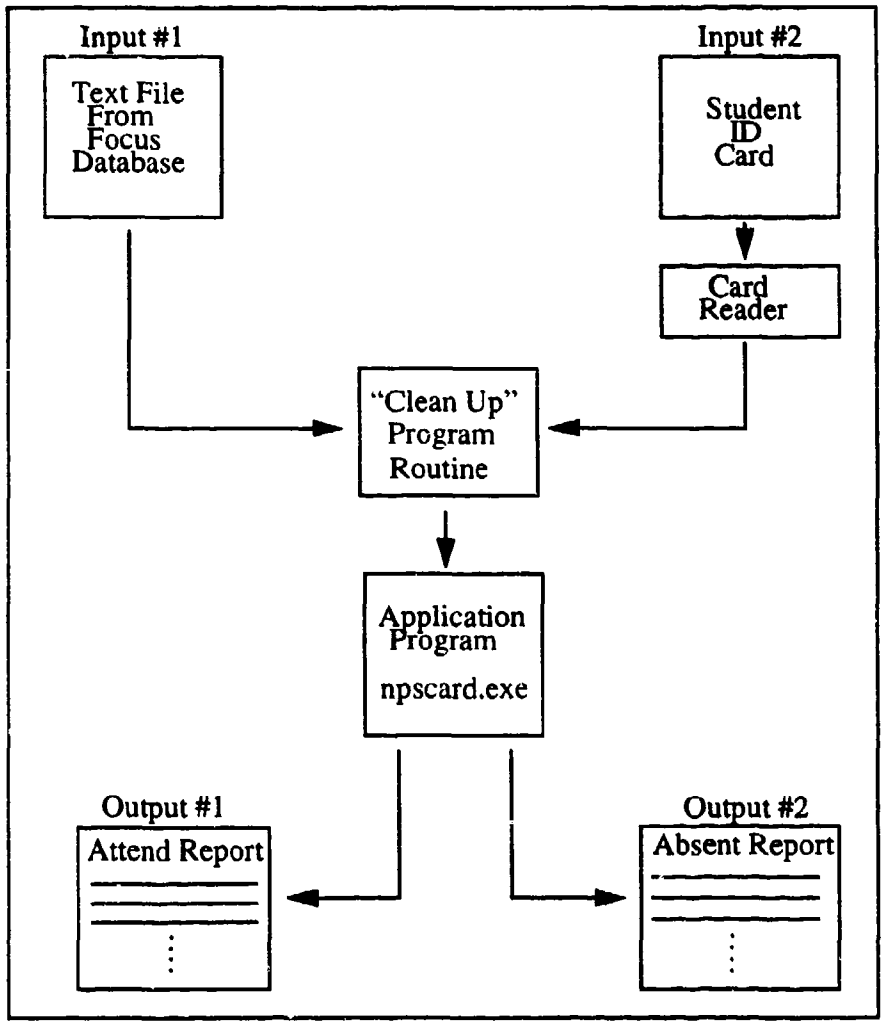

Figure 14. Magstripe Demo Program Design Overview.

a. Input

The magnetic stripe demonstration program has two input data sources. The first input is a flat text file from the school's "Focus" database. The data format of this text file is shown in Figure 15 below.

% 547475556 / DOLLETE, RODOLFO G / 03 / 368 / 32 /

start sentinel:	**%**
end sentinel:	**/**
social security no:	**547475556**
full name:	**DOLLETE, RODOLFO G**
rank:	**03**
curriculum:	**368**
curriculum code:	**32**

Figure 15. Format of text file from "Focus" database.

Notice that the format is almost similar to the data format of the encoded student data on the magnetic stripe illustrated in Figure 13. The only differences are the start/end sentinels, and the presence of blank spaces around the text delimiters on the database text file. The second input is the information from the student's ID card.

Both input data formats must go through a "clean up" program routine to convert the raw data into usable format by removing the text delimiters. The following program segment was used to convert the captured student data from the student ID card into a format that is usable by the main application program. The conversion process is graphically illustrated in Figure 16.

```
for (int a = 2; a < 60; a++)
{
      if (tempName[a] == '/')
            newFile << endl;
      else
            if (tempName[a] == '?')
            {
```

```
                    newFile << endl;
                    break;
               }
               else
                    newFile << tempName[a];
}
```

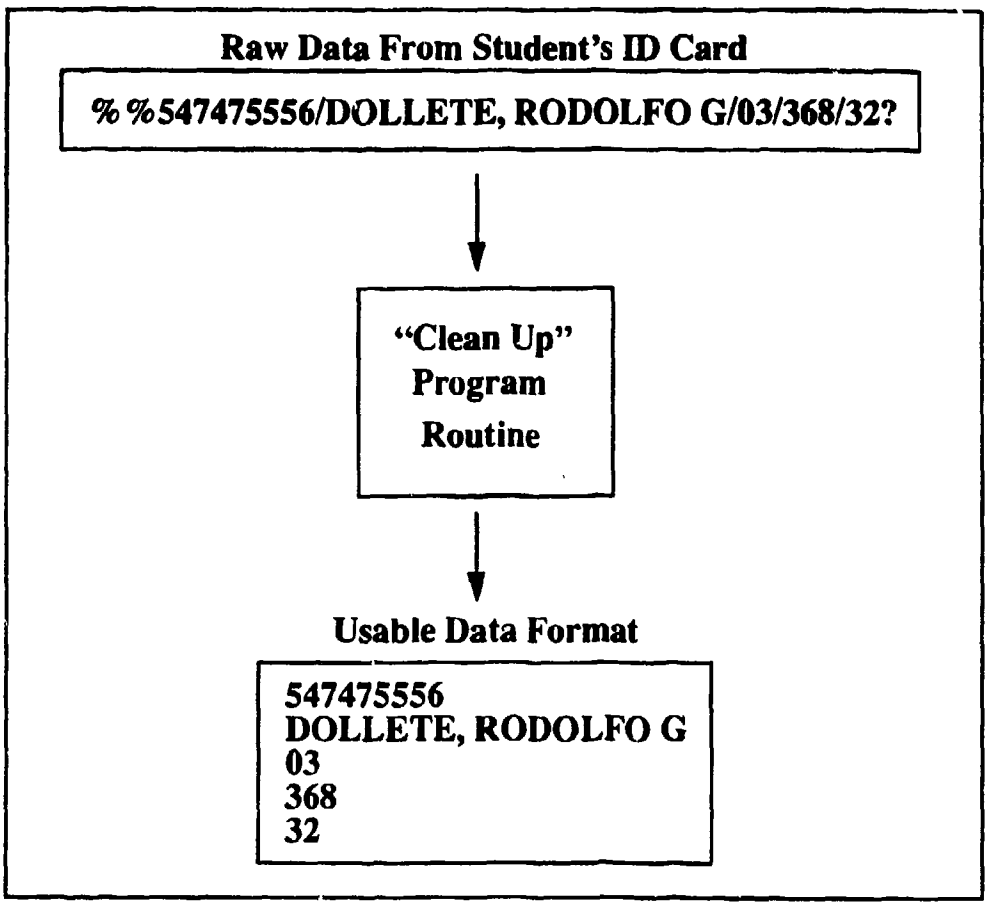

Figure 16. Conversion of Raw Student Data.

b. Output

The magnetic stripe demonstration program generates two reports as its output. The first is a text file report listing all the students who attended the Superintendent's Guest Lecture (SGL). Generating this report is quite simple as it only requires capturing of data from the students' ID cards, removing of the text delimiters, and

finally, creating a new text file with a header, current date, and all the reformatted information from each of the cards.

The second output is also a text file report listing all the students who failed to attend the Superintendent's Guest Lecture. This report is more useful to school managers than just a listing of all the attendees. The absentee report, however, is not quite as easy to generate as the first report. It requires hundreds of computer search and compare operations. This is accomplished by going through the entire student database list and comparing each student's social security number against a list of social security numbers of those students who attended the lecture. The students whose SSN's are not found on the list of attendees are the ones included in the absentee report. The absentee report format is the same as the attendee report. The following is the program segment used to generate the absentee report.

```
for (int j=0; j < totalRec; j++)
{
      getMastRecord();

      if (!compRecord())
      {
            absentFile << tempSSN;
            absentFile << "\t" <<  Dept << "\t" << Sect;
            absentFile << "\t" <<  Rank;
            absentFile << "\t" << FName << endl;
      }
}

int Card::compRecord()
{
      inFile.open("cardkey.txt", ios::in);
      int found = 0;
      char SSN[15];

      while (inFile.getline(SSN, 14))
      {
            if (!strcmp(SSN, tempSSN))
            {
                  found = 1;
                  break;
            }
      }
      inFile.close();
      return found;
}
```

D. PROBLEMS ENCOUNTERED

There were numerous problems encountered during the development of the magnetic stripe technology demonstration program. While most of the difficulties can be considered normal in developing new applications using new software and hardware products, the major ones are discussed in this section in order to preclude others who might be interested in developing new applications from encountering the same stumbling blocks.

1. Datacard 160 Configurator

This editor/compiler for programming the Datacard 160 Controller does not have its own "how to" manual which makes it difficult to learn especially for someone who has not programmed in this new environment. Although the available commands are listed in the Datacard 160 Manual, there are no instructions or sample programs that a new user can use to speed up the learning process. After several phone calls to the manufacturer's technicians, however, sample programs and assistance on how the system works were made available. It turned out that this new programming environment was not difficult to learn after all.

Future programmers or application developers can avoid the same initial difficulties by studying the encoding program included in the appendix of this thesis, and actually typing it line by line using the configurator in order to discover how the system works. The editor is menu driven which makes it easy to use. For example, the following steps will display "Hello World!" on the Datacard 160 Controller's display window:

- Go to C:\160 and type main.exe then enter.
- Choose Edit Job list from the Main Menu
- Choose Open a New file from the File Menu
- Type c:\160\hello.job as the path name of your new file.
- Type Hello World! as the description of your new job.
- While in a blank editor screen, hit the Insert key to display the Instruction Menu
- Choose Prompt from the Instruction Menu.
- Type "HELLO WORLD!" in the TEXT field.

- Choose Display. (The other choice is Printer.)
- Choose 1 for Line Number. The 160 can only display two lines.
- Hit the ESC key after making all the desired choices.
- Hit the Insert Key again and choose "Pause" from the instruction Menu.
- Choose the amount of time for the delay. The default is 3 seconds. This is the amount of time that the words "HELLO WORLD!" will remain on the display window.
- Hit the ESC key to return to the editor window.
- Hit Shift F2 to compile your job.
- Hit the ESC key twice to exit from the editor window.
- Choose Simulate 160 from the Main Menu.
- Choose 1 then press enter from the 160 Display Menu.
- Find the "HELLO" job and choose its corresponding number.
- "HELLO WORLD!" should appear on the display window for 3 seconds.

The simple "HELLO WORLD!" program created above has only two lines consisting of the text to be displayed and the amount of time for it to remain on the display window. The two commands used in the program are "Prompt" and "Pause". Other commands such as "Create", "Clear", "Set", "Label", and others can be invoked in the same manner as the steps listed above.

2. Downloading From PC To 160

Downloading of the compiled jobs from the PC to the Datacard 160 Controller is well documented in the Datacard 160 Manual. However, the manual assumes that the user has a working knowledge of both the Datacard 160 configurator and the Datacard 160 Controller. Although this was not a major difficulty encountered during the development of the magnetic stripe demonstration program, the following amplifying steps are included here to be used in conjunction with the steps provided in the manual in order to speed up future application development.

- Ensure that the Datacard 160 Controller is properly hooked up to the PC.

- Ensure that you have a compiled job to be downloaded (i.e. Hello.job)
- Choose SET UP from the 160 Display Menu
- Enter your PASSWORD when prompted.
- Choose LOAD CONFIG from the 160 Display Menu.
- Choose Download to 160 from the Configurator's Main Menu.
- Choose the appropriate communication port.
- Choose the job to be downloaded (i.e. hello.job).
- Downloading should start at this point.
- Test the downloaded program using the Datacard 160 Controller.

3. Host Mode Configuration

The Datacard 160 System's design includes a Host Mode option. This option allows the application developer to encode magnetic stripe cards without using a keyboard to manually type in the data to be encoded. Instead, the data is downloaded from the PC to the Datacard 160 encoder/decoder. This means that the data to be encoded can be in a text file or some database format. The host mode option also allows the Datacard 160 encoder/ decoder to decode data from encoded cards and save the information under a file name which can be transferred to a PC or printer at a later time. Both of these features were tested but were not fully implemented during the development of the magnetic stripe technology demonstration program. The host mode option, however, remains an attribute of the Datacard 160 System which could be further explored for future application projects.

4. Data Capture From Card To PC

For the application program to work, data from the card must be captured, manipulated, and some form of output generated. Initial attempts to use the Datacard 160 Controller to capture data for later transfer to the PC under the host mode option, as mentioned earlier, were unsuccessful. After several unsuccessful technical assistance via the telephone with the manufacturer, we decided to use the Datacard 110 card reader instead of the Datacard 160 encoder/decoder to capture data from the encoded cards. It

turned out that the Datacard 110 card reader was easier to use and should have been our first choice for capturing data from the encoded cards for the demonstration program.

V. IMPLEMENTATION OF SMART CARD TECHNOLOGY

This chapter covers the development of smart card applications to enhance the check-in/out process at NPS. Three applications are developed and embedded in the smart card chip. The first is the Personal Identification Card which serves as a source of accurate and consistent student data during check-in/out process, and a means by which the cardholder can be identified. The second application, Electronic Purse, is developed and integrated as part of the package, although not currently a problem at NPS, to demonstrate the smart card's audit trail capability. The third application embedded in the chip is the Electronic Check In/Out itself. These applications demonstrate the fact that, with smart card technology, one or more independent applications may be imbedded in one card without loss of file integrity. In addition, the applications will show three of the many advantages of smart card technology over the magnetic stripe technology. These advantages include the smart card's greater memory space, application security features, and specific file or record modification feature without re-writing the whole smart card chip.

The chapter will include discussions of the application specifications, hardware and software requirements, application development, and finally, a short discussion of the problems encountered during the application development process.

A. APPLICATION SPECIFICATION

The specification of each application embedded in the smart card chip reflects the user's business policy on how the application should behave under all possible scenarios within the application's domain. It includes the description of all the features the user wants for each application. As a minimum, it should have a description of input and output data, and the appropriate level of security to protect the application from unauthorized users. Depending on the nature of the application and the user's desires, this document can be very simple as the example applications below, or so detailed that it could fill up several pages.

1. Personal Identification Card

Implementing the personal identification card using the smart card is similar to using the magnetic stripe technology, discussed in the previous chapter, with the exception of the recording medium. In both instances, the developer needs to decide on what information is necessary to be maintained on the card. For the personal ID demonstration, an area of memory on the smart card chip must be set aside to hold one record containing the owner's name, social security number, rank, curriculum number, curriculum office code, birthday, and emergency telephone number. In order to make this example as simple as possible, submission of any type of keys will not be required.

2. Electronic Purse

Implementation of electronic purse using smart card requires setting aside a specific area of memory to hold the account's current balance and a set of logic/security functions on the smart card chip. The purse must be a dynamic account that can be credited and debited. Furthermore, the purse must be protected by the owner's personal identification number (PIN), and an audit trail of at least 10 of the most current transactions must be maintained within the smart card chip.

3. Check In/Check Out

The Electronic Check In/Check Out must be designed to ensure all students reporting to or detaching from the Naval Postgraduate School have properly checked in/out with all the departments listed on the hard copy check in/out sheet. As in the implementation of the electronic purse, implementing the Electronic Check In/Out requires setting aside a specific smart card chip memory area to hold the cardholder's check in/out status.

B. HARDWARE AND SOFWARE REQUIREMENTS

This section covers the discussions of the minimum hardware and software requirements for the implementation of smart card applications.

1. Hardware

The minimum hardware requirements listed below have been used for implementation of the smart card demonstration program for NPS.

a. IBM compatible Lap Top

This computer system was used for creating and testing the program for encoding the student ID cards, and the application program for the demonstration. In addition, the lap top computer was a major part of the system set up for capturing and manipulating of data during and after the demonstration. Although, other computer systems with the capability to run a C compiler may have accomplished the same results, the lap top model was highly preferred mainly because of its portability.

b. Datacard Series 50

The Datacard Series 50, manufactured by Datacard Corporation, is a smart card reader/writer which passes commands and data between a smart card and a host PC. It is best suited for applications where the program and decision processing reside on the host and a low-cost smart card reader is required. The Series 50 used for the Smart Card Demonstration Program has an automatic eject mechanism and an LED which indicates green when the unit is first powered on. The LED changes color from green to orange to indicate that the smart card is ready for communication with the host computer. A change of color from orange to green means that the communication with the smart card has been properly shut down. The Series 50 reader/writer must be connected to the host PC using an RS-232 cable for all types of smart card communication.

c. 2K Smart Card

The 2k EEPROM microprocessor smart card, manufactured by Datacard Corporation, was used for the NPS smart card technology demonstration program. The card conforms to ISO 7816 which promulgates the standards for credit card dimensions, and has

a total of 1920 bytes of available memory to store user application files. This card is illustrated in Figure 17 below.

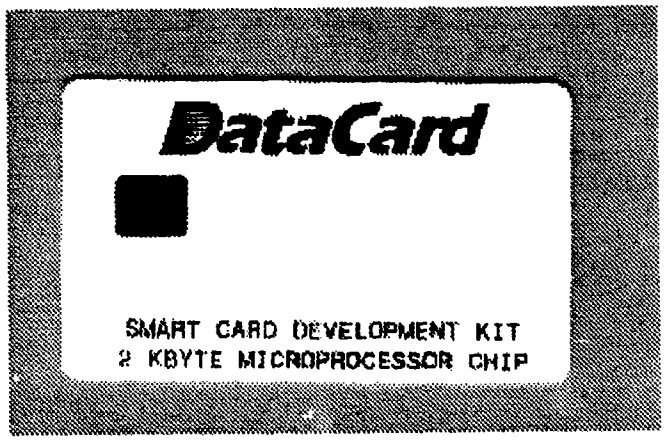

Figure 17. A 2K EEPROM Smart Card.

2. Software

The following software products were used to develop the application programs for the smart card technology demonstration.

a. Datacard Development Kit

The Development Kit software package, distributed by Datacard Corporation, contains a library of C-language functions known as the Datacard Command Set (DCOS) library which allows programmers to access the Smart Card Operating System directly. This library provides a powerful environment for creating applications. The library is a superset of the Smart Card Operating System (SCOS), and it provides complete control to read, write, edit, and format integrated circuit modules operating under SCOS.

b. Borland C++ Compiler Version 4.0

The Borland C++ compiler was utilized to create the smart card application program for the smart card technology demonstration. The program is written in C since,

as mentioned earlier, the functions in the Datacard Command Set Library (DCOS) are written in the C language.

C. APPLICATION DEVELOPMENT

Smart card application development can be done in many ways depending on the programmer's programming style and the requirements of the application. In almost all cases, however, there must be a definition of some type of data structure, a smart card file definition, and various routines for verifying or modifying the contents of the files. In developing the smart card technology demonstration for NPS, all three definitions were used and a detailed discussion of each are included below.

1. Personal Identification Card

Identification of personnel is one of the most popular applications of card technologies. Because of cost considerations and well established infrastructure, the magnetic stripe technology is the preferred medium for ID applications. The banking industry, for example, uses it extensively in the form of credit cards and ATM cards. The smart card technology can accomplish exactly the same functions as the magnetic stripe with the option of adding more ID data or security features such as multi-level password, fingerprint, voice recognition, and other biometric identification. Additional data is available because of the greater space available in the smart card chip compared to the magnetic stripe. The student ID information can be used for identifying the cardholder and as a source of basic information required by various departments during check-in/out process. Details of implementing the smart card Personal Identification application are discussed below.

a. Record Definition

The Personal Identification Card portion of the smart card application was implemented by first establishing the types of information that will be encoded in the smart card chip. For the demonstration program, the cardholder's last name, first name, middle

initial, social security number, rank, curriculum number, curriculum office code, birth date, and telephone number are included. The data structure chosen to hold the cardholder's information is the record. The program segment used to create the record components is listed below.

```
typedef struct {
      BYTE LName[10];
      BYTE FName[10];
      BYTE MI[2];
      BYTE SSN[10];
      BYTE Rank[5];
      BYTE Curric[4];
      BYTE CurrCode[3];
      BYTE BDate[7];
      BYTE TelNum[8];
      } ID_Info;
```

b. File Definition

There are two ways of creating a file definition. The first is by calling a routine called "CreateFileDef" with all the desired parameters (see Appendix B for a full listing of this function). The syntax for this function is as follows:

int CreateFileDef (BYTE FileNum, BYTE Mode, BYTE Security,
BYTE RecNum, BYTE RecLen, BYTE *FileName)

where:

FileNum ----------File number value.
Mode ----------Mode value.
Security ----------Security value.
RecNum ----------Number of records in the file.
RecLen ----------Length of the records.
*FileName ----------Pointer to file name.

For the smart card personal identification file definition, the following values were used:

FileNum ----------0 (first file)
Mode ----------1 (Normal mode)
Security ----------0 (No security)
RecNum ----------1
RecLen ----------64 (record length in bytes)

*FileName ----------ID (stands for Identification)

Using the syntax format and the corresponding parameter values listed above, defining the Personal Identification File on the smart card chip becomes as simple as making the function call: *CreateFileDef(0, 1, 0, 1, 64, "ID")*.

The second method of making file definitions on a smart card chip uses the data structure declared in DCOSDEFS.H, a header file included with the smart card development package. The structure is named "CHIP_FILE_DEF" and it contains 16 one-byte record elements. Making a smart card file definition using this method requires that a variable of type CHIP_FILE_DEF be declared and initialized with the desired parameters. Although the parameters are exactly the same as the ones declared for the first method, they appear different in the second method because they are written in hexadecimal system. Although the second method is more cumbersome to use compared to the first method, it is an excellent learning tool for someone just beginning to build smart card applications. For example, the following declaration creates the same Personal Identification file definition on the smart card chip as above. The hexadecimal numbers enclosed in brackets are the parameters used for defining the file name "ID". Table 3 explains the meaning of each of these parameters.

CHIP_FILE_DEF ID = {0x07, 0x3C, 0x00, 0x00, 0x00, 0x40, 0x01, 0x01,
0x49, 0x44, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

Record Elements	Hex Value	Decimal value	Explanation
0/1	07 / 3C	1852	Beginning address of the file "ID".
2	00	0	No protection for write or read.
3	00	0	No key required for writing.
4	00	0	No key required for reading.

Table 3: File Definition Parameters For Method 2.

Record Elements	Hex Value	Decimal value	Explanation
5	40	64	Record length.
6	01	1	Number of records in the file.
7	01	1	Mode is erasable and normal.
8/9	49 / 44		File name "ID". I = 49, D = 44
10 - 15	00	0	Always set these elements to 00.

Table 3: File Definition Parameters For Method 2.

c. Encoding Of ID Data

Once the data and file structure have been defined on the smart card chip, encoding of data is quite simple. All we need to do are assign values to the variables declared in the data structure definition, search for the desired file name in the smart card memory, and saving the information into that file name. The sample program segment below illustrates the encoding of Personal Identification data into the smart card chip under the file name "ID".

```
printf("\n\tEnter your last name: ");
scanf("%s", TempName);
strcpy(Identification->LName, TempName);
printf("\n\tEnter your first name: ");
scanf("%s", TempName);
strcpy(Identification->FName, TempName);
printf("\n\tEnter your middle initial: ");
scanf("%s", TempName);
strcpy(Identification->MI, TempName);
     :
     :
SaveFileData("ID", 1);
```

The codes for the actual searching and writing to the file named "ID" is accomplished in the "SaveFileData" function. A complete listing of this function is included in Appendix B.

d. ID Verification

The smart card application program developed for the smart card demonstration has the capability to access and display the contents of the smart card chip set aside for the ID application. This capability can be activated by choosing "Verify Cardholder's ID" from the Administrator's Menu selections. The verification process involves searching for the file named "ID" from the smart card memory, and reading its contents into a buffer. The program codes for searching and reading are contained in the function "GetFileData". The following is the program segment used to access and display the card owner's information.

```
clrscr();
DCOS_ResetChp(ResetInfo, &ResetBuffLen);
GetFileData("ID");
printf("\n\t\t\tName:       %s %s. %s \n",
     Identification->FName,
Identification->MI, Identification->LName);
printf("\t\t\tSSN:        %s \n",  Identification->SSN);
printf("\t\t\tRank:       %s \n", Identification->Rank);
printf("\t\t\tCurriculum: %s \n",
     Identification->Curric);
printf("\t\t\tCurrCode:   %s \n",
     Identification->CurrCode);
printf("\t\t\tBirthDate:  %s \n", Identification->BDate);
printf("\t\t\tTelephone:  %s \n",
     Identification->TelNum);
printf("\n\n\t\t\t %s", pressAnyKeyMsg[0]);
```

2. Electronic Purse

In implementing the Electronic purse application using smart card, a specific area of memory on the smart card chip was set aside to hold the current account balance. As required by the specification mentioned earlier, the purse account has been designed so that it can be credited and debited upon successful submission of the user's PIN key. The basic processes used in the electronic purse implementation are discussed below.

a. Record Definition

Implementing the audit trail feature of the Electronic Purse application program requires that a data structure be set up to hold the audit trail information.

Specifically, the data structure should be able to keep track of the type of transaction, i.e. debit or credit, the amount of transaction, and the date of transaction. For the audit trail feature of the smart card demonstration program, the following data structure definition was used.

```
typedef struct {
      BYTE Type;
      BYTE Date[3];
      DWORD Amount;
      } TransActInfo;
```

b. File Definition

Implementing an electronic purse on a smart card with the specifications mentioned earlier, requires two file definitions. The first file will hold the current account balance, and the second will be used to keep track of the 10 most current account transactions. With the exception of changes in parameter values, the function call to create the two electronic purse file definitions are the same as the function call for the Personal Identification file definition. The following function call with the desired parameters creates the file definition for the electronic purse account balance. Note that the mode of this file has a value "1" which means the file uses the normal mode that allows for the new account balance to overwrite the old balance as long as the writing process is done randomly. Furthermore, this file definition also has a security value of "32" which requires that the Issuer's key and a PIN key be submitted to the smart card before any changes to the current balance is accepted. The rest of the parameters with their corresponding values are explained below following the function call.

```
CreateFileDef(1, 1, 32, 1, 4, "AB");
```

FileNum ----------1 (second file on the chip)
Mode ----------1 (normal mode)
Security ----------32 (Issuer's and PIN keys required)
RecNum ----------1 (number of records in the file)
RecLen ----------4 (length of records in bytes)
*FileName ----------AB (stands for Account Balance)

The file definition for the electronic purse audit trail is very much similar to the Personal Identification file definition with the exception of the changes in parameter values. Specifically, the Mode value of this file is "2" which means that it is a non-erasable circular file. This file mode allows the oldest record to be overwritten once the maximum number of records is reached and an additional record is added to the file. This file does not require any key as indicated by the security parameter value. Listed below is the function call, with all the indicated parameter values, to create the file definition for the audit trail feature of the electronic purse.

CreateFileDef(2, 2, 0, 10, 8, "AT");

FileNum ----------2	(third file on the chip)
Mode ----------2	(Non-erasable mode)
Security ----------0	(no security)
RecNum ----------10	(number of records in the file)
RecLen ----------8	(record length in bytes)
*FileName ----------AT	(stands for Account Transaction)

c. Encoding of Electronic Purse Data

The value of the electronic purse is initialized to $200.00 at the time the card is issued to the user. In order to allow for easy data manipulation, i.e. addition, subtraction, and comparison, the value is stored in long integer which is the same as storing the value in cents instead of dollars. The initial value is stored in the file named "AB" by calling the function `SaveFileData("AB", 1)`. The second parameter is a file type which controls the manner of writing of data into the file. A value of one means that the data will be written randomly, while a zero value means sequential writing of data into the file. The sample program segment below initializes the value of the electronic purse.

```
DWORD *Balance;
Balance = (DWORD *) FileData;
*Balance = 20000L;
SaveFileData("AB", 1);
```

The encoding of data into the audit trail file is accomplished in the same manner as above with the exception of an additional variable "Type" which keeps track of

the two types of transactions, a value of one means credit transaction while a value of zero means debit transaction. Note that the encoding of the audit trail data requires a call to another function called "SaveTransAction" which handles the inclusion of the current date with every account transaction. This function, in turn, calls the SaveFileData function to save the entire transaction into the smart card chip under the file named "AT". The sample program segment below creates the audit trail record for the electronic purse application.

```
Transaction = (TransActInfo *) FileData;
Transaction ->Type = 1;
Transaction ->Amount = 20000L;
SaveTransAction(Transaction);

int SaveTransAction(TransActInfo *Transaction)
{
      int   Status;
      struct tm *curtime;
      time_t bintime;
      time (&bintime);
      curtime = localtime (&bintime);
      Transaction->Date[0] = curtime->tm_mon + 1;
      Transaction->Date[1] = curtime->tm_mday;
      Transaction->Date[2] = curtime->tm_year;
      Status = SaveFileData ("AT", 0);
      return (Status);
}
```

d. Viewing The Audit Trail

Viewing the contents of the audit trail file is accomplished by searching the smart card chip for the audit trail file name. Since we know the file name to be "AT" (refer to file definition above), the function call *Status = DCOS_FindFile("AT", &FileNum, FileDef)* will do the search for us. If the status of the search is NO_ERROR, which means the file exists, the next step is to determine if the file is empty by testing the value of the end of file marker. Displaying the contents of a non-empty file can then be easily done by randomly reading the file with a decrementing record number. This will display the most current transactions first since the first record always starts at 0. The program segment below displays the contents of the audit trail file of the smart card electronic purse application.

```
Status = DCOS_FindFile("AT", &FileNum, FileDef);
if (Status == NO_ERROR)
```

```
{
    RecLen = FileDef[5];
    EndOfFile   = FileDef[11];

    if (EndOfFile == 0)
        Status = ICERFEMP;     //IC Error File Empty
    else
    {
        Record = FileDef[6] - 1;
        clrscr();
        printf("\n%s", transActMsg[0]);
        printf("%s", transActMsg[1]);

        do {
            Status = DCOS_Rd_RdmRc(FileNum,
                Record, (BYTE *) TransAction, RecLen, (short *) &RtrnLen);

            if ((Status == NO_ERROR) && (TransAction->Amount != -1))
            {
                printf("\n\t\t%d/%d/%d", TransAction -> Date[0],
                                        TransAction -> Date[1],
                                        TransAction -> Date[2]);
                itoa(TransAction->Amount,string, 10);
                i = strlen(string);
                i -=2;
                printf("\t\t$%5d", TransAction->Amount/100);
                printf(".%c%c", string[i], string[i+1]);

                if (TransAction->Type == 0)
                    printf("\t\tDebit");
                else
                    printf("\t\tCredit");
            }
        --Record;
        } while ((Record >= 0) && (Status == NO_ERROR));
    }
}
```

3. Check In/Check Out

The check in/check out portion of the smart card demonstration program is an application which has been designed to ensure that all personnel reporting to or detaching from NPS are given the opportunity to meet with all the departments listed on the check in/ out sheet. The main contributions of this application are accurate personnel accounting for all the departments listed on the check in/out sheet, reduction of paperwork, and reduction of check-in processing time as a result of having most of the information necessary in every check-in station such as Name, Rank, SSN, etc. available in the student's smart card. Furthermore, since the smart card serves as the source of redundant information, accuracy

and consistency of student records in all departments will be maintained. The following are the processes involved in implementing this application.

a. Record Definition

The data structure listed below includes all the departments that a student reporting to or detaching from NPS need to see. The departments are also listed in the same order on the application's selection menu.

```
typedef struct {
        BYTE CurrOff;
        BYTE ServRep;
        BYTE StuMailCen;
        BYTE FamServCen;
        BYTE ClasMatCtrl;
        BYTE SCIF;
        BYTE Library;
        BYTE SecMgr;
        BYTE BOQ;
        BYTE Dental;
        BYTE PSD;
        BYTE NEX;
        BYTE LicExam;
        BYTE Registrar;
        BYTE CompCtr;
        BYTE BaseSec;
        BYTE Medical;
        BYTE HRO;
        BYTE Admin;
        } CheckInInfo;
```

b. File Definition

The file definition for the check in/out application is done in exactly the same manner as the applications previously covered with the exception of the changes in parameter values. Listed below is the function call, with all the indicated parameter values, to create the file definition for the check in/out application.

CreateFileDef(3, 1, 0, 1, 20, "CI");

FileNum	3	(fourth file on the chip)
Mode	1	(Normal mode)
Security	0	(no security)
RecNum	1	(number of records)
RecLen	20	(record length in bytes)

*FileName ----------CI (stands for Check In)

c. Encoding of Check In/Out Data

All the departments listed in the check in/out application are initialized with a zero value which means that the cardholder has neither checked in nor checked out from any department at the time the card is issued. The process includes declaring a variable of type *CheckInInfo* and assigning the value zero to all its record elements. The record is then saved under the file name "CI" for future verification or modification. The initialization process is accomplished by the following program segment.

```
CheckInInfo *CheckStatus;
CheckStatus = (CheckInInfo *) FileData;
CheckStatus -> CurrOff = 0;
CheckStatus -> ServRep = 0;
CheckStatus -> StuMailCen = 0;
CheckStatus -> FamServCen = 0;
CheckStatus -> ClasMatCtrl = 0;
CheckStatus -> SCIF = 0;
CheckStatus -> Library = 0;
      :
      :
CheckStatus -> Admin = 0;

SaveFileData ("CI", 1);   //1 means random write
```

The process of modifying the file to reflect the current check in/out status is accomplished by simply searching for the file named "CI" in the smart card's memory, changing the value assigned to the appropriate department, and saving the entire file as we have done earlier during initialization. The following program segment illustrates the codes used for checking in or out with the Curricular Officer. The variables, selection1 and selection2, refer to the type of transaction (i.e. check-in or check-out) and the appropriate department chosen from the selection menu, respectively.

```
GetFileData("CI");

switch (selection2)
{
      case (1): { if (selection1 == '1')
                    CheckInStatus -> CurrOff = 1;
                  else
                    CheckInStatus -> CurrOff = 2;
                  break;}
```

```
}
SaveFileData("CI", 1);
```

d. Check In/Out Verification

This feature of the check in/out application allows the administrator to verify the status of the student's check in/out process. It involves searching for the file named "CI" which holds all the check in/out data and displaying its contents by calling the function `PrintStatus(BYTE CheckInStatus)`. The program segment that handles the display of the application's current check in/out status is listed below.

```
GetFileData("CI");
printf("\n\t\t\tCURRENT CHECK IN/OUT STATUS\n");
printf("\n\t\tCurricular Officer:          ");
Status = CheckInStatus -> CurrOff;
PrintStatus(Status);
printf("\t\tService Representative:      ");
Status = CheckInStatus -> ServRep;
PrintStatus(Status);
printf("\t\tStudent Mail Center:         ");
Status = CheckInStatus -> StuMailCen;
PrintStatus(Status);
      :
      :

void PrintStatus(BYTE CheckInStatus)
{
      if (CheckInStatus == 0)
            printf("Not Checked In\n");
      if (CheckInStatus == 1)
            printf("Checked In\n");
      if (CheckInStatus == 2)
            printf("Checked Out\n");
}
```

D. PROBLEMS ENCOUNTERED

The problems encountered during the development of the smart card applications for NPS were not as difficult to rectify as the ones encountered for the magnetic stripe technology discussed earlier. This was due, primarily, to the excellent technical manual that came with the development kit. The following recommendations are the result of the experience gained during the NPS applications development process and are included in this thesis in order to preclude others interested in pursuing further application developments from going through the same difficulties.

1. Compile/Execute The Sample Programs

Sample smart card programs are provided with the development kit and should be studied in detail prior to attempting any actual application development. The sample programs cover the basic smart card programming techniques from initializing the communication port, reading and writing of information to the smart card chip, file definitions, creation and submission of keys, etc. The amount of time invested and experience gained from learning the sample programs will pay great dividends during the development, debugging, and testing of the actual application.

2. Learn The Datacard/Smart Card Operating Systems

Many hours of program debugging which can lead to headaches and frustration can be avoided by first having a good working knowledge of the environment in which the application program will be operating in. The environment involves two proprietary systems, developed by Datacard Corporation, which are used for communication between the host computer and the smart card chip. These are the Data Card Command Set (DCOS) and Smart Card Operating System (SCOS). Detailed discussion of these systems will not be included here since both are well documented in [Data92].

VI. CONCLUSION AND FUTURE THESIS WORK

A. CONCLUSION

Smart cards are the result of combining the technologies of Magnetic Stripe and Integrated Circuits (IC). Both of these technologies have seen enormous growth during the last decade that they are now deeply embedded in every facet of our society and upon which we are slowly becoming more and more dependent. There is no doubt that, although the magnetic stripe will continue to dominate the card-based market in the near future, the smart card will eventually take over and will win against any other non-intelligent solution. This means that those who take the initiative to learn the basics of this technology and pursue its potential benefits over existing systems will also find themselves ahead of their competitors.

In this thesis, we have shown that it is possible to develop our own applications using magnetic stripe and smart card technologies to enhance various student management functions. With minimal initial investment in hardware and modifications on current systems, the Naval Postgraduate School can realize significant dollar savings in terms of reduced man-hours in conducting labor-intensive student management functions, and readily available student data that is both accurate and consistent. As one of the Navy's re-invention laboratories, NPS has the opportunity to set an example by implementing the technologies currently available as effectively and efficiently as possible.

B. SUGGESTIONS FOR FUTURE THESIS WORK

This thesis provides the necessary foundation for future research on the potential benefits of implementing Card Technologies at the Naval Postgraduate School. The application programs for both the magnetic stripe and smart card technologies, although fully functional, require many upgrades which is considered normal for most initial software versions. The following "nice-to-have features" were discovered during testing

and program demonstration but were not included in the first version due to time constraints:

- The reports generated by the Magnetic Stripe Application Program do not list the students in any kind of ordering. Alphabetical listing by Last Name, and grouped by Code, Curriculum, and Section is the most preferred ordering.
- It was also strongly suggested by a Senior Observer to investigate the feasibility of using the bar code technology as a cheaper option over the magnetic stripe technology for the SGL application. He also recommends using the Military ID card as the bar code carrier instead of buying new plastic cards. The author feels confident that the Magnetic Stripe Application Program Version 1.0 will also work using the bar code technology with minimal hardware modifications.
- Explore and implement the "host mode" feature of the Datacard 160 encoder/ decoder.

To someone who is more interested in developing new applications, and willing to look and investigate around the campus, many other student management functions can be enhanced through implementation of smart card technology. The following are just a few of them:

- **Library Card.** Currently, the library uses two cards. One is embossed and the other uses bar codes. It would be nice to implement just one card which will also record the books checked-out by the cardholder.
- **Access Card.** Currently, the school uses cipher locks in most of its computer lab doors which does not provide any type of audit trail of lab users. In addition, the same card can be used to access, when authorized, the classified section of the library. Right now, the classified section is controlled by a person and an audit trail is kept using the paper log books.
- **Vehicle Registration Card.** The student's smart card can be programmed to include a memory set aside for keeping all the cardholder's pertinent vehicle information.

- **Medical/Dental Record Card.** Integrating this application in the smart card chip should be an interesting and challenging project to anyone interested in smart card programming. It would require interviewing various medical/Dental personnel to determine all the vital medical information that a medical person would normally want to know in case of emergency.

APPENDIX A. Magnetic Stripe Program Codes

```
/*****************************************************************************
*      MAGNETIC STRIPE TECHNOLOGY DEMONSTRATIONPROGRAM                      *
*                        Naval Postgraduate School                          *
*                        Monterey, California                               *
*                                                                           *
* Module:     NPSCARD.CPP                                                   *
*                                                                           *
* Description:     The following module contains the routines that make up  *
*                  the NPS Magstripe Demonstration Program.                 *
*                                                                           *
* Routines:   Name                  Description                             *
*             Card                  Class constructor.                      *
*             startNpsDemo          Starts the demo program.                *
*             showMainMenu          Shows menu of transactions.             *
*             takeAttendance        Takes SGL attendance.                   *
*             convertToTextFile     Prepares db file for processing.        *
*             createAttendReport    Writes attendance report file.          *
*             showAbsentees         Creates & shows absentee report.        *
*             compRecord            Compares attendance to master db.       *
*             printReport           Prints attendance/absentee files.       *
*             getMastRecord         Gets student record one at a time.      *
*             showOnScreen          Displays both reports on screen.        *
*             getTotalRecord        Counts total records in master db.      *
*             getDate               Provides current date.                  *
*                                                                           *
* History:    07 July 1994 (Created)                                        *
*                                                                           *
* Programmer: LT Rudy G. Dollete, USN   Internet: dollete@aol.com           *
*                                                                           *
*****************************************************************************/
    #include "npscard.h"
    #include "message.h"
    #define USER "547475556"
    ofstream newFile;
    ofstream keyFile;
    ofstream absentFile;
    ifstream readFile;
    ifstream inFile;
    ifstream masterFile;
```

```
/*************************************************************************
*                                                                        *
* Name:              Card                                                *
*                                                                        *
* Description:       This is the constructor for the Card class.  It also *
*                    displays the opening screen of the demonstration program. *
*                                                                        *
* Input:       None.                                                     *
* Output:      None                                                      *
*                                                                        *
*************************************************************************/

Card::Card()
{
    textcolor(YELLOW);

    for (int j =0; j < 16; j++)
        cout << "\t   " << versionMsg[j] << endl;

    cout << "\t\t\t\t   "; getch();
    getDate();
}
```

```
/*************************************************************************
 *                                                                       *
 * Name:          startNpsDemo                                           *
 *                                                                       *
 * Description:   Starts the demonstration program by asking user to swipe *
 *                his user card. If valid, it shows the main menu for the *
 *                demonstration program.                                 *
 *                                                                       *
 * Input:         None.                                                  *
 * Output:        None                                                   *
 *                                                                       *
*************************************************************************/
void Card::startNpsDemo()
{
    for (int k = 0; k < 4; k++)
    {
        clrscr();
        cout << swipeUserCardMsg[0];
        cout << swipeUserCardMsg[1];
        cin.getline(name, 80);
        strcpy(lname, "        ");

        for (int j=2;j<11; j++)
            lname[j-2] = name[j];

        if (!strcmp(lname, USER ))
        {
            clrscr();
            showMainMenu();
            break;
        }
        else
        {
            clrscr();
            cout << invalidCardMsg[0];
            cout << invalidCardMsg[1];
            getch();
        }
    }
}
```

```
/*****************************************************************
 * Name:          showMainMenu                                   *
 *                                                               *
 * Description:   Displays the main selection menu of the demonstration *
 *                program.                                       *
 *                                                               *
 * Input:         None.                                          *
 * Output:        None                                           *
 *****************************************************************/
void Card::showMainMenu()
{
    int selection;
    clrscr();

    for (int k =0; k < 11; k++)
        cout << "\n\t   " << mainMenuMsg[k];

    cout << enterSelMsg[0];
    cin >> selection;

    switch (selection)
    {
        case (1): {takeAttendance();
                   break;}
        case (2): {showAbsentees();
                   showMainMenu();  break;}
        case (3): {showOnScreen("attend.rpt");
                   showMainMenu();
                   break;}
        case (4): {printReport("absent.rpt");
                   showMainMenu();
                   break;}
        case (5): {printReport("attend.rpt");
                   showMainMenu();
                   break;}
        case (6): {break;}
        default:  {cout << invalidSelMsg[0] << endl;
                   cout << pressAnyKeyMsg[0];
                   getch();
                   showMainMenu();
                   break;}
    }
}
```

```
/*************************************************************************
 *                                                                       *
 * Name:          takeAttendance                                         *
 *                                                                       *
 * Description:   Requests the students to swipe their student cards from *
 *                which an attendance file is generated.                 *
 *                                                                       *
 * Input:         student cards                                          *
 * Output:        attend.db                                              *
 *                                                                       *
*************************************************************************/
void Card::takeAttendance()
{
    ofstream writeFile;
    totalRec = 0;
    writeFile.open("attend.db", ios::out);

    for (int k=0; k< 100; k++)
    {
        clrscr();
        cout << swipeStudCardMsg[0];
        cout << swipeStudCardMsg[1];

        cin.getline(name, 80);

        if (name[0] == 'q')
            break;
        else
        {
            writeFile << name << endl;
            totalRec += 1;
        }
    }
     writeFile.close();
     converToTextFile();
     createAttendReport();
     showMainMenu();
}
```

```
/*************************************************************************
 *                                                                       *
 * Name:          createAttendReport                                     *
 *                                                                       *
 * Description:   Takes an input file and generates an attendance report *
 *                from it. It also creates a key file which will be used *
 *                by the demonstration program to generate the absent.rpt.*
 *                                                                       *
 * Input:         attend.txt                                             *
 * Output:        attend.rpt  and cardkey.txt                            *
 *                                                                       *
*************************************************************************/
void Card::createAttendReport()
{
     masterFile.open("attend.txt", ios::in);
     newFile.open("attend.rpt", ios::out);
     keyFile.open("cardkey.txt", ios::out);

    clrscr();
     newFile << "\t" << attendMsg[0] << endl;
     newFile << "\t\t\t   " << month << " / " << day << " / " << year << endl;
     newFile << "      " << underLnMsg[0] << endl;
     newFile << fieldMsg[0] << endl;
     newFile << fieldMsg[1] << endl;

     for (int k = 0; k < totalRec; k++)
     {
          getMastRecord();
          newFile << tempSSN;
          newFile << "\t" <<  Dept << "\t" << Sect;
          newFile << "\t" <<  Rank;
          newFile << "\t" << FName << endl;
          keyFile << tempSSN << endl;
     }
     masterFile.close();
     newFile.close();
     keyFile.close();
}
```

```
/*****************************************************************************
*                                                                            *
* Name:          convertToTextFile                                           *
*                                                                            *
* Description:   Takes a raw input file and converts it to a format that     *
*                can be manipulated and used by the demonstration program.   *
*                                                                            *
* Input:         attend.db                                                   *
* Output:        attend.txt                                                  *
*                                                                            *
*****************************************************************************/

void Card::converToTextFile()
{
    char tempName[60];
    readFile.open("attend.db", ios::in);
    newFile.open("attend.txt", ios::out);
    readFile.getline(tempName, 60);

    for (int x=1; x < totalRec; x++)
    {
        readFile.getline(tempName, 60);

        for (int a = 2; a < 60; a++)
        {
            if (tempName[a] == '/')
                newFile << endl;
            else
                if (tempName[a] == '?')
                {
                    newFile << endl;
                    break;
                }
                else
                    newFile << tempName[a];
        }
    }
    readFile.close();
    newFile.close();
}
```

```
/*****************************************************************************
 *                                                                           *
 * Name:            printReport                                              *
 *                                                                           *
 * Description:     Takes a report file as an input and prints it to an       *
 *                  attached printing device.                                *
 *                                                                           *
 * Input:           report file                                              *
 * Output:          printed report                                           *
 *                                                                           *
*****************************************************************************/
void Card::printReport(char inFile[])
{
    ifstream readFile;
    ofstream writeFile;

    readFile.open(inFile, ios::in);
    writeFile.open("LPT1", ios::out);

    while (readFile.getline(name, 80))
        writeFile << name << endl;
    readFile.close();
    writeFile.close();
}

/*****************************************************************************
 *                                                                           *
 * Name:            getMastRecord                                            *
 *                                                                           *
 * Description:     Reads the master database file one record at a time.     *
 *                                                                           *
 * Input:           None.                                                    *
 * Output:          None                                                     *
 *                                                                           *
*****************************************************************************/
void Card::getMastRecord()
{
    masterFile.getline(tempSSN,15);
    masterFile.getline(FName, 35);
    masterFile.getline(Rank,10);
    masterFile.getline(Dept,10);
    masterFile.getline(Sect, 10);
}
```

```
/*************************************************************************
 *                                                                       *
 * Name:          showOnScreen                                           *
 *                                                                       *
 * Description:   Takes a report file as an input and displays it on the *
 *                screen.                                                *
 *                                                                       *
 * Input:         report file                                            *
 * Output:        screen display                                         *
 *                                                                       *
*************************************************************************/
void Card::showOnScreen(char inFile[])
{
    int ctr = 0;
    clrscr();
    ifstream readFile;
    readFile.open(inFile, ios::in);

    while (readFile.getline(name, 70))
    {
        ctr +=1;

        if (ctr == 21)
        {
            cout << pressAnyKeyMsg[0] << endl;
            ctr = 0;
            getch();
        }

    cout << name << endl;
    }

    getch();
    readFile.close();
}
```

```
/*************************************************************************
 *                                                                       *
 * Name:          showAbsentees                                          *
 *                                                                       *
 * Description:   Generates the absentee report based on the comparison of *
 *                master file and attendance file.  It also calls the function *
 *                "showOnScreen" to display the report.                  *
 *                                                                       *
 * Input:         master.txt                                             *
 * Output:        absent.rpt                                             *
 *                                                                       *
 *************************************************************************/
void Card::showAbsentees()
{
    masterFile.open("master.txt", ios::in);
    absentFile.open("absent.rpt", ios::out);
    clrscr();
    absentFile << "\t" << absentMsg[0] << endl;
    absentFile << "\t\t\t   " << month << " / " << day << " / " << year << endl;
    absentFile << "\t" << underLnMsg[0] << endl;
    absentFile << fieldMsg[0] << endl;
    absentFile << fieldMsg[1] << endl;

    getTotalRecord();

    for (int j=0; j < totalRec; j++)
    {
        getMastRecord();

        if (!compRecord())
        {
            absentFile << tempSSN;
            absentFile << "\t" <<  Dept << "\t" << Sect;
            absentFile << "\t" <<  Rank;
            absentFile << "\t" << FName << endl;
        }
    }

    masterFile.close();
    absentFile.close();
    showOnScreen("absent.rpt");
}
```

```
/*****************************************************************************
 *                                                                           *
 * Name:          compRecord                                                 *
 *                                                                           *
 * Description:   Compares the master student database file with the         *
 *                attendance file. Returns true is student was present,      *
*                 and false otherwise.                                       *
 *                                                                           *
 * Input:         cardkey.txt and student record from the master file        *
 * Output:        None                                                       *
 *                                                                           *
 *****************************************************************************/


int Card::compRecord()
{
    inFile.open("cardkey.txt", ios::in);
    int found = 0;
    char SSN[15];

    while (inFile.getline(SSN, 14))
    {
        if (!strcmp(SSN, tempSSN))
        {
            found = 1;
            break;
        }
    }
    inFile.close();
    return found;
}
```

```
/*************************************************************************
*                                                                        *
* Name:          parseRecord                                             *
*                                                                        *
* Description:   Takes the raw student master file as an input and turns *
*                it into a usable format for the demonstration program.  *
*                                                                        *
* Input:         sglfile.db                                              *
* Output:        master.txt                                              *
*                                                                        *
*************************************************************************/

void Card::parseRecord(char inFile[], char outFile[])
{
    ifstream textfile;
    ofstream writefile;
    char filestring[51];
    textfile.open (inFile, ios::in);
    writefile.open(outFile, ios::out);

    while (textfile.getline(filestring,50))
    {
        int vectCt;

        for (vectCt=0; vectCt<=50; vectCt++)
        {
            if ((filestring[vectCt]=='%')||(filestring[vectCt]=='/') ||
                    (filestring[vectCt]== ' '))
                writefile << "\n";
            else
                if (filestring[vectCt]=='?')
                    break;
                else
                    writefile << filestring[vectCt];
        }
    }

    textfile.close();
    writefile.close();
}
```

```
/*************************************************************************
*                                                                        *
* Name:          getTotalRecord                                          *
*                                                                        *
* Description:   Counts the total number of records contained in the master *
*                student database.                                       *
*                                                                        *
* Input:         None.                                                   *
* Output:        None                                                    *
*                                                                        *
*************************************************************************/

void Card::getTotalRecord()
{
    readFile.open("sglfile.db", ios::in);
    totalRec = 0;

    while (readFile.getline(name, 60))
        totalRec += 1;

    readFile.close();
}

/*************************************************************************
*                                                                        *
* Name:          getDate                                                 *
*                                                                        *
* Description:   Provides the system's date to be used for the reports   *
*                generated by the demonstration program.                 *
*                                                                        *
* Input:         None.                                                   *
* Output:        None                                                    *
*                                                                        *
*************************************************************************/
void Card::getDate()
{
    struct dosdate_t d;
    _dos_getdate(&d);
    day = d.day;
    month = d.month;
    year = d.year;
}
```

```
/*************************************************************************
 *     MAGNETIC STRIPE TECHNOLOGY DEMONSTRATION PROGRAM                  *
 *                        Naval Postgraduate School                      *
 *                          Monterey, California                         *
 *                                                                       *
 * Module:     NPSCARD.H                                                 *
 *                                                                       *
 * Description:      The following module contains the class declaration for the *
 *                   NPS magnetic stripe demonstration program.  It includes *
 *                   declaration of member data and member functions.     *
 *                                                                       *
 * History:    07 July 1994 (Created)                                    *
 *                                                                       *
 * Programmer:  LT Rudy G. Dollete, USN   Internet: dollete@aol.com      *
 *                                                                       *
 *************************************************************************/

#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <fstream.h>
#include <iomanip.h>
#include <dos.h>
#include <stdio.h>
#include <ctype.h>

class Card
{
     private:
          char name[41];
          char lname[8];
          char FName[35];
          char MI[2], SSN[9], Rank[5];
          char Dept[4], Sect[3], tempSSN[10];
          int day, month, year;
          int totalRec;
```

```
    public:

    Card();
    void startNpsDemo();
    void takeAttendance();
    void createAttendReport();
    void converToTextFile();
    void showMainMenu();
    void showOnScreen(char inFile[]);
    void printReport(char inFile[]);
    void getMastRecord();
    void getDate();
    void getTotalRecord();
    int compRecord();
    void showAbsentees();
    void parseRecord(char inFile[], char outFile[]);
};
```

```
/**************************************************************************
*      MAGNETIC STRIPE TECHNOLOGY DEMONSTRATIONPROGRAM                   *
*                        Naval Postgraduate School                        *
*                           Monterey, California                          *
*                                                                         *
* Module:      MESSAGE.CPP                                                *
*                                                                         *
* Description:       The following module contains the class declaration for the  *
*                    NPS magnetic stripe demonstration program.  It includes  *
*                    declaration of member data and member functions.     *
*                                                                         *
* History:     07 July 1994  (Created)                                    *
*                                                                         *
* Programmer:  LT Rudy G. Dollete, USN    Internet: dollete@aol.com        *
*                                                                         *
**************************************************************************/
#include “message.h”

char *versionMsg[20] ={
      “\n”,
      “\t   ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»                ”,
      “\t   º                                                   º”,
      “\t   º               SUPERINTENDENT’S                    º”,
      “\t   º      GUEST LECTURE ATTENDANCE PROGRAM             º”,
      “\t   º                  -------                          º”,
      “\t   º                                                   º”,
      “\t   º                 Version 1.0                       º”,
      “\t   º                                                   º”,
      “\t   º    A Demonstration of Magnetic Stripe Technology.  º”,
      “\t   º                                                   º”,
      “\t   º              LT RUDY G. DOLLETE                   º”,
      “\t   º                (programmer)                       º”,
      “\t   º                                                   º”,
      “\t   º                ROGER STEMP                        º”,
      “\t   º                 (adviser)                         º”,
      “\t   º                                                   º”,
      “\t   ÈÍÍÍÍÍÍÍÍÍ * <Press any key to continue> * ÍÍÍÍÍÍÍÍÍ¿         ”,
      “\t ÛÛÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿°°°°°°°°¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ  ”,
      “\t                 “};
```

```
char *mainMenuMsg[12] ={
          "\n\n\n",
   "\t  Éííííííííí * NPS CARD TRANSACTION MENU * íííííííí»            ",
   "\t  º                                                            º",
   "\t  º (1) Take Attendance      (4) Print Absentees               º",
   "\t  º                                                            º",
   "\t  º (2) Show Absentees       (5) Print Attendees               º",
   "\t  º                                                            º",
   "\t  º (3) Show Attendees       (6) Quit Application              º",
   "\t  º                                                            º",
   "\t  Èííííííííííííí * Enter Selection * íííííííííííííí¿            ",
   "\t ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿°°°°°°°°¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛ      ",
   "\t                   "};


char *swipeUserCardMsg[10] = {
   "\n\n\n",
   "\t  Éíííííííííííí ****** WARNING ****** íííííííííííí»            ",
   "\t  º                                                           º",
   "\t  º  ONLY AUTHORIZED USERS MAY USE THIS PROGRAM.              º",
   "\t  º                                                           º",
   "\t  º     Please Swipe Your User's Card now.                    º",
   "\t  º                                                           º",
   "\t  Èíííííííííííííííííííííííííííííííííííííííííííííí¿            ",
   "\t  ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿°°°°°°°°¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛ     ",
   "\n\n\n\t    "};


char *swipeStudCardMsg[11] = {
   "\n\n\n",
   "\t  Éíííííííííííí * Taking Attendance * ííííííííííííí»           ",
   "\t  º                                                           º",
   "\t  º       Tracking Program Initialized.                       º",
   "\t  º                                                           º",
   "\t  º  Please Swipe Your Student Card At This Time.             º",
   "\t  º                                                           º",
   "\t  º          ® Press <ESC> to Quit ¯                          º",
   "\t  Èíííííííííííííííííííííííííííííííííííííííííííííí¿            ",
   "\t  ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿°°°°°°°°¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛ     ",
   "\n\n\n\t    "};
```

```
char *invalidCardMsg[10] = {
    "\n\n\n",
    "\t  Éffffffffffffff * USER VALIDATION * ffffffffffffff»           ",
    "\t  º                                                              º",
    "\t  º     SORRY, YOUR USER'S CARD IS NOT VALID.                    º",
    "\t  º                                                              º",
    "\t  º        <PRESS ANY KEY TO CONTINUE>                           º",
    "\t  º                                                              º",
    "\t  Èffffffffffffffffffffffffffffffffffffffffffffffff¿             ",
    "\t  ÛÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿********¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛÛ  ",
    "\n\n\n\t                    "};

char *invalidSelMsg[1] = {"\n                    Your selection is not valid.\n"};

char *enterSelMsg[1] = {"\n\n\t\t\t\tEnter your selection: "};

char *pressAnyKeyMsg[1] = {"\n                    Press any key to continue."};

char *attendMsg[2] = {
    "THE FOLLOWING PERSONNEL ATTENDED TODAY'S LECTURE",
                    "---------------------------------------------"};

char *absentMsg[2] = {
    "THE FOLLOWING PERSONNEL WERE ABSENT IN TODAY'S LECTURE",
                    "--------------------------------------------------"};

char *fieldMsg[2] = {
                    "  SSN\t\tSECT\tCODE\tRANK\t\tNAME",
                    "  ---\t\t----\t----\t----\t\t----"};
```

```
/*****************************************************************************
*      MAGNETIC STRIPE TECHNOLOGY DEMONSTRATION PROGRAM                     *
*                          Naval Postgraduate School                        *
*                            Monterey, California                           *
*                                                                           *
* Module:          MESSAGE.H                                                *
*                                                                           *
* Description:     The following module contains the declarations for the NPS *
*                  Magnetic Stripe demonstration program display messages.  *
*                                                                           *
* History:         15 July 1994  (Created)                                  *
*                                                                           *
* Programmer:      LT Rudy G. Dollete, USN    Internet: dollete@aol.com     *
*                                                                           *
*****************************************************************************/

    extern char *versionMsg[20];
    extern char *mainMenuMsg[12];
    extern char *swipeUserCardMsg[10];
    extern char *swipeStudCardMsg[11];
    extern char *invalidCardMsg[10];
    extern char *enterSelMsg[1];
    extern char *invalidSelMsg[1];
    extern char *pressAnyKeyMsg[1];
    extern char *attendMsg[2];
    extern char *absentMsg[2];
    extern char *fieldMsg[2];
```

```
1: ************************************************************************
2: *****              PROGRAM NAME: STUDENT CARD ENCODER/DECODER        *
3: *****                            Version 1.0                         *
4: *****                                                                *
5: ***** Programmer: RUDY G. DOLLETE, USN   Adviser: ROGER STEMP       *
6: *****                                                                *
7: ***** DESCRIPTION:This program has been designed to support the     *
8: *****     SGL magstripe technology application demonstration. The    *
9: *****     program offers the user only two options: Encode or        *
10: *****    Decode a magnetic stripe card.  Encoding and decoding      *
11: *****    are done on the card's track 1 only in order to make       *
12: *****    this first version easy to understand and implement.       *
13: *****                                                               *
14: ***** VERSION UPGRADES: Future upgrades will include encoding and  *
15: *****          decoding capabilities for all tracks (1,2,3).        *
16: *****                                                               *
17: ***** FUTURE THESIS WORK: Students interested in this field may     *
18: *****          take this program and add the "Host Comm" capability *
19: *****          which will allow a user to download the data to be   *
20: *****          encoded from a PC instead of manually entering       *
21: *****          them using the keyboard. The "Host Comm" feature     *
22: *****          may also be programmed so that the data read from    *
23: *****          a card are transferred back to a PC either in        *
24: *****          batch or immediately as each card is decoded.        *
25: *****                                                               *
26: *****      Send questions/suggestions to: dollete@aol.com           *
27: ************************************************************************
28: *****
29: Create STUDENT_DATA. Alphanumeric. Variable. Length: 40.   ""
30: Create IATA. Alphanumeric. Variable. Length: 76.   ""
31: Create ATTEMP_COUNT. Numeric. Variable. Length: 1.   ""
32: Create RESPONSE. Alphanumeric. Variable. Length: 1.   ""
33: Create RESPONSE1. Alphanumeric. Initialized. Length: 2.  Reset
    for the next job. "12"
34: Prompt "      NPS MAGSTRIPE ENCODING PROGRAM"  (on theDisplay).
    Line number: 1
35: Prompt "  Programmer: LT Rudy G. Dollete, USN"  (on the Display).
    Line number: 2
36: Pause for 3 seconds
37: Prompt "   MAGNETIC STRIPE TECHNOLOGY AT NPS"   (on the Display).
    Line number: 1
38: Prompt "                Version 1.0"   (on the Display).
    Line number: 2
39: Pause for 2 seconds
40: Prompt "    This Program Encodes Or Decodes "   (on the Display).
    Line number: 1
41: Prompt "      Student Data On Track 1 Only."   (on the Display).
    Line number: 2
42: Pause for 3 seconds
43: *****
44: *****
```

```
45: Label MAIN
46: Set ATTEMP_COUNT starting at 1 of length 1 to "3"
47: Prompt "(1) ENCODE     (2) DECODE     (ESC) EXIT"
    (on the Display). Line number: 1
48: Clear line 2
49: Keyin RESPONSE starting at 1 of length 1. Row 2. Col 20.
    Enable/disable <ESC>: <ESC> enabled.  Case conversion: None.
    On <ESC> goto label:
     PROGRAM_EXIT
50: Compare RESPONSE[1] to RESPONSE1[1] for length 1.
51: On error goto CHECK_CHOICE_TWO
52: Goto START_ENCODE
53: *****
54: *****
55: Label CHECK_CHOICE_TWO
56: Compare RESPONSE[1] to RESPONSE1[2] for length 1.
57: On error goto MAIN
58: Goto START_DECODE
59: *****
60: *****
61: Label START_ENCODE
62: Prompt "SSN/LNAME, FNAME MI./RANK/CURR/OFFCODE?"
    (on the Display). Line number: 1
63: Prompt " Enter Track 1 Data Using Format Above"
    (on the Display). Line number: 2
64: Pause for 3 seconds
65: Clear line 2
66: Keyin STUDENT_DATA starting at 1 of length 40. Row 2. Col 1.
     Enable/disable <ESC>: <ESC> disabled.  Case conversion: To upper.
67: Display ÿvÿvúûPÄ^ü&ÿw$&ÿw"&ÿw&&ÿwSB&ÿ6m&ÿ6mPH
           /ÄøûPh@Fúå]ËUì, starting at -29101 of
           length 14079. Row = 16902. Col = 9873
68: Copy from STUDENT_DATA at 1 to IATA at 1 of length 40
69: Clear line 1
70: Prompt "Insert card to be encoded into the 160"
    (on the Display). Line number: 1
71: Encode Track Field
                1 IATA
    Keep or release card: Release
72: On error goto ENCODE_ERROR
73: Prompt "       Card Encoding Successful!"
    (on the Display). Line number: 2
74: Pause for 2 seconds
75: Goto MAIN
76: *****
77: *****
78: Label ENCODE_ERROR
79: Prompt "     Unable to encode magstripe..."
    (on the Display). Line number: 2
80: Pause for 2 seconds
81: Decrement ATTEMP_COUNT
```

```
 82: If ATTEMP_COUNT equals 0 goto ENCODE_FAILURE
 83: Clear line 2
 84: Prompt "Insert card and try again...."   (on the Display).
     Line number: 2
 85: Pause for 2 seconds
 86: G to START_ENCODE
 87: *****
 88: *****
 89: Label ENCODE_FAILURE
 90: Prompt "          Encoding Failure!"   (on the Display).
     Line number: 1
 91: Prompt "     Get a new card, and try again."   (on the Display).
     Line number: 2
 92: Pause for 3 seconds
 93: Goto MAIN
 94: *****
 95: *****
 96: Label START_DECODE
 97: Set ATTEMP_COUNT starting at 1 of length 1 to "3"
 98: Prompt "Insert card to be read into the 160..."   (on the Display).
     Line number: 1
 99: Read   Track Field
                1 IATA
     Keep or release card: Release
100: On error goto DECODE_ERROR
101: Prompt "       Card decoding successful!"   (on the Display).
     Line number: 2
102: Pause for 2 seconds
103: Copy from IATA at 1 to STUDENT_DATA at 1 of length 40
104: Display ÿvÿvúûPÅ^ù&ÿw$&ÿw"&ÿw&&ÿw
      SB&ÿ6m&ÿ6mPH/ÀøûPh@FúåJËUì, starting at -29101 of
      length 14079. Row = 16902. Col = 9873
105: Prompt "    Press enter to continue..."   (on the Display).
     Line number: 2
106: Keyin RESPONSE starting at 1 of length 1. Row 2. Col 30.
     Enable/disable <ESC>: <ESC> enabled.  Case conversion: To upper.
     On <ESC> goto label:
      MAIN
107: Goto MAIN
108: *****
109: *****
110: Label DECODE_ERROR
111: Prompt "    Unable to decode magstripe..."   (on the Display).
     Line number: 2
112: Pause for 2 seconds
113: Decrement ATTEMP_COUNT
114: If ATTEMP_COUNT equals 0 goto DECODE_FAILURE
115: Clear line 2
116: Prompt "Insert card and try again...."   (on the Display).
     Line number: 2
117: Pause for 2 seconds
```

```
118: Goto START_DECODE
119: *****
120: *****
121: Label DECODE_FAILURE
122: Prompt "            Decoding Failure!"   (on the Display).
     Line number: 1
123: Prompt "     Get a new card, and try again."   (on the Display).
     Line number: 2
124: Pause for 3 seconds
125: Goto MAIN
126: *****
127: *****
128: Label PROGRAM_EXIT
129: Prompt "THANKS FOR USING THE MAGSTRIPE PROGRAM"   (on the Display).
     Line number: 1
130: Prompt "   Send questions to: dollete@aol.com"   (on the Display).
     Line number: 2
131: Pause for 5 seconds
132: Exit
133:
```

APPENDIX B. Smart Card Program Codes

```
/*****************************************************************************
 *          SMART CARD TECHNOLOGY DEMONSTRATION PROGRAM                      *
 *                        Naval Postgraduate School                          *
 *                        Monterey, California                               *
 *                                                                           *
 * Module:          SMARTCRD.C                                               *
 *                                                                           *
 * Description:     The following module contains the demonstration routines *
 *                  for the NPS smart card technology demonstration program. *
 *                                                                           *
 * History:         15 July 1994  (Created)                                  *
 *                                                                           *
 * Programmer:      LT Rudy G. Dollete, USN   Internet: dollete@aol.com      *
 *                                                                           *
 *****************************************************************************/

#include "messages.h"
#include "smartcrd.h"
int main(void)
{
      short nError;
      nError = InitializeCommPort();

      if (nError != NO_ERROR)          /* this is backward... */
          return (0);
      else
      {
          ShowVersionMsg();
          ShowMainMenu();
      }

 return (0);
}
```

```
/*************************************************************************
 *                                                                       *
 * Function Name:   nWaitForCard()                                       *
 *                                                                       *
 * Description:     This function waits for a card to be inserted into the series 50  *
 *                  smart card reader unit before proceeding. The series 50 card      *
 *                  reader is also referred to as the Card Acceptor Unit (CAU).       *
 *                                                                       *
 * History:         15 July 1994  (Created)                              *
 *                                                                       *
 *************************************************************************/

short nWaitForCard ()
{
    short nError;
    printf("\n\t\t\t%s\n", insertCardMsg[0]);
    printf("\n\t\t\t\t%s\n\n", pressEscMsg[0]);
    printf("\t\t%s", cursorMsg[0]);

    while (NUM)
    {
        nError = DCOS_Slct_CAU (NUM);
        if (nError != ERRNCHIP)
            break;
        if (kbhit ())
        {
            if (getch () == ESC_KEY)
            {
                nError = -1;
                break;
            }
        }
    }

    return (nError);
}
```

```
/*****************************************************************
 * Function Name:     CreateFileDef ( BYTE FileNum, BYTE Mode,   *
 *                                    BYTE Security, BYTE RecNum,*
 *                                    BYTE RecLen, BYTE *FileName)*
 *                                                               *
 * Description:       This function creates the file definition on the smart card with *
 *                    all the parameters specified by the programmer.                  *
 *                                                               *
 * History:           15 July 1994  (Created)                    *
 *                                                               *
 *****************************************************************/

int CreateFileDef ( BYTE FileNum, BYTE Mode,
                    BYTE Security, BYTE RecNum,
                    BYTE RecLen, BYTE *FileName)
{
    int Status;
    BYTE FileDef[16];

    memset(FileDef, NUL, 16);
    FileAddress -= (RecNum * RecLen);
    FileDef[0] = (FileAddress >> 8) & 0x00FF;
    FileDef[1] = FileAddress & 0x00FF;
    FileDef[2] = Security;
    FileDef[5] = RecLen;
    FileDef[6] = RecNum;
    FileDef[7] = Mode;
    FileDef[8] = *FileName++;
    FileDef[9] = *FileName;

    Status = DCOS_WrtFlDef (FileNum, FileDef, 16);
    return Status;
}
```

```
/*************************************************************************
 *                                                                       *
 * Function Name:     IssueStudentCard()                                 *
 *                                                                       *
 * Description:       The following module formats the smart card and prepares it *
 *                    for three user applications. These applications are: electronic *
 *                    purse, check in/out, and student ID card.          *
 *                                                                       *
 * History:           15 July 1994 (Created)                             *
 *                                                                       *
 *************************************************************************/

void IssueStudentCard()
{
    int  i, Key, Status;
    short RtrnLen;
    DWORD *Balance;
    WORD *CheckStatus;
    BYTE  KeyLen, PINKey[9];
    BYTE  TempBuff[32];
    TransActInfo *Transaction;

    Status = nWaitForCard();
    if (Status > 0)
    {
        memset(PINKey, ' ', 9);
        printf("\n\t\tEnter the Personal ID Number you wish to use: ");
        scanf("%8s", PINKey);
        KeyLen = strlen(PINKey);
        memset(FileData, NUL, 512);
        Status = DCOS_ResetChp(TempBuff, &RtrnLen);

        if (Status == NO_ERROR)
        {
            FileAddress = ((TempBuff[2] - '0') * 10) + (TempBuff[3] - '0');
            FileAddress = ((FileAddress * 1024) / 8) - 132;
            Status = DCOS_SubmitKy(1, "DATACARD", 8);
            DCOS_EraseChp();
            DCOS_WriteKey(2, PINKey, KeyLen);
            DCOS_SubmitKy(2, PINKey, KeyLen);
            CreateUserID();
            CreateFileDef(1, 1, 32, 1, 4, "PB");
```

```
            Balance = (DWORD *) FileData;
            *Balance = 20000L;
            SaveFileData("AB", 1);
            CreateFileDef(2, 2, 0, 10, 8, "AT");
            Transaction = (TransActInfo *) FileData;
            Transaction ->Type = 1;
            Transaction ->Amount = 20000L;
            SaveTransAction(Transaction);
            CreateCheckInFile();
            printf("\n\t\t\t      Card Format Complete!\n");
            printf("\t\t\t   <Press any key to continue>");
            getch();
            EjectCard();
        }
    }
    ShowAdminMenu();
}
```

```
/*************************************************************************
 * Function Name:     CreateUserID()                                     *
 *                                                                       *
 * Description:       The following module contains the routine for creating the *
 *                    ID card demonstration program.                     *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *************************************************************************/
void CreateUserID()
{
    ID_Info *Identification;
    BYTE TempName[30];
    CreateFileDef(0, 1, 0, 1, 64, "ID");
    Identification = (ID_Info *) FileData;
    printf("\n\tEnter your last name: ");
    scanf("%s", TempName);
    strcpy(Identification->LName, TempName);
    printf("\n\tEnter your first name: ");
    scanf("%s", TempName);
    strcpy(Identification->FName, TempName);
    printf("\n\tEnter your middle initial: ");
    scanf("%s", TempName);
    strcpy(Identification->MI, TempName);
    printf("\n\tEnter your SSN (no spaces): ");
    scanf("%s", TempName);
    strcpy(Identification->SSN, TempName);
    printf("\n\tEnter your rank (LT, CAPT, etc): ");
    scanf("%s", TempName);
    strcpy(Identification->Rank, TempName);
    printf("\n\tEnter your curriculum number: ");
    scanf("%s", TempName);
    strcpy(Identification->Curric, TempName);
    printf("\n\tEnter your curriculum office code: ");
    scanf("%s", TempName);
    strcpy(Identification->CurrCode, TempName);
    printf("\n\tEnter your birth date (i.e. 053078): ");
    scanf("%s", TempName);
    strcpy(Identification->BDate, TempName);
    printf("\n\tEnter your local phone number (3930107): ");
    scanf("%s", TempName);
    strcpy(Identification->TelNum, TempName);
    SaveFileData("ID", 1);
}
```

```
/*************************************************************************
*                                                                        *
* Function Name:     CreateCheckInFile()                                 *
*                                                                        *
* Description:       The following module contains the routine for initializing the *
*                    check in/out demonstration program. The following values *
*                    identify the status of the student's check in/out process. *
*                                                                        *
*                    Initial issue = 0                                   *
*                    Checked In = 1                                      *
*                    Checked Out = 2                                     *
*                                                                        *
* History:           15 July 1994  (Created)                             *
*                                                                        *
*************************************************************************/

void CreateCheckInFile()
{
    CheckInInfo *CheckStatus;
    CreateFileDef(3, 1, 0, 1, 20, "CI");
    CheckStatus = (CheckInInfo *) FileData;
    CheckStatus -> CurrOff = 0;
    CheckStatus -> ServRep = 0;
    CheckStatus -> StuMailCen = 0;
    CheckStatus -> FamServCen = 0;
    CheckStatus -> ClasMatCtrl = 0;
    CheckStatus -> SCIF = 0;
    CheckStatus -> Library = 0;
    CheckStatus -> SecMgr = 0;
    CheckStatus -> BOQ = 0;
    CheckStatus -> Dental = 0;
    CheckStatus -> PSD = 0;
    CheckStatus -> NEX = 0;
    CheckStatus -> LicExam = 0;
    CheckStatus -> Registrar = 0;
    CheckStatus -> CompCtr = 0;
    CheckStatus -> BaseSec = 0;
    CheckStatus -> Medical = 0;
    CheckStatus -> HRO = 0;
    CheckStatus -> Admin = 0;
    SaveFileData ("CI", 1);
}
```

```
/*****************************************************************
 *                                                               *
 * Function Name:     SaveFileData (BYTE *FileName, BYTE FileType)  *
 *                                                               *
 * Description:       The following module contains the routine that searches for a *
 *                    file by filename given by the user and if found saves the *
 *                    contents of the buffer on the smart card under the given *
 *                    filename.                                  *
 *                                                               *
 * History:           15 July 1994  (Created)                    *
 *                                                               *
 *****************************************************************/
int SaveFileData (BYTE *FileName, BYTE FileType)
{
    int   i, Status;
    BYTE  FileNum, MaxRecs, RecLen;
    BYTE  *Record;
    BYTE  FileDef[MAX_FILE_DEF_LEN];

    Record = FileData;
    Status = DCOS_FindFile (FileName, &FileNum, FileDef);

    if (Status == NO_ERROR)
    {
        RecLen = FileDef[5];
        MaxRecs = FileDef[6];

        if (FileType == 0)              /* Sequential Record Access. */
            Status = DCOS_WrtSeqRc(FileNum, Record, RecLen);
        else                            /* Random Record Access. */
            for (i=0; ((i < MaxRecs) && (Status == NO_ERROR)); ++i)
            {
            Status = DCOS_WrtRdmRc(FileNum, i, Record, RecLen);
            Record += RecLen;
            }
    }
    return (Status);
}
```

```
/*****************************************************************************
 *                                                                           *
 * Function Name:     SaveTransAction(TransActInfo *Transaction)             *
 *                                                                           *
 * Description:       The following module contains the routine that gets the current *
 *                    date from the PC and saves the transaction data on the smart   *
 *                    card.  The transaction data will serve as the audit trail for all *
 *                    electronic purse transactions.                         *
 *                                                                           *
 * History:           15 July 1994  (Created)                                *
 *                                                                           *
 *****************************************************************************/

int SaveTransAction(TransActInfo *Transaction)
{
    int  Status;
    struct tm *curtime;
    time_t bintime;
    time (&bintime);
    curtime = localtime (&bintime);
    Transaction->Date[0] = curtime->tm_mon + 1;
    Transaction->Date[1] = curtime->tm_mday;
    Transaction->Date[2] = curtime->tm_year;
    Status = SaveFileData ("AT", 0);              //zero means sequential write
    return (Status);
}
```

```
/*************************************************************************
 *                                                                       *
 * Function Name:     ShowVersionMsg()                                   *
 *                                                                       *
 * Description:       The following routine shows the initial screen of the NPS  *
 *                    demonstration program.  It includes the name of the program,  *
 *                    the version number, and the name of the programmer.  *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/
void ShowVersionMsg()
{
    int i;
    clrscr();

    for (i=0; i < 19; i++)
        printf("%s\n", versionMsg[i]);

    printf("%s", versionMsg[19]);
    getch();
}
```

```
/*************************************************************************
 *                                                                       *
 * Function Name:     ShowMainMenu()                                     *
 *                                                                       *
 * Description:       The following routine displays the main menu selection for  *
 *                    the NPS smart card technology demonstration program.       *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/

void ShowMainMenu()
{
    int i; char selection;
    clrscr();
    for (i =0; i < 15; i++)
    printf("%s\n", mainMenuMsg[i]);
    printf("%s", mainMenuMsg[15]);
    selection = getch();

    switch (selection)
    {
        case ('1'): {ShowPurseMenu();
                     break;}
        case ('2'): {ShowCheckInOutMenu();
                     break;}
        case ('3'): {ShowAdminMenu();
                     break;}
        case ('4'): {DCOS_EjectCAU();
                     ShowMainMenu();
                     break;}
        case ('5'): {ShowQuitMsg();
                     break;}
        default:   {printf("\n\t\t\t   %s\n", invalidSelMsg[0]);
                    printf("\t\t\t   %s\n", pressAnyKeyMsg[0]);
                    getch();
                    ShowMainMenu();
                    break;}
    }
}
```

```
/*****************************************************************
 *                                                               *
 * Function Name:     InitializeCommPort()                       *
 *                                                               *
 * Description:       The following routine initializes the communication port  *
 *                    between the Series 50 and the host computer.  *
 *                                                               *
 * History:           15 July 1994  (Created)                    *
 *                                                               *
 *****************************************************************/
short InitializeCommPort()
{
    short nError;
    BYTE cardResetInfo[17];    /* reponse to reset array          */
    WORD cardBufLength;        /* length of the reset information */

    EjectCard();
    nError = DCOS_InitCommWithCid(COM1_PORT,
                BPS_9600 | DATA_SZ_8 | TWO_STOP_BITS | NO_PARITY);

    if (nError == NO_ERROR)
    {
        printf("\n\n\t\tCommunication port <COMM1> initialized.\n");
        DCOS_ResetChp(cardResetInfo, &cardBufLength);
    }
    else
    {
        printf("\n\n\t\tCommunication port intializing error.\n");
        printf("\t\tCheck all connections, and try again.\n");
    }
    return (nError);
}
```

```
/*************************************************************************
 * Function Name:     ShowPurseMenu()                                    *
 *                                                                       *
 * Description:       The following routine displays the electronic purse selection  *
 *                    menu for the NPS smart card  demonstration program.  *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/
void ShowPurseMenu()
{
    int k; char selection;
    clrscr();

    for (k=0; k<15; k++)
        printf("%s\n", purseMenuMsg[k]);

    printf("%s", purseMenuMsg[15]);
    selection = getch();

    switch (selection)
    {
        case ('1'): {PurchaseAmount();
                     break;}
        case ('2'): {DepositAmount();
                     break;}
        case ('3'): {ViewBalance();
                     getch();
                     EjectCard();
                     break;}
        case ('4'): {ViewTransAction();
                     getch();
                     EjectCard();
                     break;}
        case ('5'): {break;}
        default:    {printf("\n\t\t\t   %s\n", invalidSelMsg[0]);
                     printf("\t\t\t   %s\n", pressAnyKeyMsg[0]);
                     getch();
                     break;}
    }

    ShowMainMenu();
}
```

```
/*****************************************************************
 *                                                               *
 * Function Name:    PurchaseAmount()                            *
 *                                                               *
 * Description:      The following routine handles the debit portion of the  *
 *                   electronic purse demonstration program.     *
 *                                                               *
 * History:          15 July 1994  (Created)                     *
 *                                                               *
 *****************************************************************/

void PurchaseAmount()
{
    int Status;
    DWORD *Balance; long Cost;
    BYTE Keylen, PINKey[9];
    char string[25], endptr;
    long TempBal;
    TransActInfo *TransAction;

    TransAction = (DWORD *) FiieData;
    Balance = (DWORD *) FileData;
    nWaitForCard();
    memset(PINKey, ' ', 9);
    printf("\n\t\t\tPlease Enter your PIN > ");
    scanf("%s", PINKey);
    Keylen = strlen(PINKey);
    ViewBalance();
    Status = DCOS_SubmitKy(2, PINKey, Keylen);

    if (Status == NO_ERROR)
    {
        GetFileData("AB");
        itoa(*Balance, string, 10);
        Cost = GetAmount(1);
        TempBal = strtol(string, &endptr, 10);

        if (Cost > TempBal)
        {
            printf("\n\t\t\tSorry, purchase amount exceeds current balance.");
            printf("\n\n\t\t\t<Press any key to continue>");
        }
```

```
        else
          {
                *Balance = *Balance - Cost;
                SaveFileData("AB", 1);
                TransAction -> Type = 0;
                TransAction -> Amount = Cost;
                SaveTransAction(TransAction);
                ViewBalance();
          }

    }
    else
          printf("\n\n\t\t<Your PIN is not valid. Press ESC to continue>");

      getch();
      EjectCard();
}
```

```
/*************************************************************************
 *                                                                       *
 * Function Name:     GetAmount(int DspFlag)                             *
 *                                                                       *
 * Description:       The following routine queries the user to enter the amount  *
 *                    of purchase or deposit for the electronic purse demonstration *
 *                    program.                                           *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *                                                                       *
 *************************************************************************/

long GetAmount(int DspFlag)
{
    char input[25], temp[25], *endptr;
    long value;
    int InputLen, k;

    if (DspFlag == 1)
        printf("\n\t\t\tPlease enter the amount of purchase> $ ");
    else
        printf("\n\t\t\tPlease enter the amount of deposit> $ ");

    scanf("%s", input);
    InputLen = strlen(input);
    InputLen -=3;
    memset(temp, ' ', 25);

    for (k=0; k<InputLen; k++)
        temp[k] = input[k];

    temp[InputLen] = input[InputLen+1];
    temp[InputLen+1] = input[InputLen+2];
    value = strtol(temp, &endptr, 10);
    return (value);
}
```

```
/*****************************************************************
 * Function Name:     DepositAmount()                            *
 *                                                               *
 * Description:       The following routine handles the credit portion of the  *
 *                    electronic purse demonstration program.    *
 *                                                               *
 * History:           15 July 1994  (Created)                    *
 *                                                               *
 *****************************************************************/
void DepositAmount()
{
    int Status;
    DWORD *Balance, Deposit;
    BYTE Keylen, PINKey[9];
    char string[25];
    TransActInfo *TransAction;
    TransAction = (DWORD *) FileData;
    Balance = (DWORD *) FileData;
    memset(PINKey, ' ', 9);

    printf("\n\t\t\tPlease Enter your PIN > ");
    scanf("%s", PINKey);
    Keylen = strlen(PINKey);
    ViewBalance();
    Status = DCOS_SubmitKy(2, PINKey, Keylen);

    if (Status == NO_ERROR)
    {
        Deposit = GetAmount(2);
        *Balance = *Balance + Deposit;
        SaveFileData("AB", 1);
        TransAction -> Type = 1;
        TransAction -> Amount = Deposit;
        SaveTran(TransAction);
        ViewBalance();
    }
    else
        printf("\n\n\t\t<Your PIN is not valid. Press ESC to continue>");

    getch();
    EjectCard();
}
```

```
/*****************************************************************************
 * Function Name:     ViewBalance()                                          *
 *                                                                           *
 * Description:       The following routine searches and obtains the current  *
 *                    balance of the electronic purse and displays it on the screen. *
 *                    The balance is converted from integer to string.       *
 *                                                                           *
 * History:           15 July 1994  (Created)                                *
 *                                                                           *
 *****************************************************************************/
void ViewBalance()
{
    int i, k, Status;
    BYTE ResetInfo[17];
    WORD ResetBuffLen;
    DWORD *Balance;
    char string[25];
    Balance = (DWORD *) FileData;
    Status = DCOS_Slct_CAU (1);
    if (Status != NO_ERROR)
        nWaitForCard();
    DCOS_ResetChp(ResetInfo, &ResetBuffLen);
    GetFileData("AB");

    itoa(*Balance, string, 10);
    i = strlen(string);
    printf("\n\t\t\tYour Current Balance is: $ ");
    if (i==1)
        printf("0.0%c", string[0]);
    else
    {
        i -=2;
        printf("%d", *Balance/100);
        printf(".%c%c\n", string[i], string[i+1]);
    }
}
```

```
/*************************************************************************
 * Function Name:   ViewTransAction()                                    *
 *                                                                       *
 * Description:     The following routine displays the contents of the transaction  *
 *                  file for audit trail viewing of the electronic purse demonstration*
 *                  program.                                             *
 *                                                                       *
 * History:         15 July 1994 (Created)                               *
 *                                                                       *
 *************************************************************************/

void ViewTransAction()
{
    int Record, Status, i, k;
    short RtrnLen;
    BYTE RecCtr, RecLen;
    BYTE FileNum, EndOfFile;
    BYTE TempData[3], TempBuff[32];
    char string[20], temp[20];
    BYTE FileDef[MAX_FILE_DEF_LEN];
    int Data_WRef;

    TransActInfo  *TransAction;
    TransAction = (TransActInfo *) FileData;
    memset(TempData, NUL, 3);
    nWaitForCard();
    Status = DCOS_ResetChp(TempBuff, &RtrnLen);
    Status = DCOS_FindFile("AT", &FileNum, FileDef);

    if (Status == NO_ERROR)
    {
        RecLen = FileDef[5];
        EndOfFile   = FileDef[11];

        if (EndOfFile == 0)
            Status = ICERFEMP;
        else
        {
            Record = FileDef[6] - 1;
            clrscr();
            printf("\n%s", transActMsg[0]);
            printf("%s", transActMsg[1]);
```

```
            do {
                Status = DCOS_Rd_RdmRc(FileNum, Record, (BYTE *) TransAction,
                RecLen, (short *) &RtrnLen);

                if ((Status == NO_ERROR) && (TransAction->Amount != -1))
                {
                    printf("\n\t\t%d/%d/%d", TransAction -> Date[0],
                                             TransAction -> Date[1],
                                             TransAction -> Date[2]);
                    itoa(TransAction->Amount, string, 10);
                    i = strlen(string);
                    i -=2;
                    printf("\t\t\t$%5d", TransAction->Amount/100);
                    printf(".%c%c", string[i], string[i+1]);

                    if (TransAction->Type == 0)
                        printf("\t\tDebit");
                    else
                        printf("\t\tCredit");
                }
            --Record;
            } while ((Record >= 0) && (Status == NO_ERROR));
        }
    }
    printf("\n\n%s\n", transActMsg[1]);
    printf("\t\t\t%s\n", pressAnyKeyMsg[0]);
    printf("\t\t%s", cursorMsg[0]);
}
```

```
/*****************************************************************
 *                                                               *
 * Function Name:    GetFileData(BYTE *FileName)                 *
 *                                                               *
 * Description:      The following routine searches for a file by filename and if  *
 *                   found, reads the contents of the file into a buffer (FileData).  *
 *                                                               *
 * History:          15 July 1994  (Created)                     *
 *                                                               *
 *****************************************************************/

void GetFileData(BYTE *FileName)
{
    int i;
    short RtrnLen;
    BYTE FileNum, MaxRecs, RecLen;
    BYTE *Record;
    BYTE FileDef[MAX_FILE_DEF_LEN];

    Record = FileData;
    DCOS_FindFile(FileName, &FileNum, FileDef);
    RecLen = FileDef[5];
    MaxRecs = FileDef[6];

    for (i = 0; i < MaxRecs; ++i)
    {
        DCOS_Rd_RdmRc (FileNum, i, Record, RecLen, (short *) &RtrnLen);
        Record += RecLen;
    }
}
```

```
/*************************************************************************
 * Function Name:     ShowCheckInOutMenu()                               *
 *                                                                       *
 * Description:       The following routine displays the main selection menu for  *
 *                    the NPS check in/out demonstration program.        *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/
void ShowCheckInOutMenu()
{
    int k; char selection;
    clrscr();
    for (k=0; k<13; k++)
        printf("%s\n", checkInOutMsg[k]);

    printf("%s", checkInOutMsg[13]);
    selection = getch();

    switch (selection)
    {
        case ('1'): {ShowCheckSelect(selection);
                     break;}
        case ('2'): {ShowCheckSelect(selection);
                     break;}
        case ('3'): {VerifyCheckInStatus():
                     getch();
                     EjectCard();
                     break;}
        case ('4'): {break;}
        default:    {printf("\n\t\t\t   %s\n", invalidSelMsg[0]);
                     printf("\t\t\t   %s\n", pressAnyKeyMsg[0]);
                     getch();
                     break;}
    }

    ShowMainMenu();
}
```

```
/*********************************************************************
 * Function Name:     ShowCheckSelect(char selection)                *
 *                                                                   *
 * Description:       The following routine displays all the departments listed on *
 *                    the NPS check in/out sheet. The current status may be changed*
 *                    by selecting the department's appropriate selection number. *
 *                                                                   *
 * History:           15 July 1994 (Created)                         *
 *                                                                   *
 *********************************************************************/
void ShowCheckSelect(char selection)
{
    int k, Status, selection2, Invalid;
    BYTE ResetInfo[45];
    WORD ResetBuffLen;
    CheckInInfo *CheckInStatus;
    clrscr();
    for (k = 0; k < 24; k++)
        printf("%s\n", checkInMsg[k]);

    printf("%s", checkInMsg[24]);
    scanf("%d", &selection2);
    CheckInStatus = (CheckInInfo *) FileData;
    Status = DCOS_Slct_CAU(1);

    if (Status != NO_ERROR)
        nWaitForCard();
    clrscr();
    DCOS_ResetChp(ResetInfo, &ResetBuffLen);
    GetFileData("CI");

    switch (selection2)
    {
    case (1): { if (selection1 == '1')
                    CheckInStatus -> CurrOff = 1;
              else
                    CheckInStatus -> CurrOff = 2;
              break;}
    case (2): { if (selection1 == '1')
                    CheckInStatus -> ServRep = 1;
              else
                    CheckInStatus -> ServRep = 2;
              break;}
```

```
case (3): { if (selection1 == '1')
              CheckInStatus -> StuMailCen = 1;
          else
              CheckInStatus -> StuMailCen = 2;
          break;}
case (4): { if (selection1 == '1')
              CheckInStatus -> FamServCen = 1;
          else
              CheckInStatus -> FamServCen = 2;
          break;}
case (5): { if (selection1 == '1')
              CheckInStatus -> ClasMatCtrl = 1;
          else
              CheckInStatus -> ClasMatCtrl = 2;
          break;}
case (6): { if (selection1 == '1')
              CheckInStatus -> SCIF = 1;
          else
              CheckInStatus -> SCIF = 2;
          break;}
case (7): { if (selection1 == '1')
              CheckInStatus -> Library = 1;
          else
              CheckInStatus -> Library = 2;
          break;}
case (8): { if (selection1 == '1')
              CheckInStatus -> SecMgr = 1;
          else
              CheckInStatus -> SecMgr = 2;
          break;}
case (9): { if (selection1 == '1')
              CheckInStatus -> BOQ = 1;
          else
              CheckInStatus -> BOQ = 2;
          break;}
case (10): { if (selection1 == '1')
              CheckInStatus -> Dental = 1;
          else
              CheckInStatus -> Dental = 2;
          break;}
case (11): { if (selection1 == '1')
              CheckInStatus -> PSD = 1;
          else
```

```
                CheckInStatus -> PSD = 2;
            break;}
case (12): { if (selection1 == '1')
                CheckInStatus -> NEX = 1;
            else
                CheckInStatus -> NEX = 2;
            break;}
case (13): { if (selection1 == '1')
                CheckInStatus -> LicExam = 1;
            else
                CheckInStatus -> LicExam = 2;
            break;}
case (14): { if (selection1 == '1')
                CheckInStatus -> Registrar = 1;
            else
                CheckInStatus -> Registrar = 2;
            break;}
case (15): { if (selection1 == '1')
                CheckInStatus -> CompCtr = 1;
            else
                CheckInStatus -> CompCtr = 2;
            break;}
case (16): { if (selection1 == '1')
                CheckInStatus -> BaseSec = 1;
            else
                CheckInStatus -> BaseSec = 2;
            break;}
case (17): { if (selection1 == '1')
                CheckInStatus -> Medical = 1;
            else
                CheckInStatus -> Medical = 2;
            break;}
case (18): { if (selection1 == '1')
                CheckInStatus -> HRO = 1;
            else
                CheckInStatus -> HRO = 2;
            break;}
case (19): { if (selection1 == '1')
                CheckInStatus -> Admin = 1;
            else
                CheckInStatus -> Admin = 2;
            break;}
default:   {printf("\n\t\t\t   %s\n", invalidSelMsg[0]);
```

```
                printf("\n\t\t\t  <Press any key to continue>");
                Invalid = 1;
                break;}
    }
    SaveFileData("CI", 1);
    if ((selection1 == '1') && (Invalid != 1))
    {
        printf("\n\t\t\tCheck-In process complete!");
        printf("\n\t\t\t<Press any key to continue>");
    }
    if ((selection1 == '2') && (Invalid != 1))
    {
        printf("\n\t\t\tCheck-Out process complete!");
        printf("\n\t\t\t<Press any key to continue>");
    }
    getch();
}
```

```
/*************************************************************************
 * Function Name:     ShowAdminMenu()                                    *
 *                                                                       *
 * Description:       The following routine diplays the Card Administrator's *
 *                    selection menu for the demonstration program.      *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/

void ShowAdminMenu()
{
    int k; char selection;
    clrscr();

    for (k=0; k<11; k++)
        printf("%s\n", adminMenuMsg[k]);

    printf("%s", adminMenuMsg[11]);
    selection = getch();

    switch (selection)
    {
        case ('1'): {IssueDemoCard();
                     break;}
        case ('2'): {EraseDemoCard();
                     break;}
        case ('3'): {CheckIdCard();
                     break;}
        case ('4'): {ShowMainMenu();
                     break;}
        default:    {printf("\n\t\t\t   %s\n", invalidSelMsg[0]);
                     printf("\t\t\t   %s\n", pressAnyKeyMsg[0]);
                     getch();
                     ShowMainMenu();
                     break;}
    }
}
```

```
/*********************************************************************
 * Function Name:    EraseDemoCard()                                 *
 *                                                                   *
 * Description:      The following routine erases the contents of the smart card  *
 *                   chip for future re-use upon successful issuance of the issuer's *
 *                   key.                                            *
 *                                                                   *
 * History:          15 July 1994 (Created)                          *
 *                                                                   *
*********************************************************************/

void EraseDemoCard()
{
    short nError;
    BYTE ResetInfo[17];
    WORD ResetBuff;
    BYTE IssuerKeyNum;
    BYTE IssuerKeyLen;
    BYTE IssuerKey[9];
    nError = nWaitForCard();

    if (nError > 0)
    {
        DCOS_ResetChp(ResetInfo, &ResetBuff);
        printf("\n\t\t\tPlease submit the Issuer's Key >> ");
        scanf("%s", IssuerKey);
        IssuerKeyLen = strlen(IssuerKey);
        IssuerKeyNum = 1;

        DCOS_SubmitKy(IssuerKeyNum, IssuerKey, IssuerKeyLen);
        nError = DCOS_EraseChp();

        if (nError == NO_ERROR)
            printf("\n\t\t\tSmart Card Chip Erased!\n");
        else
            printf("\n\t\t\tError In Erasing Smart Card Chip!\n");

        getch();
        EjectCard();
    }
    ShowAdminMenu();
}
```

```
/*****************************************************************
 * Function Name:     CheckIdCard()                              *
*                                                                 *
* Description:       The following routine reads the ID section of the smart card,  *
*                    and displays its contents for viewing and verification.        *
*                                                                 *
* History:           15 July 1994  (Created)                    *
*                                                                 *
*****************************************************************/

void CheckIdCard()
{
    int Status;
    BYTE ResetInfo[45];
    WORD ResetBuffLen;
    ID_Info *Identification;
    Identification = (ID_Info *) FileData;
    Status = DCOS_Slct_CAU(1);

    if (Status != NO_ERROR)
        nWaitForCard();

    clrscr();
    DCOS_ResetChp(ResetInfo, &ResetBuffLen);
    GetFileData("ID");
    printf("\n\t\t\tName:      %s %s. %s \n", Identification->FName,
    Identification->MI, Identification->LName);
    printf("\t\t\tSSN:       %s \n",  Identification->SSN);
    printf("\t\t\tRank:      %s \n", Identification->Rank);
    printf("\t\t\tCurriculum: %s \n", Identification->Curric);
    printf("\t\t\tCurrCode:  %s \n", Identification->CurrCode);
    printf("\t\t\tBirthDate: %s \n", Identification->BDate);
    printf("\t\t\tTelephone: %s \n", Identification->TelNum);
    printf("\n\n\t\t\t %s", pressAnyKeyMsg[0]);
    getch();
    EjectCard();
    ShowAdminMenu();
}
```

```
/*************************************************************************
 * Function Name:     VerifyCheckInStatus()                              *
 *                                                                       *
 * Description:       The following routine reads and displays the contents of the *
 *                    check in/out section of the smart card memory for viewing *
 *                    and verification.                                  *
 *                                                                       *
 * History:           15 July 1994 (Created)                             *
 *                                                                       *
 *************************************************************************/
void VerifyCheckInStatus()
{
    int Status;
    BYTE ResetInfo[45];
    WORD ResetBuffLen;
    CheckInInfo *CheckInStatus;
    CheckInStatus = (CheckInInfo *) FileData;
    Status = DCOS_Slct_CAU(1);

    if (Status != NO_ERROR)
        nWaitForCard();
    clrscr();
    DCOS_ResetChp(ResetInfo, &ResetBuffLen);
    GetFileData("CI");
    printf("\n\t\t\tCURRENT CHECK IN/OUT STATUS\n");
    printf("\n\t\tCurricular Officer:        ");
    Status = CheckInStatus -> CurrOff;
    PrintStatus(Status);
    printf("\t\tService Representative:    ");
    Status = CheckInStatus -> ServRep;
    PrintStatus(Status);
    printf("\t\tStudent Mail Center:       ");
    Status = CheckInStatus -> StuMailCen;
    PrintStatus(Status);
    printf("\t\tFamily Services Center:    ");
    Status = CheckInStatus -> FamServCen;
    PrintStatus(Status);
    printf("\t\tClassified Material Control: ");
    Status = CheckInStatus -> ClasMatCtrl;
    PrintStatus(Status);
    printf("\t\tSCIF:                      ");
    Status = CheckInStatus -> SCIF;
    PrintStatus(Status);
```

```
    printf("\t\tLibrary:                ");
    Status = CheckInStatus -> Library;
    PrintStatus(Status);
    printf("\t\tSecurity Manager:       ");
    Status = CheckInStatus -> SecMgr;
    PrintStatus(Status);
    printf("\t\tBOQ Office:             ");
    Status = CheckInStatus -> BOQ;
    PrintStatus(Status);
    printf("\t\tDental:                 ");
    Status = CheckInStatus -> Dental;
    PrintStatus(Status);
    printf("\t\tPersonnel Support Detachment:");
    Status = CheckInStatus -> PSD;
    PrintStatus(Status);
    printf("\t\tNavy Exchange Admin Office: ");
    Status = CheckInStatus -> NEX;
    PrintStatus(Status);
    printf("\t\tLicensing Examiner:     ");
    Status = CheckInStatus -> LicExam;
    PrintStatus(Status);
    printf("\t\tRegistrar:              ");
    Status = CheckInStatus -> Registrar;
    PrintStatus(Status);
    printf("\t\tComputer Center:        ");
    Status = CheckInStatus -> CompCtr;
    PrintStatus(Status);
    printf("\t\tBase Security/Vehicle Reg: ");
    Status = CheckInStatus -> BaseSec;
    PrintStatus(Status);
    printf("\t\tMedical:                ");
    Status = CheckInStatus -> Medical;
    PrintStatus(Status);
    printf("\t\tHousing Referral Office:   ");
    Status = CheckInStatus -> HRO;
    PrintStatus(Status);
    printf("\t\tAdmin Services Office:    ");
    Status = CheckInStatus -> Admin;
    PrintStatus(Status);
    printf("\n\t\t\t<Press any key to continue> ");
}
```

```
/*****************************************************************************
 * Function Name:     PrintStatus(BYTE CheckInStatus)                        *
 *                                                                           *
 * Description:       The following routine prints the current status of the check *
 *                    in/out process.                                        *
 *                                                                           *
 * History:           15 July 1994  (Created)                                *
 *                                                                           *
 *****************************************************************************/

void PrintStatus(BYTE CheckInStatus)
{
    if (CheckInStatus == 0)
        printf("Not Checked In\n");
    if (CheckInStatus == 1)
        printf("Checked In\n");
    if (CheckInStatus == 2)
        printf("Checked Out\n");
}
```

```
/*************************************************************************
 * Function Name:     EjectCard()                                        *
 *                                                                       *
 * Description:       The following routine ejects the smart card from the Series 50  *
 *                    if it's still inside the unit.                     *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
 *************************************************************************/

void EjectCard()
{
    short Status;
    Status = DCOS_Slct_CAU (1);

    if (Status == NO_ERROR)
        DCOS_EjectCAU();
}

/*************************************************************************
 * Function Name:     ShowQuitMsg()                                      *
 *                                                                       *
 * Description:       The following routine displays the last message to the user  *
 *                    before exiting the demonstration program.          *
 *                                                                       *
 * History:           15 July 1994  (Created)                            *
 *                                                                       *
*************************************************************************/

void ShowQuitMsg()
{
    int k;
    clrscr();
    EjectCard();
    printf("\n");

    for (k=0; k<14; k++)
        printf("%s\n", quitMsg[k]);

    printf("%s", quitMsg[14]);
    getch();
}
```

```
/*****************************************************************
*          SMART CARD TECHNOLOGY DEMONSTRATION PROGRAM          *
*                        Naval Postgraduate School              *
*                        Monterey, California                   *
*                                                               *
* Module:       SMARTCRD.H                                      *
*                                                               *
* Description:  The following module contains the definitions, function *
*               prototypes, and all the required data structures for the *
*               smart card demonstration program.               *
*                                                               *
* History:      15 July 1994  (Created)                         *
*                                                               *
* Programmer:   LT Rudy G. Dollete, USN   Internet: dollete@aol.com *
*                                                               *
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <string.h>
#include <mem.h>
#include <float.h>
#include <math.h>
#include <c:\devkit\include\dcosdefs.h>
#include <c:\devkit\include\dcosptyp.h>
#define NUM 1
#define ESC_KEY 0x1B

typedef struct {
     BYTE LName[10];
     BYTE FName[10];
     BYTE MI[2];
     BYTE SSN[10];
     BYTE Rank[5];
     BYTE Curric[4];
     BYTE CurrCode[3];
     BYTE BDate[7];
     BYTE TelNum[8];
     } ID_Info;
```

```
typedef struct {
    BYTE Type;
    BYTE Date[3];
    DWORD Amount;
    } TransActInfo;

typedef struct {
    BYTE CurrOff;
    BYTE ServRep;
    BYTE StuMailCen;
    BYTE FamServCen;
    BYTE ClasMatCtrl;
    BYTE SCIF;
    BYTE Library;
    BYTE SecMgr;
    BYTE BOQ;
    BYTE Dental;
    BYTE PSD;
    BYTE NEX;
    BYTE LicExam;
    BYTE Registrar;
    BYTE CompCtr;
    BYTE BaseSec;
    BYTE Medical;
    BYTE HRO;
    BYTE Admin;
    } CheckInInfo;



    WORD FileAddress;
    BYTE FileData[512];

    int CreateFileDef ( BYTE FileNum, BYTE Mode, BYTE Security, BYTE RecNum,
                                                BYTE RecLen, BYTE *FileName);

    int SaveFileData (BYTE *FileName, BYTE FileType);
    int SaveTransAction(TransActInfo *Transaction);
    void IssueStudentCard(void);
    void EraseDemoCard(void);
    short nWaitForCard(void);
    void ShowVersionMsg(void);
```

```
void ShowMainMenu(void);
short InitializeCommPort(void);
void ShowPurseMenu(void);
void ViewBalance(void);
void GetFileData(BYTE *FileName);
void ShowCheckInOutMenu(void);
void ShowCheckSelect(char selection);
void ShowAdminMenu(void);
void CheckIdCard(void);
void ShowQuitMsg(void);
void EjectCard(void);
void DisplayAmount(DWORD Balance, BYTE MaxDigits);
void PurchaseAmount(void);
void DepositAmount(void);
long GetAmount(int DspFlag);
void DspAmount(long Amount);
void ViewTransAction(void);
void CreateUserID(void);
void CreateCheckInFile(void);
void VerifyCheckInStatus(void);
void PrintStatus(BYTE CheckInStatus);
```

```
/***********************************************************************
*          SMART CARD TECHNOLOGY DEMONSTRATION PROGRAM                 *
*                    Naval Postgraduate School                         *
*                      Monterey, California                            *
*                                                                      *
* Module:          MESSAGES.C                                          *
*                                                                      *
* Description:     The following module contains the NPS Smart Card    *
*                  demonstration program display messages.             *
*                                                                      *
* History:         15 July 1994  (Created)                             *
*                                                                      *
* Programmer:      LT Rudy G. Dollete, USN    Internet: dollete@aol.com *
*                                                                      *
***********************************************************************/

#include "messages.h"

char *versionMsg[20] ={
    "\n",
    "\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»",
    "\t  º                                                   º",
    "\t  º            MULTIPLE APPLICATION                   º",
    "\t  º        IDENTIFICATION CARD PROGRAM                º",
    "\t  º                  --------                         º",
    "\t  º                                                   º",
    "\t  º                 Version 1.0                       º",
    "\t  º                                                   º",
    "\t  º    A Demonstration of Smart Card Technology.      º",
    "\t  º                                                   º",
    "\t  º             LT RUDY G. DOLLETE                    º",
    "\t  º               (programmer)                        º",
    "\t  º                                                   º",
    "\t  º               ROGER STEMP                         º",
    "\t  º                (adviser)                          º",
    "\t  º                                                   º",
    "\t  ÈÍÍÍÍÍÍÍÍÍ * <Press any key to continue> * ÍÍÍÍÍÍÍÍÍ¿  ",
    "\t  ÛÛÛÛÛÛÛÛÛÛ2222222222¿¿¿¿********¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
    "\t                  "};
```

```
char *mainMenuMsg[16] ={
      "\n\t              SMART CARD TRANSACTION MENU                         ",
      "\t   Éíííííííííííííííííííííííííííííííííííííííííííííííííí»                  ",
      "\t   º                                                          º",
      "\t   º         <1> ELECTRONIC PURSE                             º",
      "\t   º                                                          º",
      "\t   º         <2> STUDENT CHECK IN/OUT                         º",
      "\t   º                                                          º",
      "\t   º         <3> CARD ADMINISTRATOR                           º",
      "\t   º                                                          º",
      "\t   º         <4> EJECT SMART CARD                             º",
      "\t   º                                                          º",
      "\t   º         <5> QUIT APPLICATION                             º",
      "\t   º                                                          º",
      "\t   Èíííííííííí * <Enter your selection> * íííííííííí¿               ",
      "\t   ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿••••••••¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
      "\t                  "};


char *purseMenuMsg[16] ={
      "\n\t                ELECTRONIC PURSE MENU                             ",
      "\t   Éíííííííííííííííííííííííííííííííííííííííííííííííííí»                  ",
      "\t   º                                                          º",
      "\t   º         <1> PURCHASE                                     º",
      "\t   º                                                          º",
      "\t   º         <2> DEPOSIT                                      º",
      "\t   º                                                          º",
      "\t   º         <3> VIEW BALANCE                                 º",
      "\t   º                                                          º",
      "\t   º         <4> VIEW TRANSACTIONS                            º",
      "\t   º                                                          º",
      "\t   º         <5> RETURN TO MAIN MENU                          º",
      "\t   º                                                          º",
      "\t   Èíííííííííí * <Enter your selection> * íííííííííí¿               ",
      "\t   ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿••••••••¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
      "\t                  "};
```

```
char *checkInOutMsg[14] ={
	"\n\t              CHECK IN / CHECK OUT MENU                         ",
	"\t   Éíííííííííííííííííííííííííííííííííííííííííííííííí»                  ",
	"\t   º                                                               º",
	"\t   º       <1> CHECK IN                                            º",
	"\t   º                                                               º",
	"\t   º       <2> CHECK OUT                                           º",
	"\t   º                                                               º",
	"\t   º       <3> VIEW CHECK IN/OUT STATUS                            º",
	"\t   º                                                               º",
	"\t   º       <4> RETURN TO MAIN MENU                                 º",
	"\t   º                                                               º",
	"\t   Èííííííííííí * <Enter your selection> * íííííííííí¿               ",
	"\t   ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿""""""""¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
	"\t                  "};

char *checkInMsg[25] ={
	"\n\t              NPS CHECK IN/OUT MENU                             ",
	"\t   Éíííííííííííííííííííííííííííííííííííííííííííííííí»                  ",
	"\t   º <1> CURRICULAR OFFICER       <11> PSD                         º",
	"\t   º                                                               º",
	"\t   º <2> SERVICE REP              <12> NAVY EXCHANGE               º",
	"\t   º                                                               º",
	"\t   º <3> STUDENT MAIL CENTER      <13> LICENSINGD                  º",
	"\t   º                                                               º",
	"\t   º <4> FAMILY SERVICE CENTER <14> REGISTRAR                      º",
	"\t   º                                                               º",
	"\t   º <5> CLASSIFIED MATERIAL      <15> COMPUTER CENTER             º",
	"\t   º                                                               º",
	"\t   º <6> SCIF                     <16> BASE SECURITY               º",
	"\t   º                                                               º",
	"\t   º <7> LIBRARY                  <17> MEDICAL                     º",
	"\t   º                                                               º",
	"\t   º <8> SECURITY MANAGER         <18> HOUSING                     º",
	"\t   º                                                               º",
	"\t   º <9> BOQ OFFICE               <19> ADMIN                       º",
	"\t   º                                                               º",
	"\t   º <10> DENTAL                  <20> QUIT                        º",
	"\t   º                                                               º",
	"\t   Èííííííííííí * <Enter your selection> * íííííííííí¿               ",
```

```
char *adminMenuMsg[14] ={
     "\n\t              CARD ADMINISTRATOR'S MENU                        ",
     "\t   Éííííííííííííííííííííííííííííííííííííííííííííííííííí»                         ",
     "\t   º                                                             º",
     "\t   º        <1> ISSUE SMART CARD                                 º",
     "\t   º                                                             º",
     "\t   º        <2> ERASE SMART CARD                                 º",
     "\t   º                                                             º",
     "\t   º        <3> VERIFY CARDHOLDER'S ID                           º",
     "\t   º                                                             º",
     "\t   º        <4> RETURN TO MAIN MENU                              º",
     "\t   º                                                             º",
     "\t   Èíííííííííí * <Enter your selection> * íííííííííí¿               ",
     "\t   ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿""""""""¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
     "\t                 "};


char *quitMsg[15] ={
     "\n",
     "\t   Éííííííííííííííííííííííííííííííííííííííííííííííííííí»                         ",
     "\t   º                                                             º",
     "\t   º           THANK YOU FOR USING                               º",
     "\t   º   THE NPS SMART CARD DEMONSTRATION PROGRAM                  º",
     "\t   º                                                             º",
     "\t   º     SUGGESTIONS AND COMMENTS ABOUT THIS                     º",
     "\t   º           PROGRAM ARE WELCOME.                              º",
     "\t   º                                                             º",
     "\t   º        Roger Stemp:  (408) 656-2395                         º",
     "\t   º        Rudy Dollete:  dollete@aol.com                       º",
     "\t   º                                                             º",
     "\t   Èíííííííííí ** <Press any key to quit> ** íííííííííí¿             ",
     "\t   ÛÛÛÛÛÛÛÛÛÛ222222222¿¿¿¿""""""""¿¿¿¿2222222222ÛÛÛÛÛÛÛÛÛÛ ",
     "\t                 "};

char *insertCardMsg[1]  = {"Insert smart card into Series 50."};
char *pressEscMsg[1]    = {"Press <ESC> to exit."};
char *pressAnyKeyMsg[1] = {"<Press Any Key To Continue>"};
char *invalidSelMsg[1]  = {"Your Selection Is Invalid."};
char *cursorMsg[1]      = {"                "};
char *transActMsg[2]    = {"\t\tDate\t\t\tAmount\t\t\tType\n",

"\t\t================================================\n"};
```

```
/*****************************************************************************
*                  SMART CARD TECHNOLOGY DEMONSTRATION PROGRAM              *
*                                Naval Postgraduate School                  *
*                                Monterey, California                       *
*                                                                           *
* Module:          MESSAGES.H                                               *
*                                                                           *
* Description:     The following module contains the declarations for the NPS *
*                  Smart Card demonstration program display messages.       *
*                                                                           *
* History:         15 July 1994  (Created)                                  *
*                                                                           *
* Programmer:      LT Rudy G. Dollete, USN    Internet: dollete@aol.com     *
*                                                                           *
*****************************************************************************/

    extern char *versionMsg[20];
    extern char *mainMenuMsg[16];
    extern char *purseMenuMsg[16];
    extern char *checkInOutMsg[14];
    extern char *checkInMsg[25];
    extern char *adminMenuMsg[14];
    extern char *quitMsg[15];
    extern char *insertCardMsg[1];
    extern char *pressEscMsg[1];
    extern char *pressAnyKeyMsg[1];
    extern char *invalidSelMsg[1];
    extern char *cursorMsg[1];
    extern char *transActMsg[2];
```

LIST OF REFERENCES

[Acad94] Academic American Encyclopedia, Vol. 5, 1994, p. 160a.

[Anth92] Anthes, G. H., "Food Stamp Program Soon To Be Electronic," *Computer World*, vol. 26, pp. 152-152, 13 January 1992.

[Ayre94] Ayres, G., "The DoD Experience," *CardTech Conference*, April 1994.

[Blac93] Blackburn, M., "Access Technology: Distributive vs. On-line Processing," *NACAS*, April 1993.

[Byce65] Bycer, B. B., *Digital Magnetic Tape Recording: Principles And Computer Applications*, Hayden Book Company, Inc., New York, 1965.

[Data92] Smart Card Programmer's Reference Manual (680-IC), DataCard Corporation, Minnetonka, MN, 1992.

[Hono94] Honold, F., "The Advantages Of Contactless Cards," *Smart Card Technology International*, London, 1994.

[Klas93] Klass, S., " An Integrated On-line/Off-line Cashless Campus Model," *NACAS*, June 1993.

[Line93] Lineback, R. J., "Siemens, Phillips Join Smart Card Project," *Electronic News*, Vol. 39, p. 19, 8 February 1993.

[McGe92] McGeary, T. C., "The History Of Magnetic Stripe Technology And Its Applications," *Conference Proceedings - CardTech '92*, 8 April 1992.

[Mill94] Miller, B. L., "Biometric Identification: The Power to Protect People, Places and Privacy," *Conference Proceedings - CardTech '94*, 10 April 1994.

[Mill94] Miller, B., "Smart Cards In The United States: The Sleeper Is Awakening," *Security Technology & Design*, Vol. 4, No. 3, pp. 22-25, April 1994.

[Ogni93] Ognibene, P. J., "Smart Cards Could Save Lives And Dollars: A Computerized Personal Medical Record Would Assist Doctors And Pharmacists And Avert Dangerous Errors," *Los Angeles Times*, Vol. 112, p. B7, 12 April 1993.

[Pear67] Pear, C. B., *Magnetic Recording*, Reinhold Publishing Corporation, New York, 1967.

[Pemb94] Pemberton, J., "Contactless Cards - The Solution To All The Problems?," *Smart Card Technology International*, London, 1994.

[Pfle89] Pfleeger, C. P., *Security In Computing*, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[Powe93] Power, K., "Smart Cards And Public Keys Unlock Crypto's Potential Uses; More Powerful Microchips Let Agencies Move Away From Simple Passwords," *Government Computer News*, Vol. 12, p. 75, 1 February 1993.

[Shep92] Sheppard, J. J., "Magnetic Stripe Applications: Where and Why," *Conference Proceedings - CardTech '92*, 8 April 1992.

[Stan93] Stanford, C. J., "What Is A Smart Card?," Cardware Ltd, 1993.

[Tcha93] Tchashchin, K., " Russia - Smart Cards Arrive," *Newsbytes*, 8 March 1993.

[Svig87] Svigals, J., *Smart Cards*, MacMillan Publishing Company, New York, 1987.

[Town93] Townend, R., "The Value of Collecting Plastic," *Conference Proceedings - CardTech '93*, 18 April 1993.

[Watt92] Watts, A., "Vision For The Future Smartcard IC's, " *SGS-THOMSON Microelectronics*, September 1992.

[Will94] William, J. T., "Challenges In Implementing EBT: View from Different Stages of Development," *Conference Proceedings - CardTech '94*, 10 April 1994.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Dr. Ted Lewis, Code CS/Lt — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5002

4. Dr. Timothy Shimeall, Code CS/SM — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5002

5. Roger Stemp, Code CS/SP — 1
   Computer Science Department, Code CS/ST
   Naval Postgraduate School
   Monterey, CA 93943-5002

6. Lieutenant Rodolfo G. Dollete — 2
   1198 Sunbright Drive,
   Oceanside, CA 92056

7. Lieutenant Leslie Bower — 1
   R.D. #1, Box 487-B
   Beech Creek, PA 16822

8. Commander — 1
   Naval Computer and Telecommunication Command
   4401 Massachusetts Ave., N. W.
   Washington, DC 20394-5000

9. Defense Information Systems Agency — 1
   TFEF
   3701 North Fairfax Drive
   Arlington, VA 22203-1713

10. National Institute of Standards and Technology (NIST) 1
Computer Security Division/Computer Systems Laboratory
Gaithersburg, MD 20899